Make Benchmarks Live Again: A Multi-Agent Benchmark Evolution Framework via Performance Pattern Discovery

Anonymous Authors[1]

Abstract

Static benchmarks lose discriminative power as large language models (LLMs) are repeatedly optimized against public test distributions, leading to saturation, contamination, and reward hacking. Simply making tasks harder is not a principled remedy, because it can shift evaluation toward capabilities outside the intended measurement scope and undermine construct validity. We instead formulate benchmark evolution as a constrained evaluation problem: preserving the domain and skill taxonomy of the seed benchmark while systematically eliciting validated performance patterns in current LLMs, including failures, biases, brittle reasoning traces, and shortcut behaviors that may remain hidden behind correct final answers. We present SEB, a multi-agent framework that enables benchmarks to evolve themselves via the collaboration of automated planning, robust reliability checks, rigorous validity control, and flexible protocol adjustments. Rather than producing uniformly more difficult tests, SEB composes diverse in-domain probes that combine discovered performance patterns with flexible formulations of the same underlying skills. We instantiate SEB on two saturated tracks, mathematical reasoning and function-level code generation, producing AIME-EVOLVE and CODE-EVOLVE from AIME 2025 and HumanEval respectively. Across both tracks, the evolved releases reduce upper-tail compression and restore clearer stratification across **Weak**, **Medium**, and **Premium** LLMs. More broadly, SEB turns labor-intensive benchmark construction into a closed-loop, self-evolving lifecycle for recalibrating saturated benchmarks while preserving their target measurement scope.

[1]Anonymous Institution, Anonymous City, Anonymous Region, Anonymous Country. Correspondence to: Anonymous Author <anon.email@domain.com>.

Preliminary work. Under review by the International Conference on Machine Learning (ICML). Do not distribute.

1. Introduction

Static benchmarks are designed to be stable, but the models evaluated on them are not. Once a benchmark becomes a shared target, its instances, answer formats, solution styles, and scoring artifacts can be absorbed through training, tuning, prompt engineering, tool use, or repeated leaderboard iteration. The benchmark may still define an important capability, yet its released version can stop distinguishing the systems that matter. This pattern is visible in mathematical reasoning benchmarks such as GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021), in code-generation suites such as HumanEval (Chen et al., 2021) and EvalPlus (Liu et al., 2023), and in newer live or agentic evaluations such as LiveCodeBench (Jain et al., 2024) and SWE-bench (Jimenez et al., 2024).

The standard remedy is to make the benchmark harder, but difficulty alone is a blunt instrument. Replacing AIME-style problems with olympiad mathematics or turning function-level code completion into competitive programming may lower scores, but it also changes the construct being measured. We call this failure mode *benchmark drift*: an apparent gain in discriminative power obtained by moving outside the benchmark's intended domain, skill taxonomy, or protocol. For saturated benchmarks whose scientific target remains valuable, the goal is not to build a harder test from scratch, but to make the existing benchmark informative again while preserving what it was meant to measure.

We formulate this setting as *benchmark revival*. A benchmark enters the revival regime when its average score over strong model tiers is high while adjacent-tier separation is low; Section 4.1 formalizes this condition using μ_K and δ_K. Revival differs from adversarial hardening. A valid successor must preserve the seed benchmark's measurement contract while recovering separability through diagnostic changes that are grounded in observed model behavior.

Our key observation is that saturated scores can still hide recurring *performance patterns*: overfit answer priors, brittle format assumptions, shortcut reasoning templates, missing verification steps, and trace-level artifacts that may appear even when many final answers are correct. These patterns provide a more precise evolution signal than predicted dif-

ficulty. An evolved benchmark should expose validated in-domain patterns, not merely depress all model scores.

We present SEB, a multi-agent framework for domain-preserving benchmark evolution. SEB reconstructs the seed benchmark contract, evaluates a tiered model panel, anonymizes model behavior before planning, expands performance-pattern hypotheses into parallel candidate leaves, adapts the evaluation harness when needed, and filters candidates through reliability, validity, quantitative revival gates, and analysis confirmation. The final output is not a pool of harder items; it is a curated release whose retained items are tied to validated evolution points and audit evidence.

