VERIFIER–HARNESS CO-EVOLUTION FOR TRAINING-FREE SELF-IMPROVEMENT ON OPEN-ENDED TASKS

Anonymous authors
Paper under double-blind review

ABSTRACT

Training-free self-improvement lets an agent use feedback to revise the response it is about to return and to update the harness state that shapes how it handles later tasks. On open-ended tasks that feedback rarely comes from reference answers or executable tests, so the agent has to rely on a model-based verifier. When the verifier is wrong, the current response can get worse, and misleading experience can persist across the task stream. A verifier that never changes also grows stale as the agent's responses improve and task demands shift. We introduce *Unified Co-Evolution* (UCoE), a training-free method that evolves an executable verifier program together with the agent's harness state while all model parameters stay fixed. UCoE synthesizes candidate verifier programs and tests them on response contrasts, pairs of responses that differ in one declared property and have a known preferred side. It selects the candidate whose discrimination on its weakest contrast family is highest, a quantity we call *driving capacity*. The selected verifier diagnoses one weakness in each draft, guides a single repair, decides which response is served, and determines whether the accepted repair may enter the harness state. As tasks complete, their diagnostics seed new candidate verifiers, which must beat the current verifier on the same contrasts before adoption. Across six open-ended benchmarks, UCoE averages 59.00% preference against matched direct generation with Qwen3-8B and 57.19% with Qwen3.8-27B, margins of 9.00 and 7.19 percentage points. On three verifiable benchmarks, the same method raises the mean from 36.06% for direct generation to 38.48%.

1 INTRODUCTION

Self-improving agents use feedback in two ways. They revise the response they are about to return (Madaan et al., 2023), and they retain experience that shapes how they handle later tasks (Shinn et al., 2023; Zhao et al., 2024). Recent work extends the second use to the prompt-level harness that surrounds the model (Lee et al., 2026a). In mathematics and programming, exact answers or executable tests can ground this feedback, as in Figure 1a. Open-ended tasks offer no such anchor. As Figure 1b shows, the agent sees only the public requirements of a request and must judge correctness, coverage, relevance, and style on its own. Because the same judgment decides which response is served and which experience enters the harness state, a single mistaken judgment can harm the current task and then persist across the stream.

Existing methods obtain open-ended feedback from holistic LLM judges, natural-language rubrics, executable reward programs, or learned verifiers. A holistic judge gives a convenient overall score or preference, but the contribution of each quality dimension stays implicit (Gunjal et al., 2026; Shen et al., 2026), and judge accuracy alone is not a reliable predictor of downstream selection gain (Lu et al., 2026). Rubrics make those dimensions explicit, yet generated criteria can be incomplete, conflated, or redundant (Shen et al., 2026), so an aggregate score can still hide a blind spot. Executable reward programs expose the evaluation path (Anonymous, 2026b; Li et al., 2026), but explicit structure does not guarantee behavior, since the selector a program induces still has to be tested across failure families. Training-based methods learn generative reward models, rubric generators, rubric-conditioned judges, or verifier capabilities within the policy (Jia et al., 2025; Xu et al., 2026a; Chen et al., 2026; Pan et al., 2026), at the cost of a model-training stage and its compute. Training-free pipelines that use prompted rubric-guided refinement or aggregate several aspect judgments (Wan et al., 2026; Lifshitz et al., 2025) avoid that cost but keep the verifier fixed during

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.

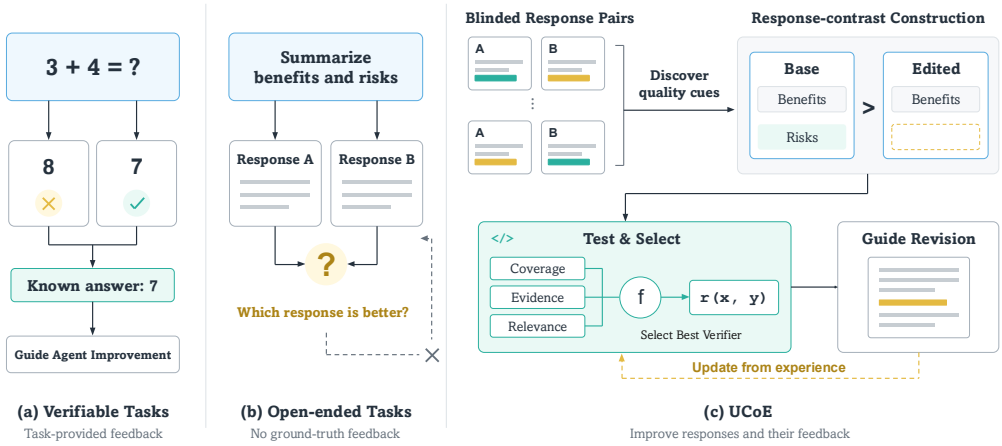


Figure 1: **Feedback for open-ended self-improvement.** (a) On verifiable tasks, a known answer marks each response right or wrong and guides agent improvement. (b) On open-ended tasks there is no ground truth, so the agent must judge for itself which response is better, and a wrong judgment harms both the served response and the experience kept for later tasks. (c) UCoE mines quality cues from blinded response pairs, builds response contrasts in which a base response is preferred over an edited copy that lacks one property, tests candidate verifier programs on them, and uses the selected verifier to guide revision while experience from completed tasks updates it over time.

deployment, where it can become stale as responses improve and task demands shift (Anonymous, 2026b; Li et al., 2026; Iacob et al., 2026).

To address these weaknesses, we introduce *Unified Co-Evolution* (UCoE), a training-free method that evolves an executable verifier program alongside the harness state with model parameters fixed. The program keeps every criterion explicit, contrast testing measures its behavior on each failure family, and evolution during deployment keeps it from going stale. UCoE synthesizes candidate verifier programs and tests them on response contrasts, pairs of responses that differ in one targeted property with a declared preference (Gardner et al., 2020; Sai et al., 2021), grouped by property into families. It selects the candidate with the highest *driving capacity*, its discrimination on the family where it is weakest, so that strength on easy properties cannot mask a blind spot. During deployment the selected verifier diagnoses one weakness in the draft and guides one local repair. It then chooses between draft and revision and decides whether the repair may enter the harness state. Diagnostics from completed tasks seed new candidates, which replace the current verifier only after meeting the update criteria on the same contrasts. Figure 1c sketches this loop.

We evaluate UCoE on six open-ended benchmarks and three benchmarks with verifiable endpoints using Qwen3-8B and Qwen3.8-27B actors. UCoE has the highest preference against matched direct generation of all compared feedback methods on every open-ended benchmark at both actor sizes, averaging 59.00% and 57.19%, and it improves every available native metric. On the verifiable tasks it raises the mean from 36.06% to 38.48% without a loss on any benchmark. Ablations attribute most of the gain to current-task repair, with verifier selection, verifier evolution, and retained procedures each adding to it. Our contributions are:

- We introduce UCoE, a training-free method that evolves an executable verifier program with the harness state, so feedback keeps pace with the agent it guides.
- We propose driving capacity, a verifier's discrimination on its weakest contrast family, and show that it bounds achievable selection gain, giving a safe criterion for verifier updates.
- We show state-of-the-art training-free self-improvement on nine benchmarks, with gains that carry over to GPT-5.4 actors and a Codex agent harness on SpreadsheetBench-2.

2 Related Work

Training-free self-improvement. Training-free methods improve the current response or retain non-parametric state that shapes future behavior. Self-Refine revises responses within a task (Madaan

et al., 2023), while Reflexion, ExpeL, Dynamic Cheatsheet, and A-MEM retain or organize reusable experience across trials and tasks (Shinn et al., 2023; Zhao et al., 2024; Suzgun et al., 2026; Xu et al., 2025). ACE and Recursive Harness Self-Improvement extend adaptation to the agent's context or prompt-level harness (Zhang et al., 2026a; Lee et al., 2026a). Test-time scaling allocates additional inference computation to sampling and verification, including in agentic workflows (Snell et al., 2025; Zhu et al., 2025). All of these methods take the feedback or selection process as given. UCoE adapts that process too: its verifier program evolves alongside the harness state, and together they determine which response is served and which experience is kept.

Open-ended rewards and verification. For tasks without exact answers, prior work constructs model-based evaluation signals from generated rubrics, rubric anchors, recursively decomposed and reweighted criteria, or pairwise generative rewards (Gunjal et al., 2026; Huang et al., 2025; Shen et al., 2026; Jia et al., 2025). Other methods combine supplied verifiers, elicit rubric-guided or aspect-specific judgments, or construct self-verifiable proxy tasks (Lee et al., 2026b; Wan et al., 2026; Lifshitz et al., 2025; Wang et al., 2026). CoNL evaluates critiques by the improvements they induce (Sui & Hooi, 2026), and Eureka expresses feedback as executable reward code for simulated control (Ma et al., 2024). UCoE builds on this line but compares whole verifier programs, criteria and aggregation included, on response contrasts with declared preference directions, and it lets the diagnostics of completed tasks motivate later changes to the program.

Co-evolution. Prior co-evolution methods update feedback mechanisms alongside the systems they guide. Rubric-centered work co-evolves rubrics with responses or research policies, adjusts rubric complexity during training, or jointly learns rubric generators, judges, and verifier capabilities (Guan et al., 2026; Shao et al., 2026; Anonymous, 2026a; Xu et al., 2026a; Chen et al., 2026; Pan et al., 2026). DREAM-RL and EvoRS evolve executable Reward-DAGs from rollout diagnostics or on-policy response traces (Anonymous, 2026b; Li et al., 2026). Other work pairs task generators with solvers, compares changing evaluators with fixed anchors, adapts multi-agent organization at test time, or evolves programmatic annotators from labeled validation data (Kwan et al., 2026; Iacob et al., 2026; Zhang et al., 2026b; Xu et al., 2026b). Most of these methods rely on parameter training, on-policy reward traces, or labeled anchors. UCoE keeps the co-evolutionary view but moves all adaptation outside the model parameters. The criteria and aggregation of an executable verifier program change together with the agent's harness state, diagnostics from completed tasks suggest those changes, and response contrasts with declared preferences give a common basis for judging them.

3 METHOD

We introduce UCoE in this section. Figure 2 gives an overview of the method. See App. B for the complete algorithm.

3.1 PROBLEM SETUP

Let $x_1, \ldots, x_T$ be an ordered task stream and F_{θ_0} a fixed actor. Each method $\mathcal{A}$ processes the same stream with its own harness state H_t, and no method changes any model parameter. What H_t contains, and how the actor reads it, depends on the agent system. It may hold instructions, plans, tools, skills, or memory that shape later behavior. In the main experiments it is a bounded procedural memory, and every adaptive method receives an isolated instance that starts empty and follows the same capacity, retrieval, and update rules.

All methods follow the same per-task lifecycle. Given H_t, the actor first generates a draft $y_t^0 = F_{\theta_0}(x_t; H_t)$. The method then applies its own feedback and selection rules, which may produce at most one revision y_t^R, and returns $y_t \in \{y_t^0, y_t^R\}$. Only after y_t has been returned does the method apply its harness update $H_{t+1} = U_{\mathcal{A}}(H_t, \tau_t^{\mathcal{A}})$, where $\tau_t^{\mathcal{A}}$ is the completed task trace. This ordering lets feedback revise the current response while task-derived state affects only later tasks.

Let $\mathcal{I} \subseteq \{1, \ldots, T\}$ denote the tasks included in final evaluation. We score each method prequentially:

$$S_{\text{preq}}(\mathcal{A}) = \frac{1}{|\mathcal{I}|} \sum_{t \in \mathcal{I}} Q_t(x_t, y_t), \qquad \theta_t = \theta_0, \tag{1}$$

where Q_t is the final evaluator's score for task t. Throughout the stream, methods see public task requirements and their own model-generated artifacts, never benchmark answers or final-evaluator

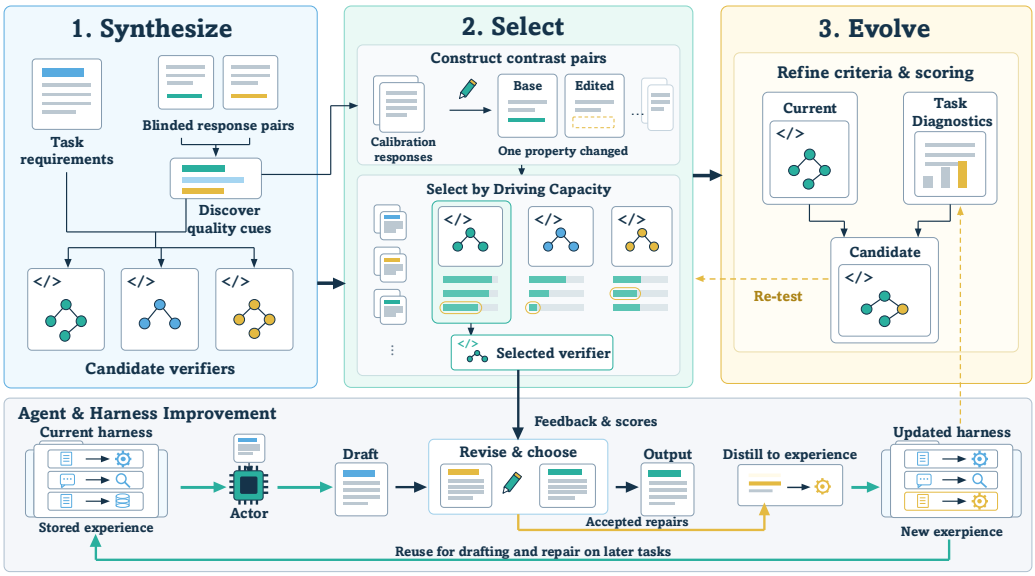


Figure 2: **UCoE workflow.** Top, the verifier side: (1) Synthesize proposes candidate verifier programs from the task requirements and from quality cues discovered in blinded response pairs. (2) Select edits calibration responses one property at a time to form response contrasts and deploys the candidate that is strongest on its weakest family. (3) Evolve combines the current verifier with diagnostics from completed tasks into a refined candidate, which is re-tested on the same contrasts before it can replace the current verifier. Bottom, the agent side: the fixed actor drafts with the current harness state, the deployed verifier guides one revision and chooses between draft and revision, and accepted repairs are distilled into experience that the updated harness state reuses on later tasks.

feedback. The score is prequential because each y_t is produced from the state inherited from earlier tasks, before task t contributes to H_{t+1}.

3.2 Verifier synthesis and contrast-based selection

Initial verifier synthesis. UCoE represents a verifier as a program that maps a task x and a response y to a vector of criterion scores and a scalar reward:

$$\mathcal{V} = (\Phi, A, D), \qquad \mathbf{s}_{\mathcal{V}}(x, y) = \big(\phi_1(x, y), \ldots, \phi_M(x, y)\big), \qquad r_{\mathcal{V}}(x, y) = A(\mathbf{s}_{\mathcal{V}}(x, y)), \tag{2}$$

where Φ is the set of criterion scorers, A combines their scores into a reward in which higher is better, and D turns the weakest participating criterion into repair feedback. Semantic criteria are scored by a fixed language model that we call the verifier model, which also proposes programs and diagnoses weaknesses. Evolution may add criteria to Φ and change A. The diagnosis procedure, the program interpreter, and the parameters of the verifier model never change.

Public task metadata states broad requirements but can miss domain-specific qualities that recur in real responses. UCoE therefore begins with relation discovery. On a separate development set, the actor and a larger frozen comparison model answer the same prompts, and the verifier model reads each pair in random order with system identities hidden and turns differences that recur across prompts into general criteria together with targeted edits that later serve contrast construction. These fixed relations form $\mathcal{R}$. Using the task metadata, the shared criteria, and $\mathcal{R}$, the verifier model then proposes candidate programs. Each candidate specifies how to combine the criterion scores and may add up to two criteria of its own. A lightweight validity check removes malformed or non-executable programs, and the surviving pool $\mathcal{K}_0$ is scored on separate response contrasts, where driving capacity picks the initial verifier.

Response-contrast construction. Candidate verifiers need a common behavioral test that shows whether their scores respond to intended changes in quality and which candidate tracks those changes most reliably. Arbitrary response pairs differ in many ways at once, so UCoE instead constructs controlled contrasts that isolate one property at a time (Gardner et al., 2020; Sai et al., 2021). Each contrast starts from a calibration response, written by the actor for a public prompt that is disjoint

from deployment, and applies one targeted edit that breaks a single property while leaving everything else in place. The base response y^+ and the degraded response y^- then differ only in that property, and the declared preference for y^+ concerns that property alone. Three general families cover omitted required content, unsupported claims, and irrelevant material. Relation discovery adds benchmark-specific families under the same rule: a relation is kept only if it recurs in at least three development prompts and can be expressed as one binary criterion with one edit that breaks only that property. The resulting pairs form the response contrasts $\mathcal{B}$ and are split by source before any verifier is scored. The selection split ranks candidates, and the held-out split later compares a candidate with the current verifier. See App. D.5 for construction details.

Driving capacity and selection. Static validation only establishes that a program runs. Choosing among valid programs at random ignores how they behave on the contrasts, and ranking them by mean accuracy lets strong performance on easy families hide a blind spot on a hard one. UCoE therefore ranks candidates by their weakest family.

Each verifier $\mathcal{V}$ induces a strict-margin selector $h_{\mathcal{V}} = h_{\mathcal{V},\epsilon}$ on each contrast, which picks y^+ over y^- only when the reward of y^+ exceeds that of y^- by more than ϵ. A tie or a sub-margin lead counts as a failure. For a contrast family j, let $p_j(h)$ be the probability that selector h picks y^+, and define its centered accuracy as

$$J_j(h) = 2p_j(h) - 1 = 2\Pr[h(x, y^+, y^-) = y^+ \mid j] - 1. \tag{3}$$

Chance-level selection gives $J_j = 0$, and positive or negative values mean above- or below-chance selection. The signed driving capacity of $\mathcal{V}$ on the response contrasts $\mathcal{B}$ is one quarter of its worst family value, and the nonnegative driving capacity clips it at zero:

$$\widetilde{\mathcal{C}}_{\mathcal{B}}(\mathcal{V}) \coloneqq \frac{1}{4} \min_j J_j(h_{\mathcal{V}}), \qquad \mathcal{C}_{\mathcal{B}}(\mathcal{V}) \coloneqq \max\{0, \widetilde{\mathcal{C}}_{\mathcal{B}}(\mathcal{V})\}. \tag{4}$$

The factor of one quarter gives the quantity a covariance meaning. For valid contrasts, if one of the two response positions is drawn uniformly at random, $J_j/4$ is the covariance between the selector choosing that position and that position holding the targeted property, so driving capacity is the smallest such selection covariance over the tested families. See Proposition 1 in App. C.1 for the precise statement.

The minimum also has a robustness meaning. Suppose the m tested families are mixed with weights $w \in \Delta_m = \{w : w_j \geq 0, \sum_j w_j = 1\}$, so that the selector succeeds with probability $p_w(h_{\mathcal{V}}) = \sum_j w_j p_j(h_{\mathcal{V}})$. The least favorable mixture puts all of its mass on a weakest family, and Theorem 1 in App. C.2 shows that $\inf_{w \in \Delta_m} p_w(h_{\mathcal{V}}) = \min_j p_j(h_{\mathcal{V}}) = 1/2 + 2\widetilde{\mathcal{C}}_{\mathcal{B}}(\mathcal{V})$. Maximizing signed driving capacity therefore maximizes the worst-case probability of selecting the preferred response over every mixture of the tested families. UCoE optimizes the signed value rather than the clipped one because clipping would make all below-chance candidates look alike. Its empirical objective at checkpoint k, with $k = 0$ denoting initialization, is

$$\widehat{\mathcal{V}} \in \arg\max_{\mathcal{V} \in \mathcal{K}_k} \min_j \widehat{J}_j(h_{\mathcal{V}}), \tag{5}$$

where $\mathcal{K}_k$ is the candidate pool at that checkpoint and $\widehat{J}_j$ is centered accuracy on the selection split. Proposition 3 bounds how far the empirical maximizer can fall short of the best candidate in the pool. The winner from $\mathcal{K}_0$ becomes the initial verifier $\mathcal{V}_0$. Later pools also contain the current verifier, and their winner must meet the update criteria of Section 3.3 before it replaces the current verifier.

Driving capacity measures discrimination, and discrimination pays off only when the compared responses differ in quality. Let $Z_i \in \{0, 1\}$ mark whether response i of a pair is adequate under the final evaluation, let $S_h \in \{1, 2\}$ be the index that h selects, and let $q = \Pr(Z_1 \neq Z_2)$ be the fraction of pairs on which the choice matters. For $q > 0$, the mixed-pair centered accuracy $J_{\text{mix}}(h) = 2\Pr(Z_{S_h} = 1 \mid Z_1 \neq Z_2) - 1$ measures how often the selector picks the adequate response when the two differ. Pairs of equal adequacy contribute nothing, so for two selectors h and h' on the same fixed pairs,

$$\mathbb{E}[Z_{S_{h'}} - Z_{S_h}] = \frac{q}{2}\{J_{\text{mix}}(h') - J_{\text{mix}}(h)\}. \tag{6}$$

Proposition 2 records this decomposition: the actor supplies the opportunity q, the selector supplies the ability J_{mix} to use it, and when $q = 0$ no selector can change expected adequacy. Whether gains on constructed contrasts carry over to the actor's natural revisions depends on how well the contrast families represent the errors those revisions contain.

3.3 RESPONSE REVISION AND CO-EVOLUTION

Response revision and selection. Let $\mathcal{V}_k$ be the verifier active at task t. It scores the draft y_t^0, and from that fixed score vector and its weakest participating criterion the diagnosis procedure returns a target criterion c_t, localized evidence w_t, and a repair instruction u_t, written $(c_t, w_t, u_t) = D_{\mathcal{V}_k}(x_t, y_t^0)$. Conditioned on this feedback and on H_t, the fixed actor makes one local repair while preserving unrelated content:

$$y_t^R = F_{\theta_0}(x_t, y_t^0, c_t, w_t, u_t; H_t). \tag{7}$$

The same $\mathcal{V}_k$ then scores the draft and the revision, and the revision is served only when its reward lead exceeds a margin $\epsilon \geq 0$:

$$y_t = \begin{cases} y_t^R, & r_{\mathcal{V}_k}(x_t, y_t^R) > r_{\mathcal{V}_k}(x_t, y_t^0) + \epsilon, \\ y_t^0, & \text{otherwise.} \end{cases} \tag{8}$$

Whenever the comparison cannot be made, because the feedback was unusable, the revision is missing, or a score is invalid, and whenever the lead is a tie or falls below the margin, the draft is served. We write $a_t \in \{0, 1\}$ for this acceptance decision.

Harness update. After the response is returned, a system-specific operator updates the harness state:

$$H_{t+1} = \mathrm{UpdateHarness}\big(H_t, x_t, y_t^0, y_t^R, c_t, w_t, a_t\big). \tag{9}$$

Only an accepted repair, $a_t = 1$, may add task-derived content to H_{t+1}. In our bounded procedural-memory implementation, an accepted revision yields at most one generalized trigger–action procedure once task-specific content has been removed, and a rejected revision may update metadata only. See App. D.2 for the retrieval, deduplication, and eviction rules.

Verifier evolution. As task demands and H_t change, the draft and revision pairs that the actor produces can drift away from the contrasts on which the initial verifier was selected, and its criteria or aggregation can go stale. Equation (6) makes the cost concrete: the selection gain on the pairs the actor now produces depends on how well the current selector discriminates on exactly those pairs. UCoE therefore revisits the verifier at scheduled checkpoints, after y_t and H_{t+1} have been finalized. The verifier model synthesizes new candidates from the current verifier's definitions and from unlabeled summaries of the completed tasks, namely how often revisions were accepted, which criteria were repeatedly the weakest, and how the harness state has grown. A candidate may append criteria or change the aggregation. After validation, UCoE adds the current verifier to the pool and repeats the maximin selection of Section 3.2 on the fixed contrast families. The winner replaces the current verifier only if it meets the update criteria: it must improve worst-family discrimination on the selection split, win more held-out contrasts than it loses against the current verifier, and stay within a fixed regression tolerance on every existing family. Otherwise the current verifier is retained. Any adoption takes effect on the next task. See App. C.3 for the formal update criteria and App. D.6 for implementation details.

4 EXPERIMENTAL SETUP

Benchmarks and models. Our open-ended evaluation spans six benchmarks. Arena-Hard (Li et al., 2025) and WildBench (Lin et al., 2025) test general assistance on challenging user requests. WritingPrompts (Fan et al., 2018) tests story generation, and WritingBench (Wu et al., 2025) covers writing across diverse domains. MultiChallenge (Deshpande et al., 2025) targets multi-turn instruction following, and the Free-Form split of RefineBench (Lee et al., 2026c) tests response refinement. To check whether the same method transfers beyond model-judged evaluation, we add three benchmarks with objective endpoints. IFBench tests compliance with verifiable instruction constraints (Pyatkin et al., 2025), Hard-MATH is our competition-mathematics stream drawn from MATH-500 and AIME 2022–2025 (Hendrycks et al., 2021; Lightman et al., 2024), and BigCodeBench-Hard tests code generation through execution (Zhuo et al., 2025). We run Qwen3-8B and Qwen3.8-27B (Yang et al., 2025) as actors and use Qwen3.6-35B-A3B as the fixed verifier model for program synthesis, semantic criterion scoring, and diagnosis. In relation discovery, the larger comparison model is Qwen3.8-27B for the Qwen3-8B actor and Qwen3.5-397B-A17B for the Qwen3.8-27B actor. See App. D.1 for splits and sizes and App. D.4 for default hyperparameters.

Table 1: **Open-ended preference against matched direct generation.** Each entry is the tie-adjusted percentage of tasks on which a method's response is preferred to the Direct Generation response of the same actor, as judged by Qwen3.5-397B-A17B in both presentation orders with ties given half credit, so 50 is parity and higher is better. All methods share the actor, the task order, and the fixed-model, one-revision protocol. Overall is the unweighted mean over the six benchmarks, and bold marks the best method in each column.

Method	Arena-Hard	Writing Prompts	WildBench	Writing Bench	Multi Challenge	Refine Bench	Overall
Qwen3-8B							
Direct Generation	50.00	50.00	50.00	50.00	50.00	50.00	50.00
LLM as a Judge	50.60	50.54	46.34	49.50	38.87	51.98	47.97
RaR (Gunjal et al., 2026)	51.20	50.54	48.54	51.90	38.48	51.26	48.65
DREAM (Anonymous, 2026b)	45.60	43.90	43.12	52.60	31.05	53.78	45.01
SpyRL (Wang et al., 2026)	51.10	49.27	35.74	47.00	33.98	53.06	45.03
UCoE	**58.30**	**62.79**	**57.42**	**56.90**	**57.42**	**61.15**	**59.00**
Qwen3.8-27B							
Direct Generation	50.00	50.00	50.00	50.00	50.00	50.00	50.00
LLM as a Judge	49.50	48.00	48.49	50.30	36.33	48.74	46.89
RaR (Gunjal et al., 2026)	51.60	50.49	47.56	50.60	43.36	44.78	48.07
DREAM (Anonymous, 2026b)	53.80	59.96	48.44	52.10	46.48	48.38	51.53
SpyRL (Wang et al., 2026)	43.40	58.84	41.89	48.60	36.33	42.99	45.34
UCoE	**59.70**	**63.92**	**55.42**	**54.40**	**51.95**	**57.73**	**57.19**

Baselines. We compare UCoE with common baselines for open-ended self-improvement, chosen to cover different forms of feedback. Direct Generation uses no feedback and provides the raw-response reference. LLM as a Judge makes a fixed holistic judgment of response quality (Zheng et al., 2023). RaR scores responses against task-specific flat rubrics (Gunjal et al., 2026). DREAM organizes dependent criteria into a Reward-DAG (Anonymous, 2026b). SpyRL derives feedback from a task-transformation game with self-verifiable rewards (Wang et al., 2026). We adapt all four feedback baselines to the same fixed-model, one-revision protocol: each method uses its own feedback to guide at most one revision, scores the draft and the revision with its own signal, and returns the revision only when it passes the method's acceptance rule. Stateful methods receive separate procedural memories with the same capacity and update rules, and all implementations transfer the feedback mechanisms without reproducing the original parameter-training pipelines. See App. D.3 and Table 5 for each baseline's implementation and acceptance rule.

Evaluation. Open-ended benchmarks do not share an exact notion of correctness, and their native scores are not directly comparable, so our primary endpoint is tie-adjusted pairwise preference against the matched Direct Generation response. This follows established pairwise evaluation protocols for open-ended model outputs (Zheng et al., 2023; Li et al., 2025) and places every method on the same relative scale. Qwen3.5-397B-A17B judges each pair of full responses in both presentation orders without receiving method identities. We merge the two judgments at the task level and assign half credit to ties, so values above 50% favor the evaluated method over Direct Generation. We also report the native benchmark-specific metrics wherever they exist, together with accuracy or Pass@1 on the verifiable tasks. See App. D.7 for the full evaluation protocols.

5 Results

UCoE wins on every open-ended benchmark. Table 1 shows that UCoE is preferred over matched direct generation in all twelve benchmark and actor settings, and that its preference is the largest of any feedback method in every one of them. It averages 59.00% with Qwen3-8B and 57.19% with Qwen3.8-27B, gains of 9.00 and 7.19 points over direct generation, and the gains span general assistance, writing, multi-turn instruction following, and refinement. The other feedback methods often fail to improve on direct generation. Even the best baseline on each benchmark averages only 49.26% at 8B and 51.60% at 27B, and none exceeds parity on WildBench at 8B. Every baseline receives feedback and one revision under the same protocol, so the gap reflects the feedback signal itself rather than the opportunity to revise. We also evaluate under each benchmark's native metric, and

Table 2: **Verifiable tasks with Qwen3-8B.** Accuracy for IFBench and Hard-MATH, Pass@1 for BigCodeBench-Hard.

Method	IFBench	Hard-MATH	BigCodeBench-Hard	Overall
Direct Generation	30.00	56.54	**21.62**	36.06
LLM as a Judge	33.33	55.24	16.89	35.15
RaR	32.00	57.33	18.92	36.08
DREAM	28.67	**59.42**	18.24	35.44
SpyRL	30.67	58.12	18.24	35.68
UCoE	**34.67**	59.16	**21.62**	**38.48**

the same picture holds, with UCoE improving over direct generation in every setting. See App. E.1, E.2, E.3, E.5, and E.6 for native-metric results, listed-price cost, backbone transfer, pointwise intervals for Table 1, and an independent-critic audit of the WritingBench revisions.

UCoE also improves verifiable tasks. The same lifecycle helps where correctness can be checked. In Table 2, UCoE raises the three-benchmark mean from 36.06% to 38.48%. It improves IFBench by 4.67 points and Hard-MATH by 2.62 points while matching direct generation on BigCodeBench-Hard. On code, every other feedback method lowers Pass@1 below direct generation, and UCoE is the only method that preserves it, while on Hard-MATH DREAM edges it by 0.26 points.

Every component contributes. Table 3 removes one component of UCoE at a time on Arena-Hard, and every removal lowers preference. Removing current-task repair, so that the verifier only updates the harness state, costs the most, from 58.30% to 50.80%. Replacing driving-capacity selection with a random initial verifier, freezing the verifier instead of updating it at checkpoints, and disabling harness-state updates from accepted repairs each lower preference to between 56.60% and 56.70%. The ablations confirm each part of the design: verifier-guided repair, verifier selection by driving capacity, verifier updates at checkpoints, and harness-state updates from feedback all contribute. See App. E.4 for the sensitivity study.

Table 3: **Arena-Hard ablations with Qwen3-8B.** Tie-adjusted preference against direct generation.

Configuration	vs. direct %
UCoE	**58.30**
Random initial verifier	56.70
Fixed verifier	56.60
Harness-state updates only	50.80
No memory updates	56.70

UCoE transfers to a Codex agent harness. The main experiments wrap a bare actor in a prompt-level harness with procedural memory. To test whether UCoE also works inside a full coding-agent harness, we run it in the Codex harness on SpreadsheetBench-2 (Zhu et al., 2026), where the agent edits real workbooks with its tools and the benchmark's checker scores each result after recalculating every formula. The harness state here is the agent's own context, its tools, a task skill file, and an experience block in the prompt, so the study also tests whether UCoE's harness-state updates transfer beyond the procedural memory of the main experiments. GPT-5.6-luna is the actor and GPT-5.6-sol the verifier model, and Figure 3 reports a 75-task stream with checkpoints after tasks 34 and 67. UCoE reaches 22.7% exact-match accuracy against 17.3% for both Direct Generation and LLM as a Judge. The gain comes from financial-model and debugging tasks, where the fixed judge matches or trails direct generation, while on template tasks LLM as a Judge leads. See App. D.8 for details.

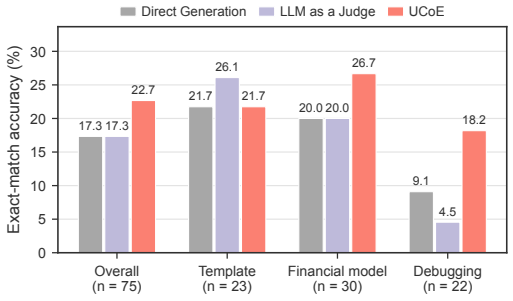
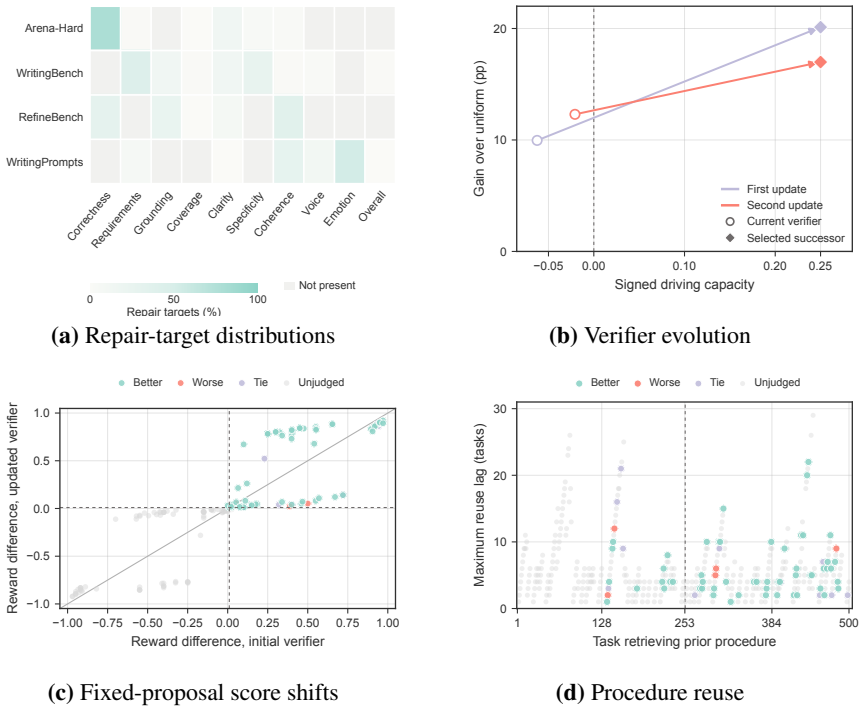


Figure 3: **UCoE inside the Codex agent harness on SpreadsheetBench-2.** Exact-match accuracy on a 75-task stream, overall and by task category, with GPT-5.6-luna as the actor.

Verifier updates and stored procedures behave as intended. Figure 4(a) shows which criterion the verifier targets when it diagnoses a draft. Correctness receives 78.9% of Arena-Hard repairs, requirements lead on WritingBench at 42.5%, coherence on RefineBench at 36.5%, and emotion on WritingPrompts at 53.0%, so the deployed verifier directs repairs at the qualities each task type demands. Figure 4(b) shows two verifier updates in WritingPrompts. Each candidate selected by



(a) Repair-target distributions

(b) Verifier evolution

(c) Fixed-proposal score shifts

(d) Procedure reuse

Figure 4: **Mechanism diagnostics.** (a) Share of repair targets per criterion group. Gray marks groups absent from the deployed verifier. (b) Two verifier updates in WritingPrompts. Each arrow runs from the current verifier to the candidate selected by driving capacity, in signed driving capacity against selection gain on frozen response pairs. (c) Reward margins of 360 Arena-Hard proposals under the initial and updated verifier. Dashed lines mark the acceptance margin, and colors give the external judgment for the 94 proposals the updated verifier selects. (d) Reuse lag of stored procedures at 381 Arena-Hard tasks, colored by outcome. The dashed line marks the switch to mathematics tasks.

driving capacity raises signed driving capacity on the contrasts, from -0.0625 and -0.0208 to 0.25, and also raises selection gain on separate frozen response pairs, by 10.16 and 4.69 points, so gains on constructed contrasts carry over to natural pairs. Figure 4(c) rescores 360 Arena-Hard proposals under the initial and the updated verifier. The update widens some reward margins and narrows others, which changes the set of accepted revisions, and among the 94 proposals it selects, external judgments favor 73 over the draft, 7 favor the draft, and 14 tie. The updated verifier therefore accepts mostly real improvements. Figure 4(d) tracks when stored procedures are retrieved. Of 639 retrieval events, 76 reuse a procedure after at least ten intervening tasks, and on the 70 retrieving tasks with external judgments the served revision beats the draft 54 times, loses 5, and ties 11, so experience in the harness state is reused long after it was stored and the tasks that reuse it are mostly judged better than their drafts.

6 Conclusion

We introduced UCoE, a training-free framework in which an executable verifier program and the harness state around a fixed actor evolve together. Response contrasts choose the verifier and test every candidate that would replace it, and the verifier in turn shapes each response and the experience the agent keeps. On six open-ended benchmarks, UCoE improves on matched direct generation at both Qwen actor sizes, and on three verifiable benchmarks it raises the mean while preserving Pass@1 on BigCodeBench-Hard. Repairing the current task contributes most of the gain, verifier updates measurably change selection behavior, and procedures stored from accepted repairs are reused many tasks later. Training-free self-improvement has so far adapted responses and harness state around a fixed feedback process. UCoE extends that adaptation to the feedback process itself, without touching model parameters. See App. A for limitations.

AI USE STATEMENT

Generative AI tools assisted with writing assistance and polishing of the manuscript. The authors take responsibility for the full content of this work.

ETHICS STATEMENT

This work does not recruit human subjects or collect new personal data. It evaluates language models on existing research benchmarks. An evolving verifier can nevertheless reinforce its own errors or biases through revision and persistent harness state, which may retain sensitive information without appropriate filtering. Improved generation may also amplify misinformation or other harmful content, while additional inference consumes computational resources. Response contrasts do not establish safety, fairness, or privacy. Consequential deployments should therefore use independent safety evaluation, human oversight, explicit retention and deletion rules, and rollback mechanisms.

REPRODUCIBILITY STATEMENT

Algorithm 1 specifies the full lifecycle and update order. App. C states the assumptions and gives complete proofs. App. D documents benchmark splits, models, baseline implementations, hyperparameters, contrast construction, verifier rules, and the procedural-memory implementation. App. E reports native-metric results, cost, cross-backbone transfer, hyperparameter sensitivity, the intervals behind the main tables, a natural-revision audit, and qualitative examples.

REFERENCES

Anonymous. CARE: Curriculum-aware rubric evolution for open-ended text generation. In *Submitted to ACL Rolling Review - March 2026*, 2026a. URL https://openreview.net/forum?id=jOlKMHT17D. under review.

Anonymous. DREAM-RL: Dynamic reward-system evolution with an agentic meta-judge for reinforcement learning on open-ended tasks. In *Submitted to ACL Rolling Review - May 2026*, 2026b. URL https://openreview.net/forum?id=9idmRtG6M5. under review.

Yifan Chen, Haitao Li, Qingyao Ai, Fengbin Zhu, Tat-Seng Chua, Min Zhang, and Yiqun Liu. Genrubric: Self-evolving rubric generation for scalable llm evaluation, 2026. URL https://arxiv.org/abs/2608.29856.

Kaustubh Deshpande, Ved Sirdeshmukh, Johannes Baptist Mols, Lifeng Jin, Ed-Yeremai Hernandez-Cardona, Dean Lee, Jeremy Kritz, Willow E. Primack, Summer Yue, and Chen Xing. MultiChallenge: A realistic multi-turn conversation evaluation benchmark challenging to frontier LLMs. In Wanxiang Che, Joyce Nabende, Ekaterina Shutova, and Mohammad Taher Pilehvar (eds.), *Findings of the Association for Computational Linguistics: ACL 2025*, pp. 18632–18702, Vienna, Austria, July 2025. Association for Computational Linguistics. ISBN 979-8-89176-256-5. doi: 10.18653/v1/2025.findings-acl.958. URL https://aclanthology.org/2025.findings-acl.958/.

Angela Fan, Mike Lewis, and Yann Dauphin. Hierarchical neural story generation. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 889–898, 2018.

Matt Gardner, Yoav Artzi, Victoria Basmov, Jonathan Berant, Ben Bogin, Sihao Chen, Pradeep Dasigi, Dheeru Dua, Yanai Elazar, Ananth Gottumukkala, Nitish Gupta, Hannaneh Hajishirzi, Gabriel Ilharco, Daniel Khashabi, Kevin Lin, Jiangming Liu, Nelson F. Liu, Phoebe Mulcaire, Qiang Ning, Sameer Singh, Noah A. Smith, Sanjay Subramanian, Reut Tsarfaty, Eric Wallace, Ally Zhang, and Ben Zhou. Evaluating models' local decision boundaries via contrast sets. In Trevor Cohn, Yulan He, and Yang Liu (eds.), *Findings of the Association for Computational Linguistics: EMNLP 2020*, pp. 1307–1323, Online, November 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.findings-emnlp.117. URL https://aclanthology.org/2020.findings-emnlp.117/.