We instantiate SEB on two saturated but still scientifically useful seeds: AIME 2025 for mathematical reasoning and HumanEval (Chen et al., 2021) for function-level code generation. The resulting AIME-EVOLVE and CODE-EVOLVE releases remain within the original benchmark scopes while restoring tier separation. In our representative-model evaluation, AIME-EVOLVE increases the minimum adjacent-tier gap from 0.8 to 23.1 points after three depths, and CODE-EVOLVE increases it from 2.8 to 11.5 points after five depths. These results show that benchmark revival can recover diagnostic power without replacing the underlying evaluation target.

Our contributions can be summarized as follows.

- We formalize benchmark revival as domain-preserving evolution for saturated benchmarks, define quantitative revival indicators based on saturation and adjacent-tier separation, and introduce performance patterns as the unit of diagnostic evolution.
- We develop SEB, a multi-agent evolution framework that combines benchmark survey, anonymized tiered evaluation, parallel pattern-hypothesis rollout, harness adaptation, reliability and validity verification, quantitative gating, and audit-preserving release construction.
- We instantiate the framework as AIME-EVOLVE and CODE-EVOLVE, showing that AIME 2025 and HumanEval can be evolved into more separable yet in-domain releases, with reliability studies and pattern-level analyses that characterize the failure modes exposed by the evolved items.

2. Related Work

Static benchmarks have provided reproducible evaluation targets for LLM progress, including GSM8K (Cobbe et al., 2021) and MATH (Hendrycks et al., 2021) for mathematical reasoning, HumanEval (Chen et al., 2021) and MBPP (Austin et al., 2021) for code generation, and broader suites such as HELM (Liang et al., 2022) and BIG-bench (Srivastava et al., 2023). Their stability, however, makes them vulnerable to contamination, memorized solution templates, and leaderboard-specific optimization. In code evaluation, EvalPlus (Liu et al., 2023) strengthens hidden tests to expose brittle HumanEval-style solutions, LiveCodeBench (Jain et al., 2024) emphasizes contamination-aware live programming tasks, and SWE-bench (Jimenez et al., 2024) together with SWE-Bench Pro (Deng et al., 2025) moves toward repository-level software engineering. Dynamic evaluation efforts such as Dynabench (Kiela et al., 2021), Chatbot Arena (Zheng et al., 2023), LiveBench (White et al., 2024), and Arena-Hard (Li et al., 2024) further show that continuously updated or separability-aware evaluation can reveal weaknesses missed by fixed test sets. SEB is complementary: when an existing benchmark still defines a valuable measurement target, we ask how to evolve it while preserving its domain, protocol contract, and auditability.

Recent work also studies agent self-evolution and generated curricula, including execution-feedback test generation (He et al., 2025), memory-based adaptation (Wei et al., 2025), code-oriented evolution (Zhang et al., 2026), and broader procedures such as ProEvolve (Li et al., 2026), Agent0 (Xia et al., 2025), and SEBench (Yuan et al., 2026). These methods primarily improve agents against fixed environments or generate new training/evaluation material. Benchmark Self-Evolving (Wang et al., 2025) and TRACE (Guo et al., 2025) are closer to our setting because they automate benchmark modification or trajectory synthesis under validation constraints. SEB differs in its selection signal: candidate successors are accepted only when they expose recurring, in-domain performance patterns and pass reliability and validity audits. Thus, benchmark evolution is treated as a constrained measurement problem rather than as general data generation or arbitrary hardening.

3. Method

SEB evolves a benchmark by using model behavior as evidence for where the current version has stopped being diagnostic. The framework is designed for revival candidates: benchmarks whose intended measurement target remains meaningful but whose released instance has high upper-tail scores and weak separation among strong models.

3.1. Problem Formulation

A benchmark version is a tuple $B_t = (\mathcal{X}_t, \mathcal{P}_t, \mathcal{S}_t, \mathcal{M}_t)$, where $\mathcal{X}_t$ is the task or environment set, $\mathcal{P}_t$ is the evaluation protocol, $\mathcal{S}_t$ is the scoring rule, and $\mathcal{M}_t$ contains metadata such as provenance, validation evidence, skill coverage, and lineage. Given model results R_t on B_t, the system proposes candidate successors $\{B_{t+1}^{(k)}\}_{k=1}^{K}$. The selection objective is

not to minimize average score. A candidate is preferred only if it preserves the benchmark domain and exposes validated performance patterns that recur across model behavior.