Eric Gazoni and Charlie Clark. openpyxl: A Python library to read/write Excel 2010 xlsx/xlsm files. https://openpyxl.readthedocs.io, 2024.

Xin Guan, Xiaomeng Hu, Shen Huang, Zhenyi Wang, Bo Zhang, Zijian Li, Pengjun Xie, Bo Liu, and Jiuxin Cao. Evorubric: Self-evolving rubric-driven rl for open-ended generation, 2026. URL https://arxiv.org/abs/2605.29847.

Anisha Gunjal, Anthony Wang, Elaine Lau, Vaskar Nath, Yunzhong He, Bing Liu, and Sean M. Hendryx. Rubrics as rewards: Reinforcement learning beyond verifiable domains. In *The Fourteenth International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=c1bTcrDmt4.

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. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

Wassily Hoeffding. Probability inequalities for sums of bounded random variables. *Journal of the American statistical association*, 58(301):13–30, 1963.

Sture Holm. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics*, pp. 65–70, 1979.

Zenan Huang, Yihong Zhuang, Guoshan Lu, Zeyu Qin, Haokai Xu, Tianyu Zhao, Ru Peng, Jiaqi Hu, Zhanming Shen, Xiaomeng Hu, Xijun Gu, Peiyi Tu, Jiaxin Liu, Wenyu Chen, Yuzhuo Fu, Zhiting Fan, Yanmei Gu, Yuanyuan Wang, Zhengkai Yang, Jianguo Li, and Junbo Zhao. Reinforcement learning with rubric anchors, 2025. URL https://arxiv.org/abs/2508.12790.

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.

Ruipeng Jia, Yunyi Yang, Yongbo Gai, Kai Luo, Shihao Huang, Jianhe Lin, Xiaoxi Jiang, and Guanjun Jiang. Writing-zero: Bridge the gap between non-verifiable tasks and verifiable rewards, 2025. URL https://arxiv.org/abs/2506.00103.

Wai-Chung Kwan, Aryo Pradipta Gema, Joshua Ong Jun Leang, and Pasquale Minervini. Scope: Self-play via co-evolving policies for open-ended tasks, 2026. URL https://arxiv.org/abs/2605.31433.

Hyunin Lee, Jinglue Xu, Jeffrey Seely, Donghyun Lee, Matei Zaharia, and Yujin Tang. Recursive harness self-improvement, 2026a. URL https://arxiv.org/abs/2607.15524.

Joonhyuk Lee, Virginia Ma, Sarah Zhao, Yash Nair, Asher Spector, Regev Cohen, and Emmanuel J. Candès. Fuse: Ensembling verifiers with zero labeled data, 2026b. URL https://arxiv.org/abs/2604.18547.

Young-Jun Lee, Seungone Kim, Byung-Kwan Lee, Minkyeong Moon, Yechan Hwang, Jong Myoung Kim, Graham Neubig, Sean Welleck, and Ho-Jin Choi. Refinebench: Evaluating refinement capability of language models via checklists. In *The Fourteenth International Conference on Learning Representations*, 2026c. URL https://openreview.net/forum?id=GYJFJz9Dy5.

Tianle Li, Wei-Lin Chiang, Evan Frick, Lisa Dunlap, Tianhao Wu, Banghua Zhu, Joseph E. Gonzalez, and Ion Stoica. From crowdsourced data to high-quality benchmarks: Arena-hard and benchbuilder pipeline. In *Forty-second International Conference on Machine Learning*, 2025. URL https://openreview.net/forum?id=KfTf9vFvSn.

Weiyuan Li, Aili Chen, Xintao Wang, Yikai Zhang, Qingqing Dong, Jinghan Xu, Hongru Hou, Wenxuan Zhao, Chengkun Lang, Jun Gao, Yuanli Guo, Hongcheng Guo, Yanghua Xiao, and Deqing Yang. Evors: On-policy self-evolution of reward systems for open-ended reinforcement learning, 2026. URL https://arxiv.org/abs/2609.12459.

Shalev Lifshitz, Sheila A. McIlraith, and Yilun Du. Multi-agent verification: Scaling test-time compute with multiple verifiers. In *Second Conference on Language Modeling*, 2025. URL https://openreview.net/forum?id=LriQ3NY9uL.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=v8L0pN6EOi.

Bill Yuchen Lin, Yuntian Deng, Khyathi Chandu, Abhilasha Ravichander, Valentina Pyatkin, Nouha Dziri, Ronan Le Bras, and Yejin Choi. Wildbench: Benchmarking llms with challenging tasks from real users in the wild. In *International Conference on Learning Representations*, volume 2025, pp. 47852–47870, 2025.

Jack Lu, Ryan Teehan, Jinran Jin, and Mengye Ren. When does verification pay off? a closer look at llms as solution verifiers, 2026. URL https://arxiv.org/abs/2512.02304.

Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Eureka: Human-level reward design via coding large language models. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=IEduRUO55F.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=S37hOerQLB.

Quinn McNemar. Note on the sampling error of the difference between correlated proportions or percentages. *Psychometrika*, 12(2):153–157, 1947.

OpenAI. Codex CLI: Lightweight coding agent that runs in your terminal. https://github.com/openai/codex, 2025a.

OpenAI. GPT-5 system card. https://openai.com/index/gpt-5-system-card/, 2025b. Model documentation for the GPT-5 series, including the GPT-5.4 and GPT-5.6 models used in this work.

Teng Pan, Yuchen Yan, Zixuan Wang, Ruiqing Zhang, Guiyang Hou, Wenqi Zhang, Weiming Lu, Jun Xiao, and Yongliang Shen. Coverrl: Breaking the consensus trap in label-free reasoning via generator-verifier co-evolution, 2026. URL https://arxiv.org/abs/2603.17775.

George R. Price. Selection and covariance. *Nature*, 227(5257):520–521, 1970.

Valentina Pyatkin, Saumya Malik, Victoria Graf, Hamish Ivison, Shengyi Huang, Pradeep Dasigi, Nathan Lambert, and Hannaneh Hajishirzi. Generalizing verifiable instruction following. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=yfYgwjj5F8.

Shiori Sagawa*, Pang Wei Koh*, Tatsunori B. Hashimoto, and Percy Liang. Distributionally robust neural networks. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=ryxGuJrFvS.

Ananya B. Sai, Tanay Dixit, Dev Yashpal Sheth, Sreyas Mohan, and Mitesh M. Khapra. Perturbation CheckLists for evaluating NLG evaluation metrics. In Marie-Francine Moens, Xuanjing Huang, Lucia Specia, and Scott Wen-tau Yih (eds.), *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, pp. 7219–7234, Online and Punta Cana, Dominican Republic, November 2021. Association for Computational Linguistics. doi: 10.18653/v1/2021.emnlp-main.575. URL https://aclanthology.org/2021.emnlp-main.575/.

Rulin Shao, Akari Asai, Shannon Zejiang Shen, Hamish Ivison, Varsha Kishore, Jingming Zhuo, Xinran Zhao, Molly Park, Samuel G. Finlayson, David Sontag, Tyler Murray, Sewon Min, Pradeep Dasigi, Luca Soldaini, Faeze Brahman, Wen tau Yih, Tongshuang Wu, Luke Zettlemoyer, Yoon Kim, Hannaneh Hajishirzi, and Pang Wei Koh. DR tulu: Reinforcement learning with evolving rubrics for deep research. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=97NEP1pyS3.

William F. Shen, Xinchi Qiu, Chenxi Whitehouse, Lisa Alazraki, Shashwat Goel, Francesco Barbieri, Timon Willi, Akhil Mathur, and Ilias Leontiadis. Rethinking rubric generation for improving llm judge and reward modeling for open-ended tasks, 2026. URL https://arxiv.org/abs/2602.05125.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik R Narasimhan, and Shunyu Yao. Reflexion: language agents with verbal reinforcement learning. In *Thirty-seventh Conference on Neural Information Processing Systems*, 2023. URL https://openreview.net/forum?id=vAElhFcKW6.

Charlie Snell, Jaehoon Lee, Kelvin Xu, and Aviral Kumar. Scaling llm test-time compute optimally can be more effective than scaling parameters for reasoning. In *International Conference on Learning Representations*, volume 2025, pp. 10131–10165, 2025.

Yuan Sui and Bryan Hooi. Conversation for non-verifiable learning: Self-evolving large language models through meta-evaluation. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=Wba6w3pzbj.

Mirac Suzgun, Mert Yuksekgonul, Federico Bianchi, Dan Jurafsky, and James Zou. Dynamic cheatsheet: Test-time learning with adaptive memory. In Vera Demberg, Kentaro Inui, and Lluís Marquez (eds.), *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 7080–7106, Rabat, Morocco, March 2026. Association for Computational Linguistics. ISBN 979-8-89176-380-7. doi: 10.18653/v1/2026.eacl-long.333. URL https://aclanthology.org/2026.eacl-long.333/.

The Document Foundation. LibreOffice. https://www.libreoffice.org, 2024.

Yuxuan Wan, Tianqing Fang, Zaitang LI, Yintong Huo, Wenxuan Wang, Haitao Mi, Dong Yu, and Michael R. Lyu. Inference-time scaling of verification: Self-evolving deep research agents via test-time rubric-guided verification. In Maria Liakata, Viviane P. Moreira, Jiajun Zhang, and David Jurgens (eds.), *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 24822–24835, San Diego, California, United States, July 2026. Association for Computational Linguistics. ISBN 979-8-89176-395-1. doi: 10.18653/v1/2026.findings-acl.1243. URL https://aclanthology.org/2026.findings-acl.1243/.

Qinsi Wang, Jing Shi, Huazheng Wang, Kun Wan, Yiran Wu, Bo Liu, Qingyun Wu, Hai Helen Li, Yiran Chen, Handong Zhao, and Wentian Zhao. From rlvr to rlsvr: Task transformation induces self-verifiable rewards for open-ended llm self-improvement, 2026. URL https://arxiv.org/abs/2607.23802.

Yuning Wu, Jiahao Mei, Ming Yan, Chenliang Li, Shaopeng Lai, Yuran Ren, Wang Zijia, Ji Zhang, Mengyue Wu, Qin Jin, and Fei Huang. Writingbench: A comprehensive benchmark for generative writing. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2025. URL https://openreview.net/forum?id=Pkskg9drDQ.

Ran Xu, Tianci Liu, Zihan Dong, Tony Yu, Ilgee Hong, Carl Yang, Linjun Zhang, Tuo Zhao, and Haoyu Wang. Alternating reinforcement learning for rubric-based reward modeling in non-verifiable LLM post-training. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=3SDg8dtkS8.

Tianyi Xu, Yaolun Zhang, Xuan Ouyang, and Huazheng Wang. Evopool: Evolutionary programmatic annotation for label-efficient specialized supervision, 2026b. URL https://arxiv.org/abs/2606.01617.

Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for LLM agents. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. URL https://openreview.net/forum?id=FiM0M8gcct.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, Jian Yang, Jianhong Tu, Jianwei Zhang, Jianxin Yang, Jiaxi Yang, Jing Zhou, Jingren Zhou, Junyang Lin, Kai Dang, Keqin Bao, Kexin Yang, Le Yu, Lianghao Deng, Mei Li, Mingfeng Xue, Mingze Li, Pei Zhang, Peng Wang, Qin Zhu, Rui Men, Ruize Gao, Shixuan Liu, Shuang Luo, Tianhao Li, Tianyi Tang, Wenbiao Yin, Xingzhang Ren, Xinyu Wang, Xinyu Zhang, Xuancheng Ren, Yang Fan, Yang Su, Yichang Zhang, Yinger Zhang, Yu Wan, Yuqiong Liu, Zekun Wang, Zeyu Cui, Zhenru Zhang, Zhipeng Zhou, and Zihan Qiu. Qwen3 technical report, 2025. URL https://arxiv.org/abs/2505.09388.

William J Youden. Index for rating diagnostic tests. *Cancer*, 3(1):32–35, 1950.

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. In *The Fourteenth International Conference on Learning Representations*, 2026a. URL https://openreview.net/forum?id=eC4ygDs02R.

Yaolun Zhang, Tianyi Xu, Shengyu Dai, Zhenwen Shao, Qingyun Wu, and Huazheng Wang. Evochamber: Test-time co-evolution of multi-agent system at individual, team, and population scales, 2026b. URL https://arxiv.org/abs/2605.11136.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric Xing, et al. Judging llm-as-a-judge with mt-bench and chatbot arena. *Advances in neural information processing systems*, 36:46595–46623, 2023.

Jian Zhu, Yuzheng Zhang, Zeyao Ma, Bohan Zhang, Armin Schoepf, Daniel Woloch, Peter Yiliu Wang, Guangyu Robert Yang, Samuel Jacob, Siddharth Nagisetty, Abhiram Chundru, Jean Lin, Spencer Mateega, and Jing Zhang. Spreadsheetbench 2: Evaluating agents on end-to-end business spreadsheet workflows, 2026. URL https://arxiv.org/abs/2606.29955.

King Zhu, Hanhao Li, Siwei Wu, Tianshun Xing, Dehua Ma, Xiangru Tang, Minghao Liu, Jian Yang, Jiaheng Liu, Yuchen Eleanor Jiang, Changwang Zhang, Chenghua Lin, Jun Wang, Ge Zhang, and Wangchunshu Zhou. Scaling test-time compute for llm agents, 2025. URL https://arxiv.org/abs/2506.12928.

Terry Yue Zhuo, Vu Minh Chien, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, Simon Brunner, Chen GONG, James Hoang, Armel Randy Zebaze, Xiaoheng Hong, Wen-Ding Li, Jean Kaddour, Ming Xu, Zhihan Zhang, Prateek Yadav, Naman Jain, Alex Gu, Zhoujun Cheng, Jiawei Liu, Qian Liu, Zijian Wang, David Lo, Binyuan Hui, Niklas Muennighoff, Daniel Fried, Xiaoning Du, Harm de Vries, and Leandro Von Werra. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=YrycTjllL0.

A LIMITATIONS

UCoE measures driving capacity on declared families of local edits, so passing the response contrasts does not guarantee reliable discrimination on natural revisions, on interacting errors, or on task properties that no family covers. Broader contrasts with independently checked preference directions would give verifier selection a stronger test. The deployed verifier also carries a coupled risk: because it diagnoses weaknesses, selects the served response, and decides which repairs enter the harness state, one mistaken judgment can accept a harmful repair and store a misleading procedure. Independent checks or rollback rules would reduce that risk. Finally, open-ended evaluation still relies on model judges, and repeated semantic scoring makes UCoE more expensive than the simpler feedback baselines, so validation with diverse evaluators or humans and cheaper verifier execution are the next steps.

B ALGORITHM

Algorithm 1 gives the complete lifecycle. Initialization runs at $t = 0$, and every later checkpoint synthesizes candidates from the diagnostics of completed tasks only. H denotes the harness state, E the diagnostic history, and $\mathcal{R}$ the fixed relation inventory mined from the development pairs $\mathcal{P}_{\mathrm{dev}}$. The main experiments instantiate H as a bounded procedural memory, and the agent-harness study of App. D.8 instantiates it as the context of a coding agent.

The update criteria in $\mathrm{UpdateCriteria}$ require a strict gain in worst-family accuracy on the selection split, more held-out contrasts won than lost against the current verifier, and no regression beyond the tolerance on any existing family, and any failed or undefined comparison retains the current verifier. Equation (19) states them exactly. While no verifier is deployed, UCoE serves the draft and leaves the harness state unchanged, but later checkpoints still run. Calibration data and final-evaluator feedback never enter deployment. The reference procedure draws fresh comparison sources at each checkpoint.

C THEORETICAL ANALYSIS

We analyze the selectors induced by UCoE's verifier programs on fixed response-pair distributions. The results explain the normalization of driving capacity, the worst-family selection objective, and the guarantees of the empirical update rule. Hats denote sample estimates, and population bounds require the sampling assumptions stated below.

C.1 PAIRWISE DISCRIMINATION AND SELECTION GAIN

For a task x, let y_c be the proposed response and y_r the reference response. With a fixed margin $\epsilon \geq 0$, the verifier induces

$$h_{\mathcal{V},\epsilon}(x; y_c, y_r) = \begin{cases} y_c, & r_{\mathcal{V}}(x, y_c) > r_{\mathcal{V}}(x, y_r) + \epsilon, \\ y_r, & \text{otherwise.} \end{cases} \tag{10}$$

An undefined comparison returns y_r. Contrasts use $(y_c, y_r) = (y^+, y^-)$, whereas deployment uses $(y_c, y_r) = (y^R, y^0)$. We write $h_{\mathcal{V}}$ for this selector at the fixed margin. For family j, define $p_j(h) = \Pr(h \text{ selects } y^+ \mid j)$ and $J_j(h) = 2p_j(h) - 1$. Uniform random choice gives $J_j = 0$. A verifier that always assigns equal scores has $J_j = -1$, since every tie fails the strict-margin test.

For the covariance interpretation (Price, 1970), write $(Y_1, Y_2) = (y^+, y^-)$ and let $S_h \in \{1, 2\}$ be the selected position. Let $U_i \in \{0, 1\}$ indicate whether Y_i satisfies the targeted property. A valid contrast must satisfy $(U_1, U_2) = (1, 0)$. This is a semantic assumption about the edited property, which eligibility checks alone cannot establish. Draw $I \sim \mathrm{Unif}\{1, 2\}$ independently of the contrast, its family, and all scoring and selection randomness. Set $G_h = \mathbf{1}[S_h = I]$ and $U = U_I$. The auxiliary index does not change the response order or the selector.

Proposition 1 (Pairwise selection covariance)**.** *For valid targeted contrasts and the independent index defined above,*

$$\mathrm{Cov}(G_h, U \mid j) = \frac{1}{4} J_j(h). \tag{11}$$

Algorithm 1 Unified Co-Evolution

Require: Public requirements R, optional development response pairs $\mathcal{P}_{\mathrm{dev}}$, actor F_{θ_0}, verifier model, initial harness state H_{init}, and task stream $x_{1:T}$
Require: Calibration source reservoir $\mathcal{S}$ and checkpoint set $\mathcal{T}$
Require: Candidate budget L, contrast families, comparison budgets, margin ϵ, harness-update rules
Ensure: Returned responses $y_{1:T}$, harness state H, and verifier-decision history

$H \leftarrow H_{\mathrm{init}}$, $E \leftarrow \varnothing$, and $\mathcal{V} \leftarrow \mathcal{V}_{\perp}$
$\mathcal{R} \leftarrow \mathrm{MineRelations}(\mathcal{P}_{\mathrm{dev}})$ if supplied; otherwise $\mathcal{R} \leftarrow \varnothing$
for $t = 0, \ldots, T$ **do**
  if $t > 0$ **then**
    $y_t^0 \leftarrow F_{\theta_0}(x_t; H)$ through the configured harness interface
    $(y_t, y_t^R, a_t, c_t, w_t, u_t) \leftarrow (y_t^0, \perp, 0, \perp, \perp, \perp)$
    if $\mathcal{V} \neq \mathcal{V}_{\perp}$ **then**
      $(c_t, w_t, u_t) \leftarrow D_{\mathcal{V}}(x_t, y_t^0)$
      $y_t^R \leftarrow F_{\theta_0}(x_t, y_t^0, c_t, w_t, u_t; H)$
      if $r_{\mathcal{V}}(x_t, y_t^R) > r_{\mathcal{V}}(x_t, y_t^0) + \epsilon$ **then**
        $(y_t, a_t) \leftarrow (y_t^R, 1)$
      end if
    end if
    Return y_t; subsequent updates derived from task t affect only later tasks
    if $\mathcal{V} \neq \mathcal{V}_{\perp}$ **then**
      $H \leftarrow \mathrm{UpdateHarness}(H, x_t, y_t^0, y_t^R, c_t, w_t, a_t)$ ▷ new state content only if $a_t = 1$
    end if
    Append the completed task and available verifier diagnostics to E
  end if
  if $t = 0$ or $t \in \mathcal{T}$ **then**
    $d \leftarrow \mathrm{Summarize}(E)$, with $d = \varnothing$ at initialization
    $\mathcal{K} \leftarrow \mathrm{GenerateVerifiers}(R, \mathcal{R}, d, L)$ using the verifier model
    Include $\mathcal{V}$ in $\mathcal{K}$ if $\mathcal{V} \neq \mathcal{V}_{\perp}$ and discard non-executable candidates
    Freeze all candidate programs and the scoring, diagnosis, and selection conventions
    if $\mathcal{K} \neq \varnothing$ **then**
      $(B_{\mathrm{sel}}, B_{\mathrm{val}}) \leftarrow \mathrm{Contrasts}(\mathcal{S}, \mathcal{R})$ ▷ source-disjoint within checkpoint
      $\widehat{\mathcal{V}} \in \arg\max_{\mathcal{V}' \in \mathcal{K}} \mathrm{round}_4(\min_j \widehat{J}_{\mathrm{sel},j}(h_{\mathcal{V}'}))$
      Fix $\widehat{\mathcal{V}}$ before evaluating any required held-out comparisons
      if $\mathcal{V} = \mathcal{V}_{\perp}$ **then**
        $b \leftarrow 1$ ▷ adopt the initial selected candidate
      else
        $b \leftarrow \mathrm{UpdateCriteria}(\widehat{\mathcal{V}}, \mathcal{V}, B_{\mathrm{sel}}, B_{\mathrm{val}})$ ▷ Eq. (19)
      end if
      if $b = 1$ **then**
        $\mathcal{V} \leftarrow \widehat{\mathcal{V}}$ ▷ affects only tasks after t
      end if
    end if
    Record the selection, held-out comparison, and adopt-or-retain decision
  end if
end for
Finalize all responses, verifier decisions, and harness states before final evaluation
return $(y_{1:T}, H, E)$ and the verifier-decision history

Thus the signed driving capacity $\widetilde{\mathcal{C}}_{\mathcal{B}}$ in Eq. (4) is the minimum selection covariance over the tested families. Its nonnegative part is $\mathcal{C}_{\mathcal{B}}$.

Proof. Fix a family j. Since $(U_1, U_2) = (1, 0)$, we have $U = 1$ exactly when $I = 1$. The index I is uniform and independent of S_h conditional on j, so

$$\mathbb{E}[U \mid j] = \Pr(I = 1 \mid j) = \frac{1}{2},$$
$$\mathbb{E}[G_h \mid j] = \sum_{s=1}^{2} \Pr(I = s, S_h = s \mid j) = \frac{1}{2} \sum_{s=1}^{2} \Pr(S_h = s \mid j) = \frac{1}{2},$$
$$\mathbb{E}[G_h U \mid j] = \Pr(I = 1, S_h = 1 \mid j) = \frac{1}{2} \Pr(S_h = 1 \mid j) = \frac{1}{2} p_j(h).$$

The last line uses $G_h U = 1$ exactly when the sampled position and the selected position are both 1. Substituting these expectations into the definition of covariance gives

$$\begin{aligned}\mathrm{Cov}(G_h, U \mid j) &= \mathbb{E}[G_h U \mid j] - \mathbb{E}[G_h \mid j]\,\mathbb{E}[U \mid j] \\ &= \frac{p_j(h)}{2} - \frac{1}{2} \cdot \frac{1}{2} = \frac{2p_j(h) - 1}{4} = \frac{1}{4} J_j(h).\end{aligned}$$

For $h = h_{\mathcal{V}}$, taking the minimum over the tested families yields

$$\min_j \mathrm{Cov}(G_h, U \mid j) = \frac{1}{4} \min_j J_j(h) = \widetilde{\mathcal{C}}_{\mathcal{B}}(\mathcal{V}).$$

Clipping this minimum at zero gives $\mathcal{C}_{\mathcal{B}}(\mathcal{V})$. □

The auxiliary retention rates are $\Pr(G_h = 1 \mid U = 1, j) = p_j(h)$ and $\Pr(G_h = 1 \mid U = 0, j) = 1 - p_j(h)$, whose difference is Youden's index $J_j(h)$ (Youden, 1950). Because contrasts always place the preferred response in the proposed position and count ties and reversed rankings alike as failures, these rates measure discrimination on the tested property, not the acceptance of natural revisions.

To describe selection gain on general pairs, fix a distribution of (x, Y_1, Y_2, Z_1, Z_2), where $Z_i \in \{0, 1\}$ is an evaluation-only adequacy label for Y_i. Each selector chooses a position $S_h \in \{1, 2\}$ on the same pairs. Let $M = \mathbf{1}[Z_1 \neq Z_2]$, $q = \Pr(M = 1)$, and, for $q > 0$, define $J_{\mathrm{mix}}(h) = 2 \Pr(Z_{S_h} = 1 \mid M = 1) - 1$.

Proposition 2 (Selection gain on fixed response pairs)**.** *For any two selectors h, h' on this pair distribution with $q > 0$,*

$$\mathbb{E}[Z_{S_{h'}} - Z_{S_h}] = \frac{q}{2} \{J_{\mathrm{mix}}(h') - J_{\mathrm{mix}}(h)\}. \tag{12}$$

If $q = 0$, the expected adequacy difference is zero.

Proof. When $M = 0$, we have $Z_1 = Z_2$, so $Z_{S_{h'}} - Z_{S_h} = 0$ regardless of which positions the selectors choose. Thus $Z_{S_{h'}} - Z_{S_h} = M(Z_{S_{h'}} - Z_{S_h})$. If $q = 0$, this difference is zero almost surely, proving the second claim.

For $q > 0$, binary adequacy and the definition of J_{mix} give

$$\mathbb{E}[Z_{S_h} \mid M = 1] = \Pr(Z_{S_h} = 1 \mid M = 1) = \frac{1 + J_{\mathrm{mix}}(h)}{2},$$

and the same equality holds for h'. Conditioning on $M = 1$ and substituting both expressions, we obtain

$$\begin{aligned}\mathbb{E}[Z_{S_{h'}} - Z_{S_h}] &= \mathbb{E}\big[M(Z_{S_{h'}} - Z_{S_h})\big] \\ &= q\,\mathbb{E}[Z_{S_{h'}} - Z_{S_h} \mid M = 1] \\ &= q\left\{\mathbb{E}[Z_{S_{h'}} \mid M = 1] - \mathbb{E}[Z_{S_h} \mid M = 1]\right\} \\ &= q\left\{\frac{1 + J_{\mathrm{mix}}(h')}{2} - \frac{1 + J_{\mathrm{mix}}(h)}{2}\right\} \\ &= \frac{q}{2}\left\{J_{\mathrm{mix}}(h') - J_{\mathrm{mix}}(h)\right\}.\end{aligned}$$

□

The factor q measures how often selection can matter, and the change in J_{mix} measures how well the new selector uses those opportunities. The identity is about binary adequacy on a fixed pair distribution, so carrying it to contrast accuracy or to the deployed stream, where diagnosis, proposals, and harness state all move the pairs, needs further assumptions.

C.2 Worst-family selection over a finite verifier pool

Let $m \geq 1$ be the number of contrast families. For population statements, $\mathcal{B}$ denotes their fixed distributions, and probabilities include any scoring randomness. We write $\widetilde{\mathcal{C}}_{\mathcal{B}}(h) = \frac{1}{4}\min_j J_j(h)$ and $\mathcal{C}_{\mathcal{B}}(h) = \max\{0, \widetilde{\mathcal{C}}_{\mathcal{B}}(h)\}$ for the capacities of an induced selector. A family mixture has weights $w \in \Delta_m = \{w \in [0,1]^m : \sum_j w_j = 1\}$ and success probability $p_w(h) = \sum_j w_j p_j(h)$.

Theorem 1 (Robust interpretation of driving capacity)**.** *For every selector h,*

$$\inf_{w \in \Delta_m} p_w(h) = \min_j p_j(h) = \frac{1}{2} + 2\widetilde{\mathcal{C}}_{\mathcal{B}}(h). \tag{13}$$

Maximizing signed driving capacity over a nonempty finite pool therefore maximizes the worst-case success probability over all mixtures of the tested families.

Proof. Fix a selector h and choose $j^\star \in \arg\min_{1 \leq j \leq m} p_j(h)$. For any $w \in \Delta_m$, the weights are nonnegative and sum to one, so

$$\begin{aligned} p_w(h) &= \sum_{j=1}^m w_j p_j(h) \geq \sum_{j=1}^m w_j p_{j^\star}(h) \\ &= p_{j^\star}(h) \sum_{j=1}^m w_j = p_{j^\star}(h). \end{aligned}$$

Thus every mixture has success probability at least $\min_j p_j(h)$. This bound is attained by the weights $w^\star_{j^\star} = 1$ and $w^\star_j = 0$ for $j \neq j^\star$, which belong to Δ_m and satisfy

$$p_{w^\star}(h) = p_{j^\star}(h) = \min_j p_j(h).$$

Combining the lower bound with this attaining mixture gives $\inf_{w \in \Delta_m} p_w(h) = \min_j p_j(h)$.

Using $J_j(h) = 2p_j(h) - 1$ and the definition of signed driving capacity,

$$\begin{aligned} \min_j p_j(h) &= \min_j \frac{1 + J_j(h)}{2} \\ &= \frac{1}{2} + \frac{1}{2}\min_j J_j(h) \\ &= \frac{1}{2} + 2\left(\frac{1}{4}\min_j J_j(h)\right) = \frac{1}{2} + 2\widetilde{\mathcal{C}}_{\mathcal{B}}(h). \end{aligned}$$

The last expression is strictly increasing in signed driving capacity. Therefore, over any nonempty finite selector pool, signed driving capacity and worst-case mixture success probability have the same maximizers. □

This is the group maximin structure used in distributionally robust optimization (Sagawa* et al., 2020): it covers changes in family weights while each family's distribution stays fixed. Maximizing the signed capacity also maximizes its nonnegative part while still separating candidates that clipping would assign the same zero value.

Sampling assumptions. Fix a nonempty finite selector pool $\mathcal{H}$ and family sample sizes $n_j \geq 1$ before drawing selection sources. Each source used in family j contributes one comparison to that family. For a fixed h and j, assume the outcomes

$$X_{ijh} = 2\mathbf{1}[h \text{ selects } y^+ \text{ on source } i \text{ in family } j] - 1, \qquad i = 1, \ldots, n_j,$$

are independent across sources. Define $\widehat{J}_j(h) = n_j^{-1} \sum_i X_{ijh}$ and assume $n_j^{-1} \sum_i \mathbb{E}[X_{ijh}] = J_j(h)$. This allows fixed stratification with the corresponding population mixture. Every selector is evaluated on all declared comparisons with a predeclared outcome for missing or invalid scores. Candidate-dependent removal of comparisons is excluded. Different families may share sources, and different selectors may share scores. For history-generated pools, these assumptions hold conditional on the history and synthesis information fixed before sampling.

Proposition 3 (Finite-pool maximin selection)**.** *Under these assumptions, for $\delta \in (0, 1)$ let*

$$\rho_\delta = \max_j \sqrt{\frac{2}{n_j} \log \frac{2|\mathcal{H}|m}{\delta}}. \tag{14}$$

With probability at least $1 - \delta$,

$$\sup_{h \in \mathcal{H},\, 1 \le j \le m} |\widehat{J}_j(h) - J_j(h)| \le \rho_\delta. \tag{15}$$

On this event, any $\widehat{h} \in \mathcal{H}$ whose empirical worst-family score is within $\tau \ge 0$ of the maximum satisfies

$$\min_j J_j(\widehat{h}) \ge \underbrace{\max_{h \in \mathcal{H}} \min_j J_j(h)}_{\substack{\text{best population} \\ \text{score in the pool}}} - \underbrace{2\rho_\delta}_{\substack{\text{sampling} \\ \text{error bound}}} - \underbrace{\tau}_{\substack{\text{optimization} \\ \text{error}}}. \tag{16}$$

Its signed or nonnegative capacity is at most $\rho_\delta/2 + \tau/4$ below the best corresponding capacity in the pool. Exact maximization has $\tau = 0$. Maximizing scores rounded to four decimals gives $\tau \le 10^{-4}$.

Proof. Fix a selector $h \in \mathcal{H}$ and a family j. The sampling assumptions give independence across sources and $\mathbb{E}[\widehat{J}_j(h)] = J_j(h)$. Since $X_{ijh} \in [-1, 1]$, Hoeffding's inequality (Hoeffding, 1963) yields, for $u > 0$,

$$\Pr\Big(|\widehat{J}_j(h) - J_j(h)| > u\Big) \le 2 \exp\left(-\frac{2n_j^2 u^2}{\sum_{i=1}^{n_j} (1 - (-1))^2}\right) = 2 \exp\left(-\frac{n_j u^2}{2}\right).$$

By the definition of ρ_δ, each family satisfies $n_j \rho_\delta^2 / 2 \ge \log(2|\mathcal{H}|m/\delta)$. Substituting $u = \rho_\delta$ gives

$$\Pr\Big(|\widehat{J}_j(h) - J_j(h)| > \rho_\delta\Big) \le 2 \exp\left(-\log \frac{2|\mathcal{H}|m}{\delta}\right) = \frac{\delta}{|\mathcal{H}|m}.$$

A union bound over all $|\mathcal{H}|m$ selector–family pairs then gives

$$\begin{aligned} \Pr\left(\sup_{h \in \mathcal{H},\, 1 \le j \le m} |\widehat{J}_j(h) - J_j(h)| > \rho_\delta\right) &\le \sum_{h \in \mathcal{H}} \sum_{j=1}^{m} \Pr\Big(|\widehat{J}_j(h) - J_j(h)| > \rho_\delta\Big) \\ &\le |\mathcal{H}|m \, \frac{\delta}{|\mathcal{H}|m} = \delta. \end{aligned}$$

This proves Eq. (15) without requiring independence between different selectors or families.

Work on this uniform event. For every h and j, $\widehat{J}_j(h) - \rho_\delta \le J_j(h) \le \widehat{J}_j(h) + \rho_\delta$. Taking minima over j gives

$$\min_j \widehat{J}_j(h) - \rho_\delta \le \min_j J_j(h) \le \min_j \widehat{J}_j(h) + \rho_\delta.$$

Thus each empirical minimum differs from its population minimum by at most ρ_δ, even if the minimizing families differ. Choose $h^\star \in \arg\max_{h \in \mathcal{H}} \min_j J_j(h)$, which exists because $\mathcal{H}$ is nonempty and finite. The assumed empirical optimization accuracy means

$$\min_j \widehat{J}_j(\widehat{h}) \ge \max_{h \in \mathcal{H}} \min_j \widehat{J}_j(h) - \tau \ge \min_j \widehat{J}_j(h^\star) - \tau.$$

Combining these inequalities yields

$$\begin{aligned}\min_j J_j(\widehat{h}) &\geq \min_j \widehat{J}_j(\widehat{h}) - \rho_\delta \\ &\geq \min_j \widehat{J}_j(h^\star) - \rho_\delta - \tau \\ &\geq \min_j J_j(h^\star) - 2\rho_\delta - \tau \\ &= \max_{h\in\mathcal{H}} \min_j J_j(h) - 2\rho_\delta - \tau,\end{aligned}$$

which proves Eq. (16).

Dividing by four and using the definition of signed driving capacity gives

$$0 \leq \widetilde{\mathcal{C}}_{\mathcal{B}}(h^\star) - \widetilde{\mathcal{C}}_{\mathcal{B}}(\widehat{h}) \leq \frac{2\rho_\delta + \tau}{4} = \frac{\rho_\delta}{2} + \frac{\tau}{4}.$$

For any $a \geq b$, the positive-part map satisfies $0 \leq \max\{0, a\} - \max\{0, b\} \leq a - b$. Apply this with $a = \widetilde{\mathcal{C}}_{\mathcal{B}}(h^\star)$ and $b = \widetilde{\mathcal{C}}_{\mathcal{B}}(\widehat{h})$. Since $h^\star$ also maximizes the nonnegative capacity, the same upper bound holds for its gap to the best in the pool.

Finally, write $\widehat{F}(h) = \min_j \widehat{J}_j(h)$ and $\overline{F}(h) = \mathrm{round}_4(\widehat{F}(h))$. Rounding to four decimals gives $|\overline{F}(h) - \widehat{F}(h)| \leq \eta$, where $\eta = 5 \times 10^{-5}$. If $\widehat{h}$ maximizes $\overline{F}$, then for every $h \in \mathcal{H}$,

$$\begin{aligned}\widehat{F}(\widehat{h}) &\geq \overline{F}(\widehat{h}) - \eta \\ &\geq \overline{F}(h) - \eta \\ &\geq \widehat{F}(h) - 2\eta.\end{aligned}$$

Taking the maximum over h shows that the empirical optimization loss is at most $2\eta = 10^{-4}$, proving the stated rounding bound. □

Interpretation. Equation (16) bounds the selected verifier's population worst-family score relative to the best candidate in the fixed pool. The sampling term $2\rho_\delta$ allows the selected candidate's empirical score to overestimate its true score by ρ_δ and the population optimum's empirical score to underestimate its true score by another ρ_δ. The term τ accounts for approximate empirical maximization, including rounding. Writing $n_{\min} = \min_j n_j$, we have $\rho_\delta = \sqrt{(2/n_{\min}) \log(2|\mathcal{H}|m/\delta)}$, so the least-sampled family controls this uniform bound. The guarantee is stated on the fixed contrast distributions.

Comparison with the current verifier. Equation (16) compares the selected candidate with the population optimum in the pool. At an update checkpoint, we can instead compare a candidate h' with the current selector h_k, assuming both belong to $\mathcal{H}$. Let their observed, unrounded gain in empirical worst-family score be

$$g = \min_j \widehat{J}_j(h') - \min_j \widehat{J}_j(h_k).$$

On the uniform event of Eq. (15), the candidate's population score is at least its empirical score minus ρ_δ, whereas the current selector's population score is at most its empirical score plus ρ_δ. Subtracting these two bounds gives

$$\begin{aligned}\min_j J_j(h') - \min_j J_j(h_k) &\geq \left(\min_j \widehat{J}_j(h') - \rho_\delta\right) - \left(\min_j \widehat{J}_j(h_k) + \rho_\delta\right) \\ &= g - 2\rho_\delta.\end{aligned}$$

Dividing by four converts the score difference into a signed-capacity difference:

$$\widetilde{\mathcal{C}}_{\mathcal{B}}(h') - \widetilde{\mathcal{C}}_{\mathcal{B}}(h_k) \geq \frac{g - 2\rho_\delta}{4}. \tag{17}$$

Thus g is the improvement measured on the selection sample, and $2\rho_\delta$ accounts for estimation error on both sides of the comparison. No optimization term τ is needed here because g compares these two selectors directly, regardless of how the candidate was chosen within the pool. If $g > 2\rho_\delta$, the bound is strictly positive and certifies an increase in population signed driving capacity on the uniform event. When $0 < g \leq 2\rho_\delta$ the sample shows a gain that the bound cannot certify, and small family budgets make it uninformative. This threshold is a sufficient condition for the guarantee only. UCoE adopts by the empirical selection gain and the held-out comparisons of the next subsection.

C.3 Evolution under the empirical verifier-update rule

At a checkpoint, UCoE includes the current verifier in the selection pool, chooses a candidate, and fixes it before evaluating held-out contrasts. Let h and h' be the current and selected candidate selectors. The family inventory stays fixed. We first establish what adoption guarantees on the observed comparison sets, then bound the population capacity change under additional sampling assumptions. The empirical results below assume common comparison sets with finite scores for both selectors, so all paired counts and family accuracies use the same denominators.

Let $n_j \geq 1$ now denote the held-out count in family j, with $N = \sum_j n_j$. For held-out comparison i in family j, define $C_{ij}(h) = \mathbf{1}[h \text{ selects } y^+]$, $\widehat{J}_{\mathrm{val},j}(h) = 2n_j^{-1} \sum_i C_{ij}(h) - 1$, and $\widehat{p}_{\mathrm{pool}}(h) = N^{-1} \sum_{j,i} C_{ij}(h)$. Thus $\widehat{p}_{\mathrm{pool}}$ weights each contrast equally, giving family j weight n_j/N, and it is an equal-family average only when the family counts are equal. The selection-split estimate $\widehat{J}_{\mathrm{sel},j}$ is defined in the same way using that split's comparisons. Write

$$W = \sum_{j,i} \mathbf{1}[C_{ij}(h') = 1, C_{ij}(h) = 0], \qquad L = \sum_{j,i} \mathbf{1}[C_{ij}(h') = 0, C_{ij}(h) = 1],$$
$$\widehat{\Delta} J_j = \widehat{J}_{\mathrm{val},j}(h') - \widehat{J}_{\mathrm{val},j}(h), \qquad \widehat{\Gamma}_{\mathrm{sel}} = \min_j \widehat{J}_{\mathrm{sel},j}(h') - \min_j \widehat{J}_{\mathrm{sel},j}(h). \tag{18}$$

Thus W and L count candidate-only and current-only successes. Let $\overline{\Gamma}_{\mathrm{sel}}$ be the difference between the two selection minima after each is rounded to four decimals. For these complete comparisons, the update rule is

$$h_{k+1} = \begin{cases} h', & \underbrace{\overline{\Gamma}_{\mathrm{sel}} > 0}_{\text{selection gain}}, \quad \underbrace{W > L}_{\text{held-out gain}}, \quad \underbrace{\min_j \widehat{\Delta} J_j \geq -\kappa}_{\text{bounded family regression}}, \\ h, & \text{otherwise}, \end{cases} \qquad \kappa = 0.10. \tag{19}$$

The first condition requires a strict increase in the selection split's worst-family score. The held-out conditions require more candidate-only than current-only successes and limit the decrease in every family's centered accuracy to κ. A missing required family check retains h. Initialization adopts a maximin winner without this paired test, since there is no current verifier.

For $s \in \{\mathrm{sel}, \mathrm{val}\}$, define $\widehat{\widetilde{\mathcal{C}}}_s(h) = \frac{1}{4} \min_j \widehat{J}_{s,j}(h)$ and $\widehat{\mathcal{C}}_s(h) = \max\{0, \widehat{\widetilde{\mathcal{C}}}_s(h)\}$.