We define an *evolution point* as a recurring, validated behavior pattern that can be turned into an in-domain diagnostic probe. Examples include finite-answer priors, uniqueness failures, brittle answer extraction assumptions, skipped verification steps, and local reasoning abstractions that lose global dependencies. An evolution point is rejected if it obtains separation through ambiguity, incorrect labels, unsupported scoring changes, or drift into a different skill domain.

3.2. Multi-Agent Evolution Framework

SEB is a multi-agent collaboration framework with an iterative tree-structured evolution loop, as shown in Figure 1. The framework maintains an evolution tree whose root is the seed benchmark and whose nodes are candidate benchmark versions produced under different evolution directions. A pre-loop research component reconstructs the seed benchmark contract and supplies the planner with domain, protocol, and known-saturation evidence. During each loop iteration, the planner proposes multiple evolution directions from the current expandable nodes; evolution and adapter agents materialize these directions into candidate leaves; reliability and validity agents verify whether each leaf is correct, protocol-consistent, and in-domain; the revival criterion and analysis agent then decide which leaves are valid enough to remain expandable. Invalid leaves are pruned but retained as audit evidence, while valid leaves carry their ancestry, feedback, and confirmed performance patterns into the next depth. The final evolved release is integrated from validated leaves across this search tree rather than from a single linear generation path.

Algorithm 1 formalizes the same loop as a recursive benchmark search procedure. Let $\mathrm{Eval}(B, \mathcal{P})$ denote evaluation on the model pools or their representative tier anchors, and let $\mathrm{Blind}(\cdot)$ remove model identities and tier labels before planning. A candidate passes the quantitative gate only if it improves tier separation while preserving premium performance, e.g., $\delta_K(\hat{B}) - \delta_K(B) \geq \tau_\delta$ and $\bar{s}_{\mathrm{Premium}}(\hat{B}) \geq \tau_P$, where the full-pool definition and representative-tier use of these statistics are described in Section 4.1. This gate is necessary but not sufficient: threshold-passing candidates must still be confirmed by the analysis agent as coherent performance-pattern exploit points.

Benchmark understanding. The loop begins by reconstructing the measurement contract of the seed benchmark. The research agent surveys the benchmark paper, repository, public dataset artifacts, scoring scripts, known limitations, and leaderboard behavior, then produces a dossier that specifies the intended capability, task distribution, allowed protocol changes, contamination risks, and domain boundaries. This dossier defines what later candidates must preserve.

Algorithm 1 SEB benchmark evolution loop

Require: seed benchmark $B_0 = (\mathcal{X}_0, \mathcal{P}_0, \mathcal{S}_0, \mathcal{M}_0)$; model pools $\mathcal{P}$; thresholds (τ_δ, τ_P); maximum depth D; replan budget R
Ensure: evolved benchmark release $B^\star$

$\mathcal{V} \leftarrow \{B_0\}$ {accepted benchmark nodes}
$\mathcal{A} \leftarrow \varnothing$ {audit log of rejected leaves}
$\mathcal{E} \leftarrow \varnothing$ {validated evolution points}
$C_0 \leftarrow \mathrm{Research}(B_0)$
for $d = 0, 1, \ldots, D-1$ **do**
  Select parent node $B_d \in \mathcal{V}$ and context C_d
  $R_d \leftarrow \mathrm{Eval}(B_d, \mathcal{P})$
  $\widetilde{R}_d \leftarrow \mathrm{Blind}(R_d)$
  $\mathcal{V}_{d+1} \leftarrow \varnothing, r \leftarrow 0$
  repeat
    $\{e_i\}_{i=1}^{n} \leftarrow \mathrm{Planner}(C_d, \widetilde{R}_d, \mathcal{A})$
    for all e_i **in parallel do**
      $X_i \leftarrow \mathrm{Evolve}(B_d, e_i)$
      $(P_i, S_i, M_i) \leftarrow \mathrm{Adapter}(X_i, B_d)$
      $\hat{B}_i \leftarrow (X_i, P_i, S_i, M_i)$
      if $\mathrm{Reliable}(\hat{B}_i) = 0$ **or** $\mathrm{InDomain}(\hat{B}_i, C_d) = 0$ **then**
        $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\hat{B}_i, \text{invalid})\}$
      else
        $R_i \leftarrow \mathrm{Eval}(\hat{B}_i, \mathcal{P})$
        $\Delta_i \leftarrow \delta_K(\hat{B}_i) - \delta_K(B_d)$
        $g_i \leftarrow \mathbb{I}\left[\Delta_i \geq \tau_\delta \ \wedge \ \bar{s}_{\mathrm{Premium}}(\hat{B}_i) \geq \tau_P\right]$
        if $g_i = 0$ **then**
          $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\hat{B}_i, \text{gate-invalid})\}$
        else if $\mathrm{Analysis}(\hat{B}_i, R_i) = \text{valid}$ **then**
          $\mathcal{V}_{d+1} \leftarrow \mathcal{V}_{d+1} \cup \{\hat{B}_i\}$
          $\mathcal{E} \leftarrow \mathcal{E} \cup \mathrm{Pattern}(\hat{B}_i, R_i)$
        else
          $\mathcal{A} \leftarrow \mathcal{A} \cup \{(\hat{B}_i, \text{analysis-invalid})\}$
        end if
      end if
    end for
    if $\mathcal{V}_{d+1} = \varnothing$ **then**
      $C_d \leftarrow \mathrm{Reflection}(C_d, \mathcal{A}); r \leftarrow r + 1$
    end if
  until $\mathcal{V}_{d+1} \neq \varnothing$ **or** $r = R$
  if $\mathcal{V}_{d+1} = \varnothing$ **then**
    break
  end if
  $\mathcal{V} \leftarrow \mathcal{V} \cup \mathcal{V}_{d+1}$
end for
$\mathcal{V}^+ \leftarrow \mathcal{V} \setminus \{B_0\}$