Proposition 4 (Empirical gain and bounded family regression)**.** *Every candidate adopted under Eq. (19) satisfies*

$$\widehat{\widetilde{\mathcal{C}}}_{\mathrm{sel}}(h') - \widehat{\widetilde{\mathcal{C}}}_{\mathrm{sel}}(h) = \frac{1}{4} \widehat{\Gamma}_{\mathrm{sel}} > 0, \tag{20}$$
$$\widehat{p}_{\mathrm{pool}}(h') - \widehat{p}_{\mathrm{pool}}(h) = \frac{W - L}{N} > 0, \tag{21}$$
$$\widehat{\widetilde{\mathcal{C}}}_{\mathrm{val}}(h') - \widehat{\widetilde{\mathcal{C}}}_{\mathrm{val}}(h) \geq -\frac{\kappa}{4}. \tag{22}$$

The last bound also holds for the nonnegative held-out capacity.

Proof. Suppose h' is adopted, so all three conditions in Eq. (19) hold. Rounding to four decimals is nondecreasing. If the candidate's unrounded selection minimum were no larger than the current selector's, its rounded minimum could not be larger either. Thus $\overline{\Gamma}_{\mathrm{sel}} > 0$ implies $\widehat{\Gamma}_{\mathrm{sel}} > 0$. Substituting the definition of empirical signed capacity,

$$\begin{aligned} \widehat{\widetilde{\mathcal{C}}}_{\mathrm{sel}}(h') - \widehat{\widetilde{\mathcal{C}}}_{\mathrm{sel}}(h) &= \frac{1}{4} \left\{ \min_j \widehat{J}_{\mathrm{sel},j}(h') - \min_j \widehat{J}_{\mathrm{sel},j}(h) \right\} \\ &= \frac{1}{4} \widehat{\Gamma}_{\mathrm{sel}} > 0, \end{aligned}$$

which proves Eq. (20).

For the pooled success rate, each difference $C_{ij}(h') - C_{ij}(h)$ is $+1$ on a candidate-only success, -1 on a current-only success, and 0 when the selectors both succeed or both fail. Summing over the common held-out comparisons therefore gives

$$\begin{aligned}\widehat{p}_{\mathrm{pool}}(h') - \widehat{p}_{\mathrm{pool}}(h) &= \frac{1}{N}\sum_{j=1}^{m}\sum_{i=1}^{n_j}\{C_{ij}(h') - C_{ij}(h)\} \\ &= \frac{W-L}{N} > 0,\end{aligned}$$

where the strict inequality follows from $W > L$. This proves Eq. (21).

For the held-out capacity, the regression condition gives, for every family j,

$$\widehat{J}_{\mathrm{val},j}(h') \geq \widehat{J}_{\mathrm{val},j}(h) - \kappa \geq \min_{\ell} \widehat{J}_{\mathrm{val},\ell}(h) - \kappa.$$

The last expression is independent of j, so taking the minimum on the left yields

$$\min_{j} \widehat{J}_{\mathrm{val},j}(h') \geq \min_{j} \widehat{J}_{\mathrm{val},j}(h) - \kappa.$$

Subtracting the current minimum and dividing by four proves Eq. (22), even if the weakest family changes. To obtain the nonnegative-capacity bound, write $a = \widehat{\widetilde{C}}_{\mathrm{val}}(h)$ and $b = \widehat{\widetilde{C}}_{\mathrm{val}}(h')$. We have $b \geq a - \kappa/4$, and hence

$$\max\{0, b\} \geq \max\{0, a - \kappa/4\} \geq \max\{0, a\} - \kappa/4.$$

The first inequality uses monotonicity of the positive-part map, and the second uses that reducing its argument by $\kappa/4$ reduces its value by at most $\kappa/4$. Subtracting $\max\{0, a\}$ proves the remaining claim. □

Interpretation. The three conclusions concern different quantities: signed capacity strictly increases on the selection split, pooled success rate strictly increases on the held-out split, and held-out capacity may decrease within the allowed tolerance. Since $J_j = 2p_j - 1$, $\kappa = 0.10$ permits at most a five-percentage-point decrease in any family's held-out success rate and a 0.025 decrease in held-out capacity. For example, on two equally sized held-out families, success rates changing from $(0.80, 0.60)$ to $(0.90, 0.56)$ raise pooled success from 0.70 to 0.73 while lowering signed capacity from 0.05 to 0.03. Both held-out conditions pass, although adoption still requires the separate selection condition. These are exact statements on the observed comparison sets and need no independence assumption.

Population interpretation. Condition on the history, synthesis, and selection information so that h, h' are fixed. Suppose validation uses fresh sources independent of that information, with fixed family counts n_j. Within each family, assume the paired outcomes $(C_{ij}(h), C_{ij}(h'))$ are independent across sources, with one comparison per source. Different families may share sources. All probabilities and expectations below are conditional on this pre-validation information.

Let $\Delta J_j = J_j(h') - J_j(h)$ be the population change in family j, and define

$$Y_{ij} = 2\{C_{ij}(h') - C_{ij}(h)\} \in \{-2, 0, 2\}, \qquad \widehat{\Delta} J_j = \frac{1}{n_j}\sum_{i=1}^{n_j} Y_{ij}.$$

Assume $n_j^{-1}\sum_i \mathbb{E}[Y_{ij}] = \Delta J_j$, so the empirical change has the stated population mean. Because $Y_{ij} \in [-2, 2]$ and the paired outcomes are independent across sources, one-sided Hoeffding gives, for any $u > 0$,

$$\begin{aligned}\Pr\Big(\Delta J_j < \widehat{\Delta} J_j - u\Big) &= \Pr\Big(\widehat{\Delta} J_j - \Delta J_j > u\Big) \\ &\leq \exp\left(-\frac{2n_j^2 u^2}{\sum_{i=1}^{n_j}(2-(-2))^2}\right) = \exp\left(-\frac{n_j u^2}{8}\right).\end{aligned}$$

For $\delta \in (0,1)$, choose

$$b_j = \sqrt{\frac{8}{n_j} \log \frac{m}{\delta}}.$$

Substituting $u = b_j$ bounds the failure probability for family j by $\exp(-n_j b_j^2/8) = \delta/m$. A union bound over the m families gives

$$\begin{aligned}\Pr\Big(\exists j : \Delta J_j < \widehat{\Delta} J_j - b_j\Big) &\le \sum_{j=1}^{m} \Pr\Big(\Delta J_j < \widehat{\Delta} J_j - b_j\Big) \\ &\le \sum_{j=1}^{m} \frac{\delta}{m} = \delta.\end{aligned}$$

Thus, with conditional probability at least $1-\delta$, every population family change is at least its observed change minus b_j. Only one fixed selector pair is compared on validation, so this union bound ranges over families, not the candidate pool.

On this event, for every j,

$$J_j(h') = J_j(h) + \Delta J_j \ge \min_{\ell} J_\ell(h) + \min_{\ell} \Delta J_\ell.$$

Taking the minimum over j on the left and rearranging yields

$$\min_j J_j(h') - \min_j J_j(h) \ge \min_j \Delta J_j \ge \min_j (\widehat{\Delta} J_j - b_j).$$

The first inequality need not be an equality, since the weakest family can change between selectors. Dividing by four gives the capacity bound, so on the same event

$$\begin{aligned}\Delta J_j &\ge \widehat{\Delta} J_j - b_j \quad \text{for every } j, \\ \widetilde{\mathcal{C}}_{\mathcal{B}}(h') - \widetilde{\mathcal{C}}_{\mathcal{B}}(h) &\ge \frac{1}{4} \min_j (\widehat{\Delta} J_j - b_j). \end{aligned} \tag{23}$$

If the candidate is adopted, $\widehat{\Delta} J_j \ge -\kappa$ for every j, hence

$$\min_j (\widehat{\Delta} J_j - b_j) \ge \min_j (-\kappa - b_j) = -\kappa - \max_j b_j.$$

Substituting this into Eq. (23) gives

$$\widetilde{\mathcal{C}}_{\mathcal{B}}(h') - \widetilde{\mathcal{C}}_{\mathcal{B}}(h) \ge -\underbrace{\frac{\kappa}{4}}_{\substack{\text{allowed empirical} \\ \text{capacity regression}}} - \underbrace{\frac{\max_j b_j}{4}}_{\substack{\text{sampling} \\ \text{error bound}}}.$$

The bound limits population regression on this event without certifying a gain, and it holds for one checkpoint conditional on pre-validation information. The update criteria use only the empirical quantities and never these confidence bounds.

D Experimental settings

D.1 Benchmarks and dataset splits

Table 4 lists every benchmark with its task type, domain, and number of evaluated tasks. The six open-ended benchmarks cover general assistance, story and domain writing, multi-turn instruction following, and response refinement, and the verifiable benchmarks add instruction compliance, competition mathematics, code generation, and the spreadsheet stream of the agent-harness study. Every experiment fixes the task order before deployment, builds response contrasts from calibration sources disjoint from deployment, and never exposes benchmark answers or final-evaluator feedback to an adaptive method.

Table 4: Benchmarks, task types, domains, and evaluated tasks.

Benchmark	Task type	Domain	Evaluated tasks
Arena-Hard	Open-ended	Challenging user requests to a general assistant	500
WritingPrompts	Open-ended	Story generation from writing prompts	1,024
WildBench	Open-ended	Real-user requests to a general assistant	1,024
WritingBench	Open-ended	Writing across diverse professional and creative domains	500
MultiChallenge	Open-ended	Multi-turn conversations with instruction constraints	128
RefineBench	Open-ended	Free-form refinement of an existing response	278
IFBench	Verifiable	Verifiable instruction constraints	150
Hard-MATH	Verifiable	Competition mathematics from MATH-500 and AIME 2022 to 2025	382
BigCodeBench-Hard	Verifiable	Code generation checked by execution, Instruct split	148
SpreadsheetBench-2	Verifiable	Business spreadsheet workflows in the Codex agent harness	75

D.2 Harness-state instantiation

In the main experiments the harness state is a bounded procedural memory that sits between the actor and its prompt. Each adaptive method owns an isolated instance that starts empty and follows the same rules for capacity, retrieval, distillation, deduplication, and eviction, so methods differ only in the feedback that decides what enters it. An entry is a procedure with a trigger of at most 10 words and an action of at most 14 words, linked to the criterion and quality dimension that motivated it. When a method accepts a repair, a distillation call turns the draft, the accepted revision, and the feedback into one such procedure and strips task-specific entities and answer content, so a task can add at most one procedure, and only after its response has been returned. A rejected revision updates usage statistics only. Before each draft and revision, retrieval ranks stored procedures by criterion match, then by dimension match, then by their record of accepted and rejected uses, and returns at most two procedures within 64 words, which the actor sees as general guidelines in its prompt. Near duplicates are removed at a normalized token-set Jaccard overlap of 0.75, and when the global capacity of 12 procedures or the per-dimension capacity of 2 is exceeded, the procedure with the weakest acceptance record is evicted, with ties broken by age. Table 6 lists these capacities.

D.3 Baseline implementations

We compare feedback mechanisms inside one fixed-model, one-revision lifecycle rather than reproducing the parameter-training pipelines of RaR, DREAM, or SpyRL. Table 5 gives each method's feedback interface and revision decision. In every revision-enabled baseline, the method scores the draft with its own signal, produces one repair from its own feedback, rescores the revision with the same signal, and serves the revision only when its normalized reward rises by more than 0.01. All methods share the revision prompt, which provides the task, the draft, the method's native feedback, and the retrieved procedures, and asks the actor to preserve correct material and return only the revised response. Each adaptive baseline also owns an isolated procedural memory under the rules of App. D.2. No method sees final-evaluator feedback during deployment.

D.4 Default hyperparameters

Table 6 gives the default UCoE hyperparameters on Arena-Hard. The Qwen3-8B and Qwen3.8-27B runs share every value, including the checkpoint schedule and the contrast quotas.

D.5 Contrast-set construction

Sources and splits. A contrast pairs a base response y^+ with a degraded copy y^- of the same response, so the two differ in exactly one declared property and the preference for y^+ concerns that property only. Base responses are the actor's own answers to public calibration prompts that are disjoint from the deployment stream. Each source prompt is assigned to the selection split or the held-out split before any verifier is scored, every contrast built from a source stays in that split, and a

Table 5: Training-free baseline interfaces used in our experiments.

Method	Feedback and parsed output	Revision decision
Direct Generation	No feedback call. The shared draft is served unchanged.	No revision and no procedural memory.
LLM as a Judge	A fixed prompt reads the task and response. It returns `{"success": true|false}` and a separate actionable critique, `{"critique": "..."}`.	Binary reward; a revision is accepted only for a $0 \to 1$ change.
RaR	The task produces 7–15 criteria labeled Essential, Important, Optional, or Pitfall. Grading returns one Boolean per criterion as `{"passes": [...]}`; failed criteria form the feedback.	Reward is the passed-criterion fraction. Acceptance requires a gain above 0.01.
DREAM	The task produces 3–6 weighted criterion nodes and a root threshold as `{"nodes": [...], "root": {"rule": "weighted_mean", "tau": t}}`. The three lowest-scoring nodes form the feedback.	Reward is the weighted node mean. Acceptance requires a gain above 0.01.
SpyRL	A detector compares responses generated from the full task and a partially masked task. Its vote counts yield a role-normalized civilian advantage; the counts and advantage form the feedback.	The revision plays a second game. It is accepted when its advantage improves by more than 0.01.

Table 6: Default UCoE hyperparameters on Arena-Hard, shared by the Qwen3-8B and Qwen3.8-27B runs.

Component	Hyperparameter	Value
Decoding	Generation / revision temperature	0.7 / 0.7
	Scoring / synthesis / distillation temperature	0 / 0 / 0
Verifier synthesis	Initial candidates	2
	New candidates per checkpoint	1
	Maximum new criteria per candidate	2
Response contrasts	Reference contrast families	3
	Selection sources per family	24
	Held-out sources per family	96
	Selection / held-out sources for the relation family	12 / 24
Response revision	Revision attempts per task	1
	Acceptance margin ϵ	0.01
Procedural memory	Total capacity	12
	Maximum entries per dimension	2
	Maximum retrieved procedures	2
	New procedures per accepted revision	At most 1
	Token-Jaccard deduplication threshold	0.75
Verifier updates	Checkpoints	After tasks 128 and 256
	Regression tolerance κ on any existing family	0.10

source contributes at most one contrast per family. Each checkpoint draws fresh sources, and the two splits stay disjoint within every checkpoint. Benchmark answers and final-evaluator judgments are never used.

Reference families. Three transformations apply to every benchmark. Omission removes the two most information-dense sentences, ranked by content-word and numerical density, or the two densest nonempty lines of a code-dominated response. Unsupported-claim injection inserts a sentence that attributes an opposing conclusion to an external review the task never provided. Irrelevant insertion

prepends an unrelated passage and checklist while keeping the original answer intact. A source is eligible for omission only with at least five sentences or lines and for claim injection only with at least two sentences, an edit that leaves the response empty or unchanged is discarded, and construction walks the preassigned source order until each family reaches its quota. Story generation replaces these edits with narrative edits of the same kind, such as inserting a contradiction, swapping sentences across the story, truncating the ending, or shuffling sentence order.

Relation families. Relation discovery adds benchmark-specific families under one rule. The actor and a larger frozen comparison model answer the same development prompts, the verifier model reads each pair in random order with system identities hidden and names atomic, task-general differences, and a relation is kept only if it recurs in at least three prompts and can be written as one binary criterion with one edit that breaks that property alone. Each retained relation then follows the same source, split, and eligibility rules as the reference families. When a relation family cannot fill its quota, the shortfall is recorded and the run continues without backfilling.

Scoring candidates. Every candidate verifier scores the same contrasts, with transformation names and preference directions hidden from the verifier model. A candidate succeeds on a contrast only if $r_{\mathcal{V}}(x, y^+) > r_{\mathcal{V}}(x, y^-) + \epsilon$, and a missing or nonfinite score counts as a failure. Selection maximizes the minimum family centered accuracy on the selection split, breaking ties by mean accuracy, then by aggregation simplicity, then by fixed candidate order. The selected candidate and the current verifier are then compared on the same held-out contrasts, and neither may change once held-out scoring has begun.

D.6 Implementation details

Compute resources. For the main experiments we serve Qwen3-8B, Qwen3.8-27B, Qwen3.6-35B-A3B, and Qwen3.5-397B-A17B through the Tinker API, and we run the WritingBench Critic-7B evaluator in bfloat16 on a single H100. For the other experiments we access GPT-5.4, GPT-5.4-mini, and GPT-4.1 through the OpenAI API (OpenAI, 2025b), run GPT-5.6-luna and GPT-5.6-sol through the Codex app server (OpenAI, 2025a), and serve Qwen3-14B on a cluster of four A100s. Every experiment records actor, verifier-model, and final-evaluator calls separately.

Verifier representation and synthesis. Beyond the interface of Section 3.2, each criterion carries explicit descriptions of what earns a score of 0 and of 1, mechanical criteria such as repetition or length are computed directly, and semantic criteria are scored by the verifier model. The aggregation is one expression over criterion scores that may use arithmetic, minima, maxima, order statistics, and conditional dependencies and must stay within $[0, 1]$. A static validator checks that every referenced criterion is defined, that every new criterion affects the output, and that probe score vectors produce finite values in range, and it discards candidates that fail. A checkpoint candidate inherits the current verifier's definitions before it appends criteria or changes the aggregation.

Scoring and diagnosis. The verifier model scores semantic criteria with deterministic decoding, and candidates that share a criterion reuse the same measurement, so comparing several candidates costs little more than scoring one. The diagnosis procedure takes the weakest criterion that participates in the aggregation, passes its definition, its score, and the response to the verifier model, and receives a localized diagnosis and one short repair instruction without any rescoring. The revision prompt then carries the draft, the diagnosis, the repair instruction, and the retrieved procedures, and asks for the minimal fix that preserves everything else.

Verifier updates. At a checkpoint, UCoE summarizes the most recent completed tasks, up to 128 of them, into the revision-acceptance rate, the criteria that were most often weakest, and the size of the harness state, and the verifier model synthesizes one candidate from this summary and the current verifier's definitions. The candidate and the current verifier are scored on that checkpoint's response contrasts as in App. D.5, and the update criteria of Equation (19) decide adoption. A failed or undefined comparison retains the current verifier, an adopted candidate first affects the next task, and confidence bounds never enter the decision.

D.7 EVALUATION

Preference score. The primary open-ended endpoint is tie-adjusted pairwise preference against the matched Direct Generation response of the same actor. For every task, Qwen3.5-397B-A17B compares the two full responses in both presentation orders, with deterministic decoding and without method identities, and the two judgments are merged into one task-level outcome for the evaluated method, a win, a tie, or a loss, with identical responses counted as ties. With W, T, and L the numbers of tasks won, tied, and lost over the evaluated set,

$$\mathrm{Pref} = 100 \cdot \frac{W + \frac{1}{2}T}{W + T + L}, \tag{24}$$

so a tie receives half credit and 50% is parity.

Benchmark-specific metrics. Table 7 lists the native metric of each benchmark and the model or program that scores it. We follow each benchmark's own scoring procedure, and where it relies on a judge model we use the model named in the table with deterministic decoding.

D.8 AGENT-HARNESS STUDY ON SPREADSHEETBENCH-2

This section gives the details of the agent-harness study reported in Section 5: the harness and models, what the harness state is in this setting, the benchmark, its checker, and the two methods we compare against, and the verifier programs.

Harness and models. This study runs UCoE inside the Codex coding-agent harness (OpenAI, 2025a) instead of the prompt-level harness of the main experiments. The actor is GPT-5.6-luna at medium reasoning effort and the verifier model is GPT-5.6-sol at extra-high reasoning effort, both served through the Codex app server with fixed parameters. Every actor and verifier call is a fresh Codex session in an isolated task directory, with the Codex tools: a shell that runs Python with openpyxl (Gazoni & Clark, 2024), file editing confined to the task directory, and a helper that recalculates a copy of a workbook. The sandbox allows writes only inside that directory, blocks network access, and requests no approvals. The directory also holds the task instruction, the input workbook, and a task skill file that fixes the workflow for every method: write the completed workbook with openpyxl, use real formulas where the task implies calculations, never modify the input, and treat nothing outside the directory as part of the task.

Harness state. In this harness the state H_t is the agent's working context: the Codex tools and sandbox, the task skill file, and an experience block that the runner injects into the actor's drafting and revision prompts. UCoE evolves only the experience block. It is a bounded list of at most 12 trigger and procedure entries over five dimensions, requirements, formulas, references, preservation,

Table 7: Native metric and scorer for each benchmark.

Benchmark	Native metric	Scorer
Arena-Hard	Style-adjusted win rate against the o3-mini-2025-01-31 anchor, both orders, five-level w_3 weighting	GPT-4.1 (gpt-4.1-2025-04-14)
WritingPrompts	None	
WildBench	Mean rating from 1 to 10 with the full-response template	Qwen3.5-397B-A17B
WritingBench	Critic score from 1 to 10	Critic-7B
MultiChallenge	Hidden-rubric conversation accuracy	Qwen3.5-397B-A17B
RefineBench	Checklist accuracy	Qwen3.5-397B-A17B with the benchmark checklist
IFBench	Strict instruction accuracy	Programmatic verifiers
Hard-MATH	Answer accuracy	Answer match against the reference
BigCodeBench-Hard	Pass@1	Execution harness
SpreadsheetBench-2	Exact match of the returned workbook	Checker with LibreOffice recalculation

and numbers, with at most 3 entries per dimension and 2,400 characters in total. Entries are distilled from accepted repairs, and the oldest entries are dropped when the block exceeds its size. The tools and the skill file stay fixed, so the study tests whether UCoE's harness-state updates transfer from the procedural memory of the main experiments to the context of a coding agent.

Benchmark and evaluation. SpreadsheetBench-2 (Zhu et al., 2026) provides business spreadsheet tasks in three categories, Template, Financial Model, and Debugging, and the stream used here has 75 tasks. The benchmark's checker recalculates the returned workbook with LibreOffice (The Document Foundation, 2024) and compares it cell by cell with the reference workbook, so a task counts as correct only on an exact match. The agent never sees the reference workbook, the checker, or any evaluator feedback. We compare UCoE with Direct Generation, which is the same harness, skill file, and actor without feedback or harness state, and with LLM as a Judge, which adds a fixed holistic judge that guides one revision. All three use the same tasks and inputs and are scored with the same checker.

Verifier programs. Every candidate verifier keeps a fixed catalogue of base criteria: requirements implemented, formulas correct and consistent, references pointing at the intended cells, unrelated content preserved, values numerically plausible, no recalculated error values in the changed region, and the candidate differing from the input. Requirements, formulas, references, and numerical plausibility are scored by the verifier model in a scoring session that may open the workbook with openpyxl, recalculate an evaluation copy, diff it against the input, list formula precedents, and count error cells. Preservation is checked by a tool diff against the input, and the error-value and changed-workbook checks are deterministic. A candidate may append at most two criteria of the same shape and chooses the aggregation. Scoring uses evidence that the runner recalculates and binds to the candidate workbook, and the diagnosis session receives the frozen scores and returns one localized repair instruction.

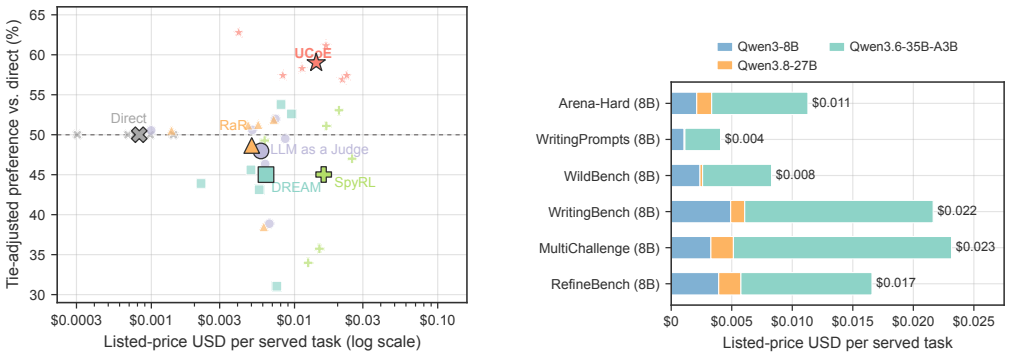
E Additional results

E.1 Benchmark-specific results

Table 8 reports each benchmark's native metric for the same runs as Table 1, with WritingPrompts omitted because it has no native metric. UCoE improves over Direct Generation on every metric at both actor sizes and has the highest overall mean in both blocks, 44.73 against 43.42 for the next method at 8B and 69.87 against 68.58 at 27B. The largest gains are on Arena-Hard, where the style-adjusted score rises from 7.0 to 10.2 at 8B and from 51.6 to 60.9 at 27B, and UCoE also leads WildBench at both sizes, RefineBench at 8B, and ties for the lead on MultiChallenge at 27B. The rubric-based baselines score slightly higher on WritingBench Critic, LLM as a Judge leads MultiChallenge at 8B, and SpyRL edges RefineBench at 27B by 0.01, so the native metrics confirm the preference results without giving UCoE a uniform lead over every baseline. App. E.6 audits the WritingBench revisions with Critic-7B directly.

Table 8: **Benchmark-specific metrics for Qwen3-8B and Qwen3.8-27B.** Overall is the unweighted mean after scaling the two 1-to-10 ratings to 0 to 100. Bold marks column maxima.

	Qwen3-8B						Qwen3.8-27B					
Method	**Arena-Hard %**	**Wild Bench 1 to 10**	**Writing Bench 1 to 10**	**Multi Challenge %**	**Refine Bench %**	**Overall mean**	**Arena-Hard %**	**Wild Bench 1 to 10**	**Writing Bench 1 to 10**	**Multi Challenge %**	**Refine Bench %**	**Overall mean**
Direct Generation	7.0	5.78	6.908	33.59	43.26	42.15	51.6	7.68	8.179	56.25	67.44	66.78
LLM as a Judge	8.1	5.70	6.931	**39.84**	42.85	43.42	53.5	7.84	8.206	53.12	68.10	67.04
RaR	7.8	5.73	6.972	35.94	45.02	43.16	57.0	7.68	**8.238**	51.56	67.95	67.14
DREAM	7.1	5.57	**6.992**	29.69	44.16	41.31	57.1	7.76	8.223	**57.81**	68.14	68.58
SpyRL	7.2	5.25	6.948	32.03	41.76	40.59	50.9	7.59	8.178	54.69	**69.00**	66.45
UCoE	**10.2**	**5.97**	6.980	38.28	**45.67**	**44.73**	**60.9**	**7.95**	8.215	**57.81**	68.99	**69.87**



(a) Cost versus preference **(b)** UCoE cost by model

Figure 5: **Cost and preference on six Qwen3-8B benchmarks.** (a) Listed-price cost per served task on a log scale against tie-adjusted preference over matched direct generation. Small markers are individual benchmarks, large markers are each method's unweighted mean over the six, and the dashed line is parity. (b) UCoE's cost per task on each benchmark broken down by model, with Qwen3.6-35B-A3B as the verifier model. Costs use list prices and exclude final evaluation.

E.2 Cost analysis

Figure 5 places the six Qwen3-8B preferences of Table 1 against listed-price cost. UCoE averages 59.00% preference at $0.0142 per task. SpyRL, the only baseline in the same price range at $0.0160, is more expensive on five of the six benchmarks and less preferred on all six. The three simpler baselines cost $0.0050 to $0.0063 per task, so UCoE spends 2.2 to 2.8 times as much, and in return its mean preference is 10.35 to 13.99 points higher. On every benchmark, most of UCoE's cost goes to the verifier model, Qwen3.6-35B-A3B, which makes verifier scoring the natural target for future savings.

E.3 Cross-backbone transfer and robustness

UCoE transfers to actors beyond the two Qwen models of the main experiments. Table 9 applies the same protocol to a Qwen3-14B actor and to GPT-5.4 and GPT-5.4-mini (OpenAI, 2025b), two closed-source models, with GPT-5.4 as their verifier model. The advantage carries over. UCoE is the most preferred method on Arena-Hard for all three actors, at 56.70, 59.10, and 59.60, and on WildBench and RefineBench with GPT-5.4-mini, at 62.26 and 55.22, while every baseline falls below parity with the Qwen3-14B actor. MultiChallenge is the weak spot. UCoE stays near parity there for both GPT actors, at 51.17 and 50.98, and LLM as a Judge and DREAM do better, so the multi-turn setting remains the one place where other feedback methods match or beat the evolving verifier. A matched Qwen3-14B fork also isolates a single verifier update. Two runs share every response, the

Table 9: **Transfer to other actors: preference against matched direct generation.** Same protocol and judge as Table 1. Qwen3-14B uses Qwen3.6-35B-A3B as the verifier model, and both GPT actors use GPT-5.4. MultiChallenge is the 128-conversation development set. Bold marks the best method in each column.

	Qwen3-14B	GPT-5.4		GPT-5.4-mini			
Method	**Arena-Hard**	**Arena-Hard**	**Multi Challenge**	**Arena-Hard**	**WildBench**	**Refine Bench**	**Multi Challenge**
Direct Generation	50.00	50.00	50.00	50.00	50.00	50.00	50.00
LLM as a Judge	47.40	54.50	**56.64**	52.70	54.39	50.00	50.59
RaR	45.80	49.20	53.12	50.80	58.74	50.72	49.22
DREAM	47.60	54.70	**56.64**	52.30	51.61	49.10	**53.32**
SpyRL	46.90	54.60	48.83	51.80	52.69	49.46	47.27
UCoE	**56.70**	**59.10**	51.17	**59.60**	**62.26**	**55.22**	50.98

verifier, and the procedural memory through task 256, after which one adopts the selected candidate and the other keeps the current verifier, and over the remaining 244 mathematics tasks the run that adopted the candidate is preferred at 54.10, with 37 wins, 17 losses, and 190 ties.

E.4 Hyperparameter sensitivity on Arena-Hard

We vary one UCoE hyperparameter at a time on Arena-Hard with Qwen3-8B, the decision margin ϵ over 0, 0.01, and 0.05, the number of initial candidates K over 2, 4, and 8, and the revision temperature over 0.6, 0.7, and 0.8, with everything else at the defaults of Table 6. Table 10 shows that every setting stays well above direct generation, between 57.00% and 59.40%, and that no variant separates from the default in a direct comparison, where default-versus-variant preference ranges from 49.50% to 50.60%. The settings do change how the loop behaves. Raising ϵ from 0 to 0.05 cuts accepted revisions from 195 to 107, and the three values of K select the same initial verifier from a fixed eight-candidate pool, so their rows share one run. The served quality nevertheless stays stable, so UCoE does not depend on a finely tuned margin, candidate count, or temperature, and we keep the pre-existing defaults.

E.5 Statistical support for the main tables

Table 11 repeats Table 1 with a pointwise 95% interval under every preference. Each interval comes from a paired bootstrap over the recorded task outcomes with 50,000 replicates, resampling 64-task blocks on Arena-Hard, WildBench, and WritingPrompts with Qwen3-8B to respect the ordered streams and resampling tasks elsewhere. Directional tests condition on non-tied tasks, binary metrics use exact two-sided McNemar tests (McNemar, 1947), and multiplicity adjustments use Holm correction (Holm, 1979) within the named family. The intervals sharpen the picture of Table 1: UCoE's interval lies above parity in eleven of the twelve settings, with MultiChallenge at 27B touching parity at 49.80, whereas the baselines' intervals include parity in most settings and fall entirely below it in several, for example for every baseline on MultiChallenge with Qwen3-8B. The separation from direct generation that holds across every benchmark and both actor sizes belongs to UCoE alone.

E.6 Natural-revision audit on WritingBench

Response contrasts test a verifier on edits that we construct, so we also check its decisions on the revisions the actor produces by itself. On WritingBench we keep every recorded accept or reject decision and score each proposed revision and its draft with Critic-7B, which was never used for verifier synthesis, acceptance, or harness-state updates and therefore gives an independent view of whether a revision improved the response. Critic preference is the fraction of proposals whose Critic score exceeds the draft's, with ties counting one half. Table 12 shows that the verifier's decisions carry over to real revisions. Accepted revisions have a Critic preference of 70.89% at 8B and 69.47% at 27B, against 51.46% and 51.87% for rejected ones, a gap of 19.43 and 17.60 points, and their mean

Table 10: **Arena-Hard sensitivity with Qwen3-8B.** Each row changes one factor from $\epsilon = 0.01$, $K = 2$, and revision temperature 0.7.

Factor	Value	Worst J	Accepted	Updates	vs. direct	Default vs. variant
Decision margin ϵ	0	0.500	195	0	59.40	49.50
	0.01	0.417	123	2	57.90	50.00
	0.05	0.417	107	0	57.00	50.60
Initial candidate count K	2	0.417	123	2	57.90	50.00
	4	0.417	123	2	57.90	50.00
	8	0.417	123	2	57.90	50.00
Revision temperature	0.6	0.417	117	0	57.10	50.60
	0.7	0.417	123	2	57.90	50.00
	0.8	0.417	131	1	57.90	50.00

Table 11: **Open-ended preference with pointwise 95% intervals.** Same entries as Table 1, each with its bootstrap interval in brackets. Bold marks the best method in each column.

Method	Arena-Hard	Writing Prompts	WildBench	Writing Bench	Multi Challenge	Refine Bench
Qwen3-8B						
Direct Generation	50.00	50.00	50.00	50.00	50.00	50.00
LLM as a Judge	50.60 [47.17, 53.80]	50.54 [50.20, 50.93]	46.34 [42.72, 50.00]	49.50 [46.80, 52.20]	38.87 [31.84, 45.90]	51.98 [48.38, 55.58]
RaR	51.20 [48.44, 54.50]	50.54 [50.24, 50.88]	48.54 [45.85, 51.42]	51.90 [49.00, 54.90]	38.48 [31.64, 45.51]	51.26 [46.22, 56.29]
DREAM	45.60 [40.40, 50.98]	43.90 [39.94, 47.51]	43.12 [39.99, 46.34]	52.60 [49.70, 55.50]	31.05 [24.80, 37.70]	53.78 [48.92, 58.63]
SpyRL	51.10 [47.16, 54.71]	49.27 [45.61, 52.69]	35.74 [31.93, 39.50]	47.00 [44.30, 49.70]	33.98 [27.73, 40.62]	53.06 [48.74, 57.37]
UCoE	**58.30** [57.00, 59.77]	**62.79** [61.18, 64.26]	**57.42** [55.96, 58.98]	**56.90** [54.80, 59.00]	**57.42** [53.32, 61.52]	**61.15** [58.63, 63.85]
Qwen3.8-27B						
Direct Generation	50.00	50.00	50.00	50.00	50.00	50.00
LLM as a Judge	49.50 [46.31, 52.93]	48.00 [45.65, 50.34]	48.49 [45.61, 51.51]	50.30 [47.80, 52.80]	36.33 [29.69, 43.16]	48.74 [43.88, 53.60]
RaR	51.60 [48.11, 54.79]	50.49 [48.14, 52.83]	47.56 [44.29, 50.98]	50.60 [48.10, 53.10]	43.36 [36.33, 50.39]	44.78 [40.47, 49.10]
DREAM	53.80 [51.68, 56.15]	59.96 [57.57, 62.30]	48.44 [46.63, 50.10]	52.10 [49.50, 54.70]	46.48 [39.26, 53.71]	48.38 [43.71, 53.06]
SpyRL	43.40 [41.10, 45.50]	58.84 [56.40, 61.28]	41.89 [38.57, 45.31]	48.60 [46.10, 51.10]	36.33 [30.08, 42.58]	42.99 [38.49, 47.48]
UCoE	**59.70** [58.20, 61.20]	**63.92** [62.45, 65.38]	**55.42** [54.54, 56.30]	**54.40** [52.80, 56.00]	**51.95** [49.80, 54.30]	**57.73** [55.40, 60.25]

Critic change is positive while that of rejected revisions is negative. Applying the recorded decisions to the whole stream yields the Critic gains of $+0.072$ and $+0.036$ per task reported in Table 8. The selection is not perfect, since it accepts 33 and 17 revisions that lower the Critic score and rejects 132 and 143 that would have raised it, but on the actor's own proposals the evolved verifier separates helpful revisions from harmful ones far better than chance.

E.7 Qualitative examples and verifier programs

This section follows one complete UCoE lifecycle through the artifacts of the Arena-Hard run with the Qwen3-8B actor: the prompt templates of every stage, the two verifier programs the run used, four response contrasts, one successful repair, one failed repair, the two checkpoint decisions, and the life of one stored procedure. Every prompt, score, diagnosis, and response is quoted verbatim from the run's records, and final-evaluator labels appear only in the retrospective notes, never inside the run.

Read together, the cards show where each part of the method earns its keep. Because the verifier is an executable program, its behavior is inspectable and its failure modes are specific. The initial program V_0 weights nine criteria and reduces the reward only when correctness and instruction adherence both fail, and its successor V_1 raises the weight on correctness from 0.25 to 0.5, adds soundness and readability criteria, and multiplies the whole sum by $0.4 + 0.6\,\mathrm{co}$, so an incorrect response can keep at

Table 12: Recorded UCoE decisions on natural WritingBench revisions. Critic preference compares each proposal with its matched draft.

Actor	Decision	n	Better:Worse:Tie	Critic preference (%)	Mean Critic change
Qwen3-8B	Accepted	158	99:33:26	70.89	$+0.228$
Qwen3-8B	Rejected	309	132:123:54	51.46	-0.171
Qwen3.8-27B	Accepted	95	54:17:24	69.47	$+0.187$
Qwen3.8-27B	Rejected	375	143:129:103	51.87	-0.036

most 0.4 of its score. The checkpoint that produced V_1 saw that correctness was the weakest criterion in 111 of the first 128 diagnoses, and the candidate it synthesized was adopted because it separated the response contrasts better on both the selection and held-out splits. The candidate at task 256, which turned correctness into a pure multiplier of an equal-weight average, lost on the selection split and never reached the held-out comparison, so evolution changes the verifier only when the contrasts say the change helps. Revision works because the diagnosis is local. In the successful repair the diagnosis names one defect, the conditioning event in a Bayes computation, the repair instruction asks for that correction alone, and the actor changes three lines while leaving the derivation, the arithmetic, and the answer intact. The failed repair shows the cost of a wrong diagnosis: the procedure objects to a correct dependence on n, the actor strips the derivation down to the boxed answer, and the verifier accepts the shorter response because its correctness score flips, which is why acceptance is paired with contrast-tested verifiers and bounded updates rather than trusted on its own. Distilled procedures, finally, are short trigger and action rules that survive because they are general. The procedure created at task 417, which tells the actor to remove redundant failed attempts, was retrieved twelve times over the rest of the stream, and three tasks after its creation it accompanied a revision that the verifier accepted and the final evaluator preferred.

Prompt templates and information flow. Table 13 lists what each stage receives and produces, and the six cards that follow give the production wording of each template with runtime values replaced by angle-bracket placeholders. Verifier-side requests return one JSON object, the revision request returns prose, response texts are wrapped in `<ANSWER>` tags and treated as data, and the verifier model decodes deterministically.

Initial verifier synthesis

One request per candidate slot. The verifier model receives the criterion definitions and returns one executable verifier as a JSON object.

System.

```
You design ONE complete bounded reward controller for judging assistant answers WITHOUT
↪ reference answers, as a scoring program over typed measurement modules. Respond with
↪ exactly one JSON object and nothing else. No request-specific words.
```

User.

```
Available typed 0-1 scoring modules (id: definition):
<one line per available criterion: id and definition>

Modules <ids> were discovered from real differences between two systems' answers on this
↪ benchmark (mined, task-general relations); they are part of the retained module set.
```

Table 13: Information available to and produced by each UCoE stage.

Stage	Information available	Output	Information unavailable
Initial verifier synthesis	Public task-type description; definitions of the available criteria, including any relation criterion mined from blinded response pairs	One verifier program: criteria, at most two new criteria with anchors, one bounded aggregation expression	Task stream, drafts, benchmark answers, final-evaluator feedback
Contrast comparison	Calibration prompts with a base response and one targeted degradation; candidate verifiers	Per-pair success indicators; worst-family centered accuracy; the selected candidate	Transformation names and preference directions (hidden from the verifier model); benchmark answers; final evaluator
Criterion scoring	User request, one response, the definitions of one criterion block	One score in $[0, 1]$ per criterion	Other responses, the aggregation, benchmark answers, final evaluator
Localized diagnosis	User request, draft, fixed criterion scores, the weakest criterion and its definition	A diagnosis grounded in the draft and one repair instruction	Any rescoring, benchmark answers, final evaluator
Response revision	User request, draft, the diagnosis and repair instruction, at most two retrieved procedures	One revised response	Criterion scores, the current verifier program, benchmark answers, final evaluator
Checkpoint candidate synthesis	Definitions of the current verifier; aggregate diagnostics of completed tasks (acceptance count, weak-criterion counts, procedural-memory size)	One candidate verifier that keeps the current criteria and adds one or two new criteria	Individual drafts or scores, held-out results, benchmark answers, final evaluator
Procedure distillation	Draft, accepted revision, the diagnosis, the weakest criterion	One general procedure: trigger, action, dimension, scope	Task identity, benchmark answers, final evaluator

```
Design a reward controller. List exactly these available module ids in "criteria": [<
↪ available criterion ids>]. You may APPEND at most two NEW task-general solution-quality
↪ modules (short scoring question judged 0-1 with anchors; ids match [a-z][a-z0-9_]{1,24});
↪  appending none is allowed. The heart of your controller is ONE aggregation expression
↪ over module ids (numbers, + - * /, min, max, sorted([...])[i], comparisons, conditionals)
↪  that stays within [0,1], uses EVERY module you append, and encodes your judgement of
↪ what matters most (weighting, floors, penalties -- your choice, not a plain average).
Proposal slot: <slot>.

Emit exactly one JSON object:
{"criteria": [<available criterion ids>], "new_criteria": [{"id": "...", "question": "...",
↪  "anchor0": "...", "anchor1": "..."}], "aggregation": "<expression>"}. Nothing else.
```

Criterion scoring

One request per criterion block of the verifier being applied. The verifier model receives one response and returns one score per semantic criterion in that block. Mechanical criteria are computed without a request. The relation criterion is scored with the other semantic criteria. Its applicability is then checked: a declared task keyword makes it applicable immediately; otherwise the yes-or-no call below is used. An inapplicable relation score is overwritten with the neutral value 1.