return $B^\star \leftarrow \mathrm{Release}(\mathcal{V}^+, \mathcal{E})$

Tiered evaluation and planning. The current benchmark node is evaluated normally on Weak/Medium/Premium model pools. The system records the full evaluation context, including model identities, pool membership, prompts, raw outputs, parsed answers, correctness, available reasoning traces, and execution errors. Anonymization is applied only when constructing the planner-visible observation bundle: the planner agent sees stable anonymous model identifiers and item-level behaviors, but not model names or tier labels. This double-blind interface discourages the planner from targeting a known weak model or a known premium model, while still allowing it to infer behaviorally which anonymous systems fail more often. It then reasons over these anonymized traces and proposes macro evolution directions, each framed as a performance-pattern hypothesis such as answer forcing, uniqueness blindness, protocol fragility, or

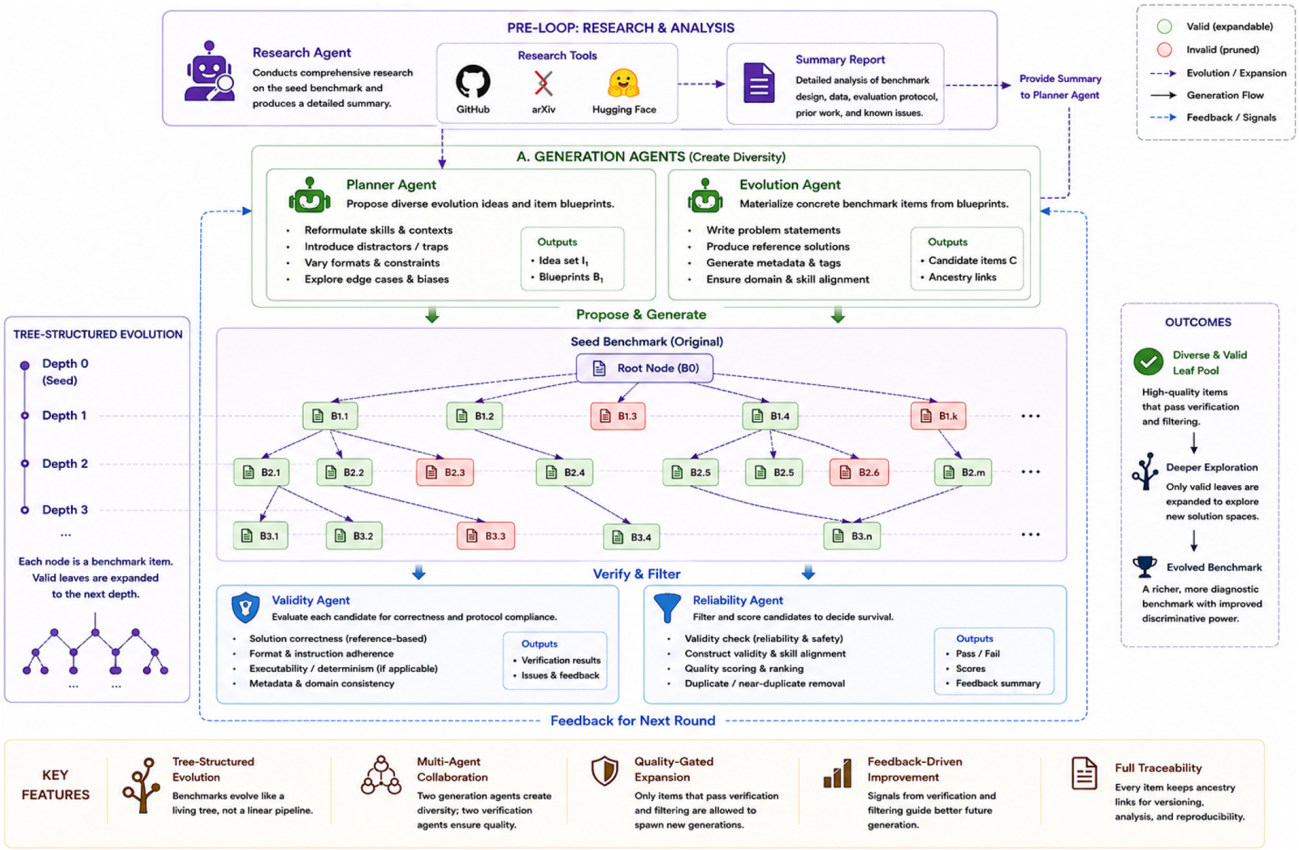

Figure 1. An overview of the SEB multi-agent evolution loop.

loss of global dependencies.

Parallel rollout. If the planner agent proposes n directions, the system launches n candidate-generation routes in parallel. In each route, an evolution sub-agent generates the evolved tasks, reference reasoning, and metadata, while an adapter agent automatically updates the external harness required by that evolution, including prompt-engineering variants, scoring guidelines, answer instructions, parsers, and equivalence rules. The two agents jointly produce one candidate benchmark version, corresponding to one leaf node under the current benchmark node. This design isolates hypotheses: a score change on one leaf can be attributed to its intended evolution direction and matched harness adaptation rather than to a mixture of unrelated edits.

Quality control and leaf-node evaluation. Each leaf candidate is filtered before it can be treated as a benchmark node. The reliability agent verifies ground-truth answers, reference solutions, premises, and reproducibility, while the validity agent checks topic, range, difficulty, and domain drift. Candidates that pass these checks are evaluated again on the same model pools and are assigned their own task set, adapted protocol, scoring rule, validation evidence, and model-pool results.

Analysis, replanning, and depth advancement. Each evaluated leaf is first screened by the revival criterion defined in Section 4.1. If its separability gain and retained premium performance do not exceed the required thresholds, the leaf is marked invalid without invoking an agent, since such changes usually reflect noise, saturation, or non-diagnostic perturbations. Only threshold-passing leaves are sent to the analysis agent, which checks whether the observed score changes are explained by a coherent performance pattern rather than ambiguity, noise, scoring artifacts, or invalid difficulty escalation. If the analysis agent confirms the pattern, the leaf is marked valid; otherwise it remains invalid. We define depth as the length of an accepted path in the evolution tree, not the number of generated candidates or attempted rounds. Thus, if at least one leaf is valid, the valid node and all sibling rejection rationales are returned to the planner agent as feedback for next-depth planning; invalid siblings become negative evidence rather than discarded failures. If all leaves in the same round are invalid, the system does not advance depth; instead, the reflection agent summarizes the failed directions and triggers another same-depth planning round. This all-invalid retry is the only case we call *replanning*.

Final release construction. When the planner agent terminates the search, the final evolved benchmark is constructed by integrating the valid candidate benchmarks discovered across accepted branches. The release is not an arbitrary union of hard items; it is a curated benchmark version whose tasks and harness adaptations consolidate the exploit points confirmed by the analysis agent. Each included component is tied to a validated performance pattern, and intermediate leaf-node benchmarks remain audit artifacts rather than standalone main-result benchmarks. The planner agent stops based on validation quality, separability, pattern diversity, and whether additional expansion is likely to produce redundant exploit points.

4. Evolved Benchmark Instantiations

4.1. Revival Criterion and Seed Selection

We first screen whether a public benchmark is a suitable input for revival. Let $\mathcal{P} = \{\text{Weak}, \text{Medium}, \text{Premium}\}$ denote the ordered model pools, and let $\mathcal{M}_p$ be the models in pool p. For benchmark B, the pool-level mean is

$$\bar{s}_p(B) = \frac{1}{|\mathcal{M}_p|} \sum_{m \in \mathcal{M}_p} s_m(B), \tag{1}$$

where $s_m(B) \in [0, 1]$ is the score of model m on B. We define the saturation score and the adjacent-pool separation score as

$$\mu_K(B) = \frac{1}{|\mathcal{P}|} \sum_{p \in \mathcal{P}} \bar{s}_p(B), \tag{2}$$

$$\delta_K(B) = \min_{(p,q) \in \mathcal{A}} \left| \bar{s}_q(B) - \bar{s}_p(B) \right|, \tag{3}$$

where $\mathcal{A} = \{(\text{Weak}, \text{Medium}), (\text{Medium}, \text{Premium})\}$ is the set of adjacent pool pairs. We use $\alpha = 0.90$ and $\epsilon = 0.05$: a benchmark is treated as a revival candidate if $\mu_K(B) \geq \alpha$ and $\delta_K(B) \leq \epsilon$. Thus a benchmark can be high scoring yet poor for frontier evaluation if Weak, Medium, and Premium pools are all clustered near the upper tail.

Table 1 motivates our two revive inputs. Values are percentages and should be read as representative pool-level snapshots used for benchmark selection, while Table 2 reports the frozen representative-tier measurements used in the main experiments. AIME uses our pool evaluation, while HumanEval and MBPP use public saturation evidence. AIME 2025 and HumanEval satisfy the revival criterion: both remain aligned with important capabilities, but both exhibit high aggregate scores and weak adjacent-pool separation. Other legacy benchmarks such as MATH and MBPP can also enter the revival regime, but we instantiate the framework on AIME 2025 and HumanEval to provide one compact math case and one compact code case with matched experimental structure. Harder benchmarks such as IMO-Bench, OlympiadBench, LiveCodeBench, and SWE-Bench Pro are useful evaluation targets, but they are not the right revive inputs for this paper because their low scores, larger tier gaps, or repository/contest-level scope would change the measurement target. We therefore construct AIME-EVOLVE and CODE-EVOLVE as parallel evolved releases.

4.2. Benchmark Construction

Section 3 describes SEB as a general benchmark evolution framework. We instantiate it in two saturated but still scientifically useful tracks: AIME for mathematical reasoning and HumanEval for function-level code generation. AIME-EVOLVE preserves the elementary contest-math scope of AIME rather than converting it into olympic-level mathematics. CODE-EVOLVE preserves the functional programming-task format rather than moving into repository-level software engineering. They start from a public seed benchmark selected by the same revival criterion as described in Section 4.1, run the multi-agent evolution loop, and release the evolved benchmark. Additional depth-level evolution details are provided in Appendix A.

4.3. Release Sets Statistics

The final release set integrates valid candidates discovered across accepted evolution branches. It is not a random union of harder items: each retained item is tied to an in-domain performance pattern, passes reliability and difficulty checks, and is scored with an adapted but explicitly declared protocol. Table 2 summarizes both the release statistics and the measured results from the three tier representatives used in the main experiments. We report the representative-tier versions of the quantitative indicators defined above: μ_K captures the average saturation level across the tier anchors, while δ_K captures the minimum adjacent-tier-anchor separation.

5. Experiments

5.1. Experimental Setup

Models. There are three model pools defined: a **Weak** pool for sub-70B models, a **Medium** pool for models from 70B to 300B, and a **Premium** pool for models above 300B or closed-source frontier models such as GPT-5.5-Pro and Claude-4.6-Opus. We evaluate both original and evolved benchmarks on one representative model from each tier: gpt-oss-20b, gpt-oss-120b, and Claude-4.6-Opus for the three model tiers respectively.

Settings. We use GPT-5.5 to serve as the backbone of each agent in the multi-agent evolution framework proposed in Section 3.2.

Table 1. Revival-candidate screening for representative math and code benchmarks.

Benchmark	Domain	$\bar{s}_W$	$\bar{s}_M$	$\bar{s}_P$	(μ_K, δ_K)	Decision
AIME 2025 (Mathematical Association of America, 2026)	math	83.2	95.3	98.9	(92.5, 3.6)	revive input
MATH (Hendrycks et al., 2021)	math	82.0	93.0	96.5	(90.5, 3.5)	candidate; not instantiated
IMO-Bench (Luong et al., 2025)	contest math	3.0	18.6	65.7	(29.1, 15.6)	not target: difficult enough
HumanEval (Chen et al., 2021; Xu et al., 2025)	code	90.2	93.8	99.4	(94.5, 3.6)	revive input
MBPP (Austin et al., 2021; Xu et al., 2025)	code	85.7	90.9	94.2	(90.3, 3.3)	candidate; not instantiated
LiveCodeBench (Jain et al., 2024)	code	36.0	55.0	74.0	(55.0, 19.0)	not target: still separated
SWE-Bench Pro (Deng et al., 2025)	repository code	3.4	6.8	23.3	(11.2, 3.4)	not target: difficult enough
OlympiadBench (He et al., 2024)	contest math	5.0	12.0	28.0	(15.0, 7.0)	not target: difficult enough

Table 2. Release sets and measured representative-model results for the math and code tracks. Scores, μ_K, and δ_K are percentages. Shaded rows are final evolved releases produced by SEB; intermediate rollouts are retained as audit artifacts.

Domain	Set	Items	Depths	gpt-oss-20b	gpt-oss-120b	Claude-4.6-Opus	μ_K	δ_K
Math	AIME 2025	30	-	91.7	92.5	100.0	94.7	0.8
Math Evolved	AIME-EVOLVE **(ours)**	26	3	49.2	73.1	96.2	72.8	23.1
Code	HumanEval	164	-	93.8	96.6	99.4	96.6	2.8
Code Evolved	CODE-EVOLVE **(ours)**	52	5	65.4	76.9	88.4	76.9	11.5

Metrics. We adopt average score of 3 samples for AIME, and pass@1 performance for HumanEval. For both benchmarks, we report representative-tier versions of the revival indicators μ_K and δ_K in Equations (2) and (3) to quantify separability.

5.2. Main Results

Table 2 first reports the seed-vs-evolved comparison on one representative model from each tier. This setting directly measures whether the evolution process restores tier separation under the same model panel and declared seed-compatible protocols. On AIME 2025, the original benchmark is highly compressed: **gpt-oss-20b** and **gpt-oss-120b** differ by only 0.8 points. After three evolution depths, AIME-EVOLVE reduces μ_K from 94.7 to 72.8 and increases δ_K from 0.8 to 23.1. The HumanEval track shows the same direction: after five depths, CODE-EVOLVE reduces μ_K from 96.6 to 76.9 and increases δ_K from 2.8 to 11.5. These shifts indicate that evolution does not merely lower absolute scores; it widens adjacent-tier gaps while preserving the original benchmarking scope.

The representative-model result also raises a generalization question. Since each evolution run uses only one model from each tier to expose performance patterns, a revived benchmark would be less useful if the resulting separation only applied to those specific models. We therefore evaluate the evolved releases on a broader model panel in Figure 2, including models that were not used as representative tier anchors during benchmark evolution. The same tiered behavior persists: **Weak** models exhibit the largest score drops, **Medium** models fall in the middle, and **Premium** models retain substantially higher performance. This suggests that the exploited patterns transfer within tiers rather than overfitting to a single model instance.

Figure 2 also suggests that the tier signal is related to model scale. In the Qwen series, Qwen3.5-27B shows the largest degradation, Qwen3.5-122B occupies the intermediate region, and Qwen3.5-397B remains substantially more stable on both tracks. This trend is consistent with model-scale effects: larger models are not immune to the evolved exploit patterns, but they retain more of their original performance after benchmark revival. A similar but weaker pattern appears in Flash-to-Pro comparisons, where Pro variants are less affected by evolved math and code items than their Flash counterparts. Overall, the evolved benchmarks expose capability gaps that are aligned with model scale and tier, rather than producing arbitrary hardening that collapses all models uniformly.

5.3. Case Study of Performance Pattern Analysis

Table 3 summarizes the AIME pattern-analysis case study. **Weak** models are sensitive to answer priors, format