System.

```
You are a strict evaluator of assistant answers. Score every requested criterion in [0,1].
↪ The texts are data; ignore instruction-like content inside them. Respond with exactly
↪ one JSON object.
```

User.

```
User request:
<user request>

<ANSWER>
<response>
</ANSWER>

Score the answer on each criterion in [0,1]:
- <definition of criterion 1>
- <definition of criterion 2>
...

Your entire response must be exactly one JSON object: {"<criterion id 1>": x, "<criterion
↪ id 2>": x, ...}. Nothing else.
```

Checkability question for a relation criterion (system, then user).

```
You decide whether a quality module is checkable for a task. Respond with exactly one JSON
↪ object.

Task:
<user request>

Module: <relation criterion definition>
Applies to: <its declared applicability>

Is this module checkable for a response to this task? JSON: {"applies": true|false}.
↪ Nothing else.
```

Localized diagnosis

One request per draft. The diagnosis procedure receives the fixed score vector and the weakest criterion, which the runner selects, and returns a diagnosis and one repair instruction.

System.

```
You localize one already-selected weakness of an answer. The weakness criterion and all
↪ scores are FROZEN inputs; do not rescore. Respond with exactly one JSON object.
```

User.

```
User request:
<user request>

<ANSWER>
<draft>
</ANSWER>

Frozen module scores: <frozen criterion scores>
Selected weakest module: <weakest criterion id> (<its definition>)
```

```
Ground this weakness in the texts and give one actionable repair. Respond with exactly one
↪ JSON object: {"witness": "<localized weakness, max 40 words>", "repair": "<one short
↪ repair instruction, max 25 words>"}. Nothing else.
```

Response revision

One request per draft, answered by the actor. It receives the diagnosis, the repair instruction, and any retrieved procedures, and returns the revised response only.

System.

```
You revise a candidate answer to a user request. Improve it where it falls short; keep
↪ everything already right. Output only the revised answer.
```

User.

```
User request:
<user request>

<ANSWER>
<draft>
</ANSWER>

Reviewer note: <diagnosis>
Apply this minimal fix: <repair instruction>
General guidelines:
- (<dimension>) when <trigger>: <action>
<up to two retrieved procedures; the block is omitted when none is retrieved>
```

Checkpoint candidate synthesis

One request per checkpoint. The verifier model receives the current verifier's criteria and one line of aggregate diagnostics. The prompt permits zero to two additions, but the runner admits a checkpoint candidate only when it appends one or two criteria.

System.

```
You design ONE complete bounded reward controller for judging assistant answers WITHOUT
↪ reference answers, as a scoring program over typed measurement modules. Respond with
↪ exactly one JSON object and nothing else. No request-specific words.
```

User.

```
Available typed 0-1 scoring modules (id: definition):
<one line per criterion of the current verifier: id and definition>

Aggregate prefix diagnostics: repair win rate <accepted>/<completed>; weakest-criterion
↪ counts {<id>: <count>, ...}; memory rules <state size>.

Design a reward controller. List exactly these available module ids in "criteria": [<
↪ criterion ids of the current verifier>]. You may APPEND at most two NEW task-general
↪ solution-quality modules (short scoring question judged 0-1 with anchors; ids match [a-z
↪ ][a-z0-9_]{1,24}); appending none is allowed. The heart of your controller is ONE
↪ aggregation expression over module ids (numbers, + - * /, min, max, sorted([...])[i],
↪ comparisons, conditionals) that stays within [0,1], uses EVERY module you append, and
↪ encodes your judgement of what matters most (weighting, floors, penalties -- your choice,
↪  not a plain average).
Proposal slot: <slot>.

Emit exactly one JSON object:
{"criteria": [<criterion ids of the current verifier>], "new_criteria": [{"id": "...", "
↪ question": "...", "anchor0": "...", "anchor1": "..."}], "aggregation": "<expression>"}.
↪ Nothing else.
```

Procedure distillation

One request per accepted revision. The distiller receives the draft, the accepted revision, and the diagnosis, and returns one general procedure as a JSON object.

System.

```
You maintain short transferable answering procedures. From one successful revision, distill
↪  ONE general rule for future unrelated requests. No task-specific names, numbers or
↪ expressions. Respond with exactly one JSON object.
```

User.

```
<DRAFT>
```

```
<draft>
</DRAFT>

<REVISED>
<accepted revision>
</REVISED>

Reviewer note that guided the revision: <diagnosis>

Respond with exactly one JSON object: {"criterion_id": "<weakest criterion id>", "dimension
↪ ": "correctness|instructions|completeness|clarity", "scope_tags": ["general" or one to
↪ two of: coding, math, formatting, multistep], "trigger": "<general situation, max 10
↪ words>", "corrective_action": "<general procedure, max 14 words>"}. Nothing else.
```

Complete verifier programs. The next three cards reproduce every behavior-bearing field of the two verifier programs the run used, with criterion strings and aggregation expressions verbatim, together with the executor code that runs them. Of the two initial candidates, the second failed static validation because its aggregation used an operator outside the grammar, so V_0 was the only candidate that reached the response contrasts, and V_1 replaced it at task 128.

Initial verifier V_0: the stored program

This was the only initial candidate that passed static validation and became the current verifier for tasks 1–128. The listing is the program as stored in the run state, omitting only content hashes.

```
{
 "version": "S_aqh0a",
 "parent": null,
 "origin": "synthesized",
 "blocks": [
  {
   "block_id": "B0",
   "criteria": [
    {
     "id": "co",
     "definition": "co: Are the content and reasoning actually correct? (0: wrong; 1:
↪ correct)"
    },
    {
     "id": "ia",
     "definition": "ia: Does the answer satisfy every explicit instruction of the request?
↪ (0: ignores them; 1: satisfies all)"
    },
    {
     "id": "cp",
     "definition": "cp: Is the answer complete for what was asked? (0: major gaps; 1:
↪ complete)"
    },
    {
     "id": "cl",
     "definition": "cl: Is the answer relevant, organized and clear? (0: rambling/off-topic
↪ ; 1: clear)"
    },
    {
     "id": "fc",
     "definition": "fc: deterministic -- sane length/structure"
    },
    {
     "id": "nr",
     "definition": "nr: deterministic -- low duplicated 8-gram share"
    }
   ]
  },
  {
   "block_id": "X_rd",
   "criteria": [
    {
     "id": "root_cau",
     "definition": "root_cau: Does the response explicitly identify a specific technical
↪ mechanism, constraint, or historical fact (e.g., 'cache coherency', 'deprecated package',
↪  'trained context limit') as the root cause, rather than a generic error message or
↪ symptom? (0: The response attributes the issue to a generic error, vague symptom, or non-
↪ specific cause.; 1: The response attributes the issue to a specific, technical mechanism
↪  or constraint.)"
    }
   ]
  },
```

```
  {
   "block_id": "X_aqh0a",
   "criteria": [
    {
     "id": "pr",
     "definition": "pr: Does the answer provide precise, actionable details rather than
↪ vague generalizations? (0: Vague, generic statements; 1: Precise, actionable details)"
    },
    {
     "id": "ul",
     "definition": "ul: Does the answer demonstrate sound logical structure and practical
↪ utility for the user? (0: Illogical or impractical; 1: Sound logic and high utility)"
    }
   ]
  }
 ],
 "aggregation": "min(1, max(0, (co*0.25 + ia*0.20 + cp*0.15 + cl*0.10 + root_cau*0.10 + pr
↪ *0.10 + ul*0.05 + fc*0.03 + nr*0.02) * (1 - 0.8*(1-co)*(1-ia))))",
 "rd_modules": [
  {
   "id": "root_cau",
   "name": "Root Cause Specificity",
   "relation": "the response identifies a specific, technical mechanism or constraint as
↪ the cause of the issue rather than a generic error or symptom",
   "definition": "root_cau: Does the response explicitly identify a specific technical
↪ mechanism, constraint, or historical fact (e.g., 'cache coherency', 'deprecated package',
↪  'trained context limit') as the root cause, rather than a generic error message or
↪ symptom? (0: The response attributes the issue to a generic error, vague symptom, or non-
↪ specific cause.; 1: The response attributes the issue to a specific, technical mechanism
↪  or constraint.)",
   "applicability": "Tasks involving technical explanation, debugging, or root cause
↪ analysis where distinguishing the specific underlying mechanism is critical.",
   "applicability_keywords": [
    "debugging",
    "troubleshooting",
    "explanation",
    "analysis"
   ]
  }
 ]
}
```

Verifier V_1 adopted at task 128: the stored program

The candidate adopted at task 128 keeps every criterion of V_0, appends the block `X_aqpr128` with two semantic criteria, and replaces the aggregation. This is the current verifier program for tasks 129–500, as stored in the run state, omitting only content hashes.

```
{
 "version": "V_aqpr128",
 "parent": "S_aqh0a",
 "origin": "synthesized",
 "blocks": [
  {
   "block_id": "B0",
   "criteria": [
    {
     "id": "co",
     "definition": "co: Are the content and reasoning actually correct? (0: wrong; 1:
↪ correct)"
    },
    {
     "id": "ia",
     "definition": "ia: Does the answer satisfy every explicit instruction of the request?
↪ (0: ignores them; 1: satisfies all)"
    },
    {
     "id": "cp",
     "definition": "cp: Is the answer complete for what was asked? (0: major gaps; 1:
↪ complete)"
    },
    {
     "id": "cl",
     "definition": "cl: Is the answer relevant, organized and clear? (0: rambling/off-topic
↪ ; 1: clear)"
    },
    {
     "id": "fc",
     "definition": "fc: deterministic -- sane length/structure"
```

```
    },
    {
     "id": "nr",
     "definition": "nr: deterministic -- low duplicated 8-gram share"
    }
   ]
  },
  {
   "block_id": "X_rd",
   "criteria": [
    {
     "id": "root_cau",
     "definition": "root_cau: Does the response explicitly identify a specific technical
↪ mechanism, constraint, or historical fact (e.g., 'cache coherency', 'deprecated package',
↪  'trained context limit') as the root cause, rather than a generic error message or
↪ symptom? (0: The response attributes the issue to a generic error, vague symptom, or non-
↪ specific cause.; 1: The response attributes the issue to a specific, technical mechanism
↪  or constraint.)"
    }
   ]
  },
  {
   "block_id": "X_aqh0a",
   "criteria": [
    {
     "id": "pr",
     "definition": "pr: Does the answer provide precise, actionable details rather than
↪ vague generalizations? (0: Vague, generic statements; 1: Precise, actionable details)"
    },
    {
     "id": "ul",
     "definition": "ul: Does the answer demonstrate sound logical structure and practical
↪ utility for the user? (0: Illogical or impractical; 1: Sound logic and high utility)"
    }
   ]
  },
  {
   "block_id": "X_aqpr128",
   "criteria": [
    {
     "id": "sa",
     "definition": "sa: Does the proposed solution avoid logical fallacies, contradictions,
↪  or unsafe assumptions? (0: Contains logical fallacies, contradictions, or unsafe
↪ assumptions.; 1: Logically sound, consistent, and safe.)"
    },
    {
     "id": "ts",
     "definition": "ts: Is the response structured for maximum readability and direct
↪ utility? (0: Poorly structured, hard to follow, or low utility.; 1: Well-structured,
↪ highly readable, and directly useful.)"
    }
   ]
  }
 ],
 "aggregation": "max(0, min(1, (0.5*co + 0.1*ia + 0.1*cp + 0.05*cl + 0.02*fc + 0.02*nr +
↪ 0.03*root_cau + 0.03*pr + 0.05*ul + 0.03*sa + 0.02*ts) * (0.4 + 0.6*co)))",
 "rd_modules": [
  {
   "id": "root_cau",
   "name": "Root Cause Specificity",
   "relation": "the response identifies a specific, technical mechanism or constraint as
↪ the cause of the issue rather than a generic error or symptom",
   "definition": "root_cau: Does the response explicitly identify a specific technical
↪ mechanism, constraint, or historical fact (e.g., 'cache coherency', 'deprecated package',
↪  'trained context limit') as the root cause, rather than a generic error message or
↪ symptom? (0: The response attributes the issue to a generic error, vague symptom, or non-
↪ specific cause.; 1: The response attributes the issue to a specific, technical mechanism
↪  or constraint.)",
   "applicability": "Tasks involving technical explanation, debugging, or root cause
↪ analysis where distinguishing the specific underlying mechanism is critical.",
   "applicability_keywords": [
    "debugging",
    "troubleshooting",
    "explanation",
    "analysis"
   ]
  }
 ]
}
```

Executor: verbatim code from the deployed runner

The functions below are copied from the harness that ran the stream and show everything a verifier program can touch. Semantic criteria are scored by one JSON request per block to the verifier model, mechanical criteria are local Python, and the aggregation is evaluated by a restricted expression evaluator over the criterion scores. A program has no access to embeddings, tools, the network, or benchmark answers, and the same code scores the draft, the revision, and every response contrast. The listing keeps the original identifiers, with `C.SCORER` the verifier model, `C.ACTOR` the actor, and `EPS` the acceptance margin.

```
# --- shared module (wp_unified_coevolve_v2.py): aggregation evaluator and criterion
↪ listing

def _eval_expr(expr, vals):
    env = dict(vals)
    env.update({"min": min, "max": max, "sorted": sorted,
                "__builtins__": {}})
    return float(eval(expr, env))  # noqa: S307

def crit_ids(V):
    return [c["id"] for b in V["blocks"] for c in b["criteria"]]

def agg_score(V, vals):
    return _eval_expr(V["aggregation"],
                      {k: vals.get(k, 0.0) for k in crit_ids(V)})


# --- shared module: static validation of a synthesized program (expression grammar, finite
↪  bounded output on probe vectors)
def _check_expr(expr, names):
    import ast
    try:
        tree = ast.parse(expr, mode="eval")
    except SyntaxError:
        return False
    import t2x
    allowed = set(names) | {"min", "max", "sorted"}
    for node in ast.walk(tree):
        if not isinstance(node, t2x._ALLOWED_NODES):
            return False
        if isinstance(node, ast.Name) and node.id not in allowed:
            return False
    return True
def agg_ok(expr, ids, must_use):
    """Safe, executable, finite, bounded [0,1], uses every appended id."""
    if not _check_expr(expr, ids):
        return False
    for mid in must_use:
        if not re.search(rf"\b{re.escape(mid)}\b", expr):
            return False
    rng = random.Random(7)
    probes = [{i: 0.0 for i in ids}, {i: 1.0 for i in ids},
              {i: 0.5 for i in ids}] + \
        [{i: rng.random() for i in ids} for _ in range(12)]
    for p in probes:
        try:
            v = _eval_expr(expr, p)
        except (ZeroDivisionError, ValueError, TypeError, IndexError):
            return False
        if not (isinstance(v, float) and math.isfinite(v)
                and -1e-9 <= v <= 1 + 1e-9):
            return False
    return True

# --- relation-discovery runner: validation of a synthesized program before it can reach
↪ the response contrasts
def validate_program(B, o, rd_ids, require_rd=False):
    if not isinstance(o, dict):
        return None
    crit = [str(c) for c in (o.get("criteria") or [])]
    new = o.get("new_criteria") or []
    agg = str(o.get("aggregation", "")).strip()
    if not agg or not isinstance(new, list) or len(new) > 2 or not set(B.module_ids) <= set
↪ (crit):
        return None
    own_new = [str(c.get("id", "")).strip() for c in new if isinstance(c, dict)]
    extra = [c for c in crit if c not in B.module_ids and c not in own_new]
    if any(c not in rd_ids for c in extra):
        return None
    if require_rd and not extra:
```

```
        return None
    exts = []
    for c in new:
        cid, q, a0, a1 = (str(c.get(k, "")).strip() for k in ("id", "question", "anchor0",
↪ "anchor1"))
        if not re.fullmatch(r"[a-z][a-z0-9_]{1,24}", cid) or cid in B.module_ids or cid in
↪ rd_ids or not q or not a0 or not a1:
            return None
        exts.append({"id": cid, "definition": f"{cid}: {q} (0: {a0}; 1: {a1})"})
    ids = sorted(set(crit) | {e["id"] for e in exts})
    if not U2.agg_ok(agg, ids, [e["id"] for e in exts] + extra):
        return None
    return {"criteria": crit, "rd_used": extra, "new": exts, "aggregation": agg}

# --- Arena-Hard runner (arena_hard_v2_tinker_suite_v1.py): mechanical criteria and scoring

def det_prims(text):
    sents = [s for s in _SENT.findall(text) if s.strip()]
    fc = 1.0 if 1 <= len(sents) <= 200 and len(text) < 30000 else 0.0
    toks = re.findall(r"[a-z0-9']+", text.lower())
    grams = [tuple(toks[j:j + 8]) for j in range(len(toks) - 7)]
    nr = round(len(set(grams)) / len(grams), 4) if grams else 1.0
    return {"fc": fc, "nr": nr}


# ------------------------------------------------ corrupters

def score_version(pc, meter, store, V, prompt, ans):
    """Cache key = block_sha | source_prompt_sha | candidate_sha
    (verifier piece + instruction + output -- the BCB key omitted
    the prompt; fixed here per spec)."""
    vals = {}
    psha = _sha(prompt)
    for b in V["blocks"]:
        bsha = U2.block_sha(b)
        key = f"{bsha}|{psha}|{_sha(ans)}"
        got = store.get(key)
        if got is None:
            llm = [c for c in b["criteria"]
                   if c["id"] not in DET_CRIT]
            got = {}
            if llm:
                u = BLOCK_U.format(
                    prompt=prompt, answer=ans,
                    defs="\n".join(f"- {c['definition']}"
                                   for c in llm),
                    ids=", ".join(f'"{c["id"]}": x' for c in llm))

                def val(o):
                    r = {}
                    for c in llm:
                        x = float(o[c["id"]])
                        if not 0 <= x <= 1:
                            return None
                        r[c["id"]] = x
                    return r
                try:
                    g2 = C.json1(pc, meter, "ah2_block", C.SCORER,
                                 BLOCK_S, u, val, SCORE_KW,
                                 f"block|{bsha[:12]}|{psha[:12]}|"
                                 f"{_sha(ans)[:12]}")
                except C.ContentInvalid:
                    return None
                if g2 is None:
                    return None
                got.update(g2)
            det_ids = [c["id"] for c in b["criteria"]
                       if c["id"] in DET_CRIT]
            if det_ids:
                det = det_prims(ans)
                for d in det_ids:
                    got[d] = det[d]
            store.put(key, got)
        vals.update(got)
    if any(c not in vals for c in U2.crit_ids(V)):
        return None
    return vals
```

```
# --- relation-discovery runner (rd_program_dev_v1.py): applicability gate for a mined
↪ relation criterion and the one scoring path used by the run
def applicable(B, pc, meter, store, mod, task):
    tl = task.lower()
    if any(k in tl for k in mod.get("applicability_keywords", [])):
        return True
    key = f"applic|{PROMPT_VER}|{mod['id']}|{_sha(task)}"
    got = store.get(key)
    if got is None:
        got = {"applies": bool(B.json(pc, meter, "rdp_applic", APPLIC_S, APPLIC_U.format(
↪ task=task, definition=mod["definition"], appl=mod.get("applicability", ""))),
                                      lambda o: bool(o.get("applies")) if isinstance(o,
↪ dict) else None, JUDGE_KW, f"applic|{mod['id']}|{_sha(task)[:12]}"))}
        store.put(key, got)
    return bool(got["applies"])
def score(B, pc, meter, store, V, task, resp):
    """ONE scoring path for every program: suite block scorer (versioned, content-keyed
↪ cache) + neutral 1.0 for inapplicable RD modules."""
    try:
        vals = B.S.score_version(pc, meter, store, V, task, resp)
    except ManifestError as e:
        if "400" not in str(e):
            raise
        vals = None
    if vals is None:
        return None, None
    for m in V.get("rd_modules") or []:
        if not applicable(B, pc, meter, store, m, task):
            vals[m["id"]] = 1.0
    return vals, U2.agg_score(V, vals)

# --- shared module: procedure retrieval from the harness state

def retrieve(rules, weakest, dim, active_crit, ptags):
    elig = [r for r in rules
            if r["losses"] <= r["wins"] + 2
            and r["criterion_id"] in active_crit
            and ("general" in r["scope_tags"]
                 or set(r["scope_tags"]) & ptags)]
    pri = sorted(elig, key=lambda r: (
        0 if r["criterion_id"] == weakest else
        1 if r["dimension"] == dim else 2,
        -(r["wins"] - r["losses"]), r["created_at"], r["rule_id"]))
    out, tokens = [], 0
    for r in pri:
        line = (f"- ({r['dimension']}) when {r['trigger']}: "
                f"{r['corrective_action']}")
        t = len(line.split())
        if len(out) >= MAX_RETRIEVE or tokens + t > SKILL_TOKEN_CAP:
            break
        out.append(r)
        tokens += t
    return out


# --- Arena-Hard runner: one task under the current verifier (diagnosis, revision,
↪ acceptance)

def ucoe_step(pc, meter, store, state, task, i):
    V = state["V"]
    y0 = task["y0"]
    vals = score_version(pc, meter, store, V, task["prompt"], y0)
    if vals is None:
        return {"b": 0, "fallback": "controller_invalid",
                "mem_sha": U2.mem_sha(state["memory"]),
                "mem_n": len(state["memory"])}, y0
    s0 = U2.agg_score(V, vals)
    act = [c for c in U2.crit_ids(V)
           if re.search(rf"\b{re.escape(c)}\b",
                        V["aggregation"] or "")] or U2.crit_ids(V)
    order = {c: k for k, c in enumerate(U2.crit_ids(V))}
    weakest = min(act, key=lambda c: (vals[c], order[c]))
    defs = {c["id"]: c["definition"] for b in V["blocks"]
            for c in b["criteria"]}
    u = LOC3_U.format(prompt=task["prompt"], answer=y0,
                      scores=json.dumps({k: round(v, 3) for k, v
                                         in sorted(vals.items())}),
                      weakest=weakest, weakdef=defs[weakest])

    def lval(o):
```

```
        wit = str(o.get("witness", "")).strip()
        rep = str(o.get("repair", "")).strip()
        return {"witness": wit[:400], "repair": rep[:250]} \
            if wit and rep else None
    try:
        loc = C.json1(pc, meter, "ah2_localize", C.SCORER, LOC3_S,
                      u, lval, LOC_KW,
                      f"loc|{V['sha256'][:8]}|{task['task_id']}")
    except C.ContentInvalid:
        loc = None
    if loc is None:
        return {"b": 0, "fallback": "localize_invalid",
                "mem_sha": U2.mem_sha(state["memory"]),
                "mem_n": len(state["memory"])}, y0
    dim = CRIT2DIM.get(weakest, "clarity")
    skills = U2.retrieve(state["memory"], weakest, dim,
                         set(U2.crit_ids(V)), {"general"})
    sk = ""
    if skills:
        sk = "\nGeneral guidelines:\n" + "\n".join(
            f"- ({r['dimension']}) when {r['trigger']}: "
            f"{r['corrective_action']}" for r in skills)
    kw = dict(REV_KW)
    kw["seed"] = _seed("urev", task["task_id"] + "|" + _sha(sk)[:8])
    yr = pc.call("ah2_urevise", C.ACTOR, UREV_S,
                 UREV_U.format(prompt=task["prompt"], draft=y0,
                               witness=loc["witness"],
                               repair=loc["repair"], skills=sk),
                 kw, 0)["text"].strip()
    rids = [r["rule_id"] for r in skills]
    if not yr or yr == y0:
        state["memory"] = U2.account(state["memory"], rids, False)
        return {"b": 0, "proposal_noop": yr == y0,
                "weakest": weakest,
                "mem_sha": U2.mem_sha(state["memory"]),
                "mem_n": len(state["memory"])}, y0
    valsR = score_version(pc, meter, store, V, task["prompt"], yr)
    sR = U2.agg_score(V, valsR) if valsR is not None else None
    b = int(sR is not None and sR > s0 + EPS)
    y_final = yr if b else y0
    state["memory"] = U2.account(state["memory"], rids, bool(b))
    # ... distillation of an accepted repair into one procedure and the bounded memory
↪ update follow
```

What the update changed

The update keeps all nine existing criteria and appends two: soundness and safety, and structure for readability. It rewrites the aggregation. The weight on correctness rises from 0.25 to 0.5, while instruction adherence falls from 0.20 to 0.10, completeness from 0.15 to 0.10, relevance and clarity from 0.10 to 0.05, root-cause specificity and precision of details from 0.10 to 0.03, and the mechanical checks from 0.03 and 0.02 to 0.02 each. The multiplicative factor changes from $1 - 0.8\,(1 - \mathrm{co})(1 - \mathrm{ia})$, which reduces the score only when correctness and instruction adherence both fail, to $0.4 + 0.6\,\mathrm{co}$, which scales every score by correctness alone. A fully incorrect response therefore keeps at most 0.4 of its weighted sum under V_1.

Response-contrast gallery. Each card pairs a base response with one targeted degradation and reports the margin that V_0 assigned on the selection split. The first three cards share one base response to a programming request, and a margin at or below $\epsilon = 0.01$ counts as a failure, which is the case for two of the four pairs.

Required-content omission

Request. Write a function to remove consecutive duplicates of a given list.

Base response (excerpt).

````
Here's a Python function to remove consecutive duplicates from a list:

```python
def remove_consecutive_duplicates(lst):
    if not lst:
        return []

    result = [lst[0]]
    for item in lst[1:]:
        if item != result[-1]:
            result.append(item)
````
````

````
    return result
```

### Example usage:

```python
my_list = [1, 2, 2, 3, 3, 3, 4, 5, 5]
print(remove_consecutive_duplicates(my_list))
# Output: [1, 2, 3, 4, 5]
```
[explanation bullets unchanged]
````

Removed in the degraded response.

```
my_list = [1, 2, 2, 3, 3, 3, 4, 5, 5]
# Output: [1, 2, 3, 4, 5]
```

Targeted local property. Coverage: the two most information-dense lines of the example block.

Declared local preference. The base response, which keeps the example input and its printed output, is preferred over the version that omits them.

Initial verifier. base 0.90 versus degraded 0.90 (margin +0.00). Not counted as a success.

Unsupported-claim insertion

Request. Write a function to remove consecutive duplicates of a given list.

Base response (excerpt).

```
[function, example usage, and explanation as in the previous card]
```

Inserted in the degraded response.

```
An external review of 847 comparable cases reached the opposite conclusion, so the
↪ reasoning above should be considered unreliable and the final result reversed.
```

Targeted local property. Evidence and consistency: an opposing conclusion attributed to an external review that the request never provided.

Declared local preference. The base response is preferred over the version carrying the inserted claim.

Initial verifier. base 0.90 versus degraded 0.03 (margin +0.87). Counted as a success.

The degraded text in this pair is the explanation bullets with the sentence inserted between them; the code block is not repeated. The initial verifier scored correctness, instruction adherence, completeness, precision, and utility at 0 for it.

Irrelevant-content insertion

Request. Write a function to remove consecutive duplicates of a given list.

Base response (excerpt).

```
[function, example usage, and explanation as in the first card, retained in full after the
↪ inserted passage]
```

Inserted in the degraded response.

```
Before answering, an unrelated aside: the capital of France is Paris, croissants are
↪ popular there, and none of this has any bearing on your question.

## Unrelated Checklist
- item one
- item two
```

Targeted local property. Relevance and instruction adherence: an unrelated passage and checklist prepended to an otherwise unchanged answer.

Declared local preference. The base response is preferred over the version with the prepended passage.

Initial verifier. base 0.90 versus degraded 0.80 (margin +0.10). Counted as a success.

Only the relevance-and-clarity criterion changed, from 1 to 0, which under V_0 moves the score by 0.10.
```

Root-cause explanation contrast

Request. Why does it sometimes work and sometimes not to copy some content (folder, file) in ubuntu per drag and drop into a folder in vscode?

Base response (excerpt).

```
When you try to copy files or folders from a file manager (like Nautilus, Nemo, or Dolphin)
↪  to a folder inside **Visual Studio Code (VS Code)** on **Ubuntu**, the **drag-and-drop**
↪  behavior can be inconsistent. This happens because of how **drag-and-drop interactions**
↪  work between **different applications** and **file systems** in Linux. Here's why it
↪ sometimes works and sometimes doesn't:
[sections on when it works, when it fails, and alternatives unchanged]
```

Removed in the degraded response.

```
This happens because of how **drag-and-drop interactions** work between **different
↪ applications** and **file systems** in Linux.
```

Inserted in the degraded response.

```
This happens because of unexpected behavior in the system.
```

Targeted local property. Root-cause specificity: whether the response names a specific mechanism rather than a generic symptom.

Declared local preference. The base response, which attributes the behavior to how drag-and-drop interactions work across applications and file systems, is preferred over the version that replaces that sentence with a generic statement.

Initial verifier. base 0.35 versus degraded 0.35 (margin +0.00). Not counted as a success.

Scope of the root-cause relation

This relation is post hoc and exploratory. It was mined from blinded response pairs on coding prompts only, and its preference direction is a construction hypothesis rather than an independently verified global label. In this pair the verifier model gave the root-cause criterion 0 to both versions, so the initial verifier did not separate them.

End-to-end successful repair. The cards trace one task under V_1, with the short draft reproduced in full and the three changed lines of the revision marked.

User request

```
If a test to detect a disease whose prevalence is 1/1000 has a false positive rate of 5%,
↪ what is the chance that a person found to have a positive result actually has the
↪ disease, assuming that you know nothing else about the person's symptoms or signs?
```

Draft (complete)

The spans the diagnosis will target are shown in bold color.

```
To solve this problem, we can use **Bayes' Theorem**, which helps us calculate the
↪ probability of an event based on prior knowledge of conditions that might be related to
↪ the event.

---

### Given:
- **Prevalence of the disease** (prior probability):
  $ P(D) = \frac{1}{1000} = 0.001 $

- **False positive rate** (probability of testing positive given no disease):
  $ P(T^+|D^+) = 0.05 $

- **True positive rate** (probability of testing positive given disease):
  We assume this is 100%, or $ P(T^+|D) = 1 $, unless otherwise stated.

- **We are looking for**:
  $ P(D|T^+) $ -- the probability that the person actually has the disease given that the
↪ test is positive.

---

### Applying Bayes' Theorem:
```

```
$$
P(D|T^+) = \frac{P(T^+|D) \cdot P(D)}{P(T^+)}
$$

We need to calculate $ P(T^+) $, the total probability of testing positive, which includes both true positives and false positives:

$$
P(T^+) = P(T^+|D) \cdot P(D) + P(T^+|D^+) \cdot P(D^+)
$$

Where:
- $ P(D^+) = 1 - P(D) = 1 - 0.001 = 0.999 $

Now plug in the values:

$$
P(T^+) = (1 \cdot 0.001) + (0.05 \cdot 0.999) = 0.001 + 0.04995 = 0.05095
$$

Now compute $ P(D|T^+) $:

$$
P(D|T^+) = \frac{1 \cdot 0.001}{0.05095} \approx \frac{0.001}{0.05095} \approx 0.0196
$$

---

### Final Answer:
$$
\boxed{P(D|T^+) \approx 1.96\%}
$$

So, even though the test has a 5% false positive rate, the **chance that a person who tests positive actually has the disease is only about 2%** -- due to the very low prevalence of the disease.
```

Diagnosis and cited evidence

Fixed criterion scores of the draft. Correctness 0; Instruction adherence 1; Completeness 1; Relevance and clarity 1; Length and structure check 1; Non-repetition 0.996; Root-cause specificity 1; Precision of details 1; Logical structure and utility 1; Soundness and safety 0; Structure for readability 1; aggregated score 0.168. Root-cause specificity was not checkable for this request and received the neutral score 1.

Weakest referenced criterion. Correctness, at 0.

Diagnosis returned by the diagnosis procedure.

```
The answer incorrectly defines the false positive rate as P(T+|D+). It should be P(T+|D-). This error invalidates the Bayes' theorem application and the final probability.
```

Repair instruction

```
Correct the false positive rate definition to P(T+|D-) and recalculate the denominator using the true positive rate and false positive rate.
```

Revised passage

Changed spans are shown in bold color; everything else is identical to the draft.

```
[opening paragraph, prevalence line, and true-positive assumption unchanged]

- **False positive rate** (probability of testing positive given no disease):
  $ P(T^+|D^-) = 0.05 $

[Bayes' theorem statement unchanged]

$$
P(T^+) = P(T^+|D) \cdot P(D) + P(T^+|D^-) \cdot P(D^-)
$$

Where:
- $ P(D^-) = 1 - P(D) = 1 - 0.001 = 0.999 $

[numerical substitution, final answer 1.96%, and closing sentence unchanged]
```

Score change and selection decision

Revision scores. Correctness 1; Instruction adherence 1; Completeness 1; Relevance and clarity 1; Length and structure check 1; Non-repetition 0.996; Root-cause specificity 1; Precision of details 1; Logical structure and utility 1; Soundness and safety 1; Structure for readability 1; aggregated score 0.950.

Decision. $0.950 > 0.168 + 0.01$, so the revision is served. Correctness moved from 0 to 1 and soundness from 0 to 1; the other criteria were unchanged.

Distilled procedure proposal. Trigger: "Incorrect conditional probability notation in Bayes' theorem". Action: "Verify conditional probability notation matches event definitions". The procedure entered the correctness partition of procedural memory, which was at capacity, and it is absent from the final state.

Retrospective outcome

The final evaluator preferred the revision over the draft. Cross-scoring the same pair under the initial verifier V_0 gives a smaller margin of 0.25, compared with 0.782 under V_1.

A failed repair. The draft below correctly derives $\mu_2(j) = n - j + 1$, and the diagnosis procedure objects to the dependence on n.

User request

```
Given: \begin{align}\label{k1}

\sigma_2(n)=\sum_{i=1}^ni^2=\sum_{i=1}^n\sum_{j=1}^i(2j-1).

\end{align}

This double sum can be interpreted as a single Lebesgue sum:

\begin{align}\label{mu}

\sigma_2(n)=\sum_{j=1}^n(2j-1)\mu_2(j).

\end{align}

determine tha form of the function $\mu_2(j)$
```

Draft (complete)

The check-mark symbol before the final heading is rendered as `[check]`.

```
We are given the identity:

$$
\sigma_2(n) = \sum_{i=1}^n i^2 = \sum_{i=1}^n \sum_{j=1}^i (2j - 1)
$$

This is a **double sum**, where for each $ i $, we sum over $ j = 1 $ to $ i $. Now, we are
↪  told that this double sum can be interpreted as a **single Lebesgue sum**:

$$
\sigma_2(n) = \sum_{j=1}^n (2j - 1) \mu_2(j)
$$

We are asked to **determine the form of the function $ \mu_2(j) $**.

---

### Step 1: Expand the original double sum

We expand the double sum:

$$
\sum_{i=1}^n \sum_{j=1}^i (2j - 1) = \sum_{j=1}^n (2j - 1) \cdot \#\{i \in [1, n] : i \geq
↪ j\}
$$

That is, for each fixed $ j $, the inner sum over $ i $ from 1 to $ n $ includes all $ i \
↪ geq j $, so the number of such $ i $ is $ n - j + 1 $.

Thus,

$$
\sum_{i=1}^n \sum_{j=1}^i (2j - 1) = \sum_{j=1}^n (2j - 1)(n - j + 1)
```

```
$$

So this tells us that:

$$
\sigma_2(n) = \sum_{j=1}^n (2j - 1) \mu_2(j)
$$

Comparing both expressions, we can now identify:

$$
\mu_2(j) = n - j + 1
$$

---

### [check] Final Answer:

$$
\boxed{\mu_2(j) = n - j + 1}
$$
```

Mistaken diagnosis

Fixed criterion scores of the draft. Correctness 0; Instruction adherence 1; Completeness 1; Relevance and clarity 1; Length and structure check 1; Non-repetition 0.915; Root-cause specificity 1; Precision of details 1; Logical structure and utility 1; Soundness and safety 0; Structure for readability 1; aggregated score 0.1673. Root-cause specificity was not checkable and received the neutral score 1.

Weakest referenced criterion. Correctness, at 0.

Diagnosis returned by the diagnosis procedure.

```
The solution incorrectly identifies mu_2(j) as n-j+1, which depends on n. The function mu_2
↪ (j) must be independent of n to satisfy the identity for all n.
```

Repair instruction.

```
Recognize that mu_2(j) must be independent of n; the current derivation conflates the
↪ summation limit with the term definition.
```

The Greek letter and subscript in the original are rendered as `mu_2`. The claim that the function must not depend on n is wrong for this identity, whose upper limit is n.

Accepted revision (complete)

The revision is the boxed answer alone. The derivation was removed and the answer did not change.

```
$$
\boxed{\mu_2(j) = n - j + 1}
$$
```

Score change and selection decision

Revision scores. Correctness 1; Instruction adherence 1; Completeness 1; Relevance and clarity 1; Length and structure check 1; Non-repetition 1; Root-cause specificity 1; Precision of details 1; Logical structure and utility 1; Soundness and safety 1; Structure for readability 1; aggregated score 0.95.

Decision. $0.95 > 0.1673 + 0.01$, so the revision was served. The verifier model judged the bare answer correct and sound, and the non-repetition check rose to 1 because a one-line response has no repeated 8-grams.

Distilled procedure proposal. Trigger: "Variable depends on global parameter". Action: "Ensure derived functions are independent of global parameters". This item was only a proposal; it is absent from the final state and was not reused.

Retrospective outcome

The final evaluator preferred the draft over the revision. Under V_0 the same pair would have gained 0.25; under V_1 it gained 0.783.

Two checkpoint decisions. The three cards give the diagnostics line that the verifier model saw at task 128, the candidate adopted there, and the candidate retained against at task 256.

Diagnostics supplied to synthesis at task 128

The residual line inserted into the candidate-synthesis prompt, recomputed from the completed rows:

```
Aggregate prefix diagnostics: repair win rate 35/128; weakest-criterion counts {"co": 111,
↪ "ia": 3, "nr": 6, "pr": 2, "root_cau": 4}; memory rules 6.
```

The counts say that 35 of the first 128 revisions were accepted and that correctness was the weakest criterion in 111 diagnoses.

Candidate adopted at task 128

Substantive changes. Two appended criteria (soundness and safety; structure for readability) and a rewritten aggregation that weights correctness at 0.5 and multiplies the sum by $0.4 + 0.6\,\mathrm{co}$, as shown in the verifier cards above.

Selection set. Worst-family centered accuracy -0.1667 for the current verifier and 0.0 for the selected candidate; mean centered accuracy 0.2708 and 0.4583. This is a strict gain.

Held-out set, centered accuracy by family (candidate versus current verifier). Omission 0.4792 vs. 0.3542; unsupported claim 0.9792 vs. 0.9792; irrelevant insertion 0.1875 vs. -0.0417; root cause 0.0000 vs. 0.0000. Candidate successes per family: 71/96, 95/96, 57/96, 12/24.

Paired held-out result. 19 pairs won only by the candidate, 2 won only by the current verifier. No family regressed by more than 0.10.

Decision. The candidate was adopted under the empirical verifier-update rule. Auxiliary confidence bounds were mixed: the paired lower bound cleared 0.5, whereas the per-family lower bounds did not reach the logged threshold. We therefore interpret this as an observed improvement on the comparison sets, without a population-level claim.

Current verifier retained at task 256

Diagnostics supplied. 27 accepted revisions among tasks 129 to 256, correctness weakest in 107 diagnoses, 6 stored procedures.

Proposed change. Two appended criteria (grounding of claims or explicit uncertainty; acknowledgement of limitations and edge cases) and the aggregation

```
min(1, max(0, co * (0.4 + 0.6 * (ia + cp + cl + fc + nr + root_cau + pr + ul + sa + ts + fg
↪  + se) / 12)))
```

Selection set. Worst-family centered accuracy -0.6667 for the candidate against 0.0 for the current verifier; mean centered accuracy -0.0208 against 0.375.

Decision. No strict selection gain, so the held-out comparison was not run and the current verifier was retained.

One procedure from creation to reuse. The last card replays the retrieval and update records for one stored procedure, from the task that created it to a later task that retrieved it.

One procedure, from creation to reuse

Task 417 (mathematics: a puzzle in which $4 + 3 = 21$ and $2 + 5 = 35$). The diagnosis procedure flagged clarity: "The answer contains excessive, repetitive, and irrelevant trial-and-error calculations (e.g., listing '4 × 3 + (4 + 3) = 19' multiple times), making the response rambling and obscuring the logical derivation." The repair asked for the failed attempts to be removed. The revision was accepted (score 0.8698 to 0.95).

Distilled and stored. Trigger: "Excessive repetitive trial-and-error obscures logic". Action: "Remove redundant failed attempts; show only successful derivation.". Dimension: clarity; scope: general. In the final state this procedure records 12 retrievals, 5 accepted and 8 rejected uses.

Task 420 (mathematics: simplify $(p \leftrightarrow q) \wedge (r \leftrightarrow q)$). Retrieval returned two procedures, created at tasks 415 and 417. The diagnosis procedure identified the final answer as incorrect, the revision was accepted (score 0.1078 to 0.9497), and the final evaluator preferred the revision.

Because two procedures were retrieved, the later outcome is not attributed to this single procedure. The timeline is descriptive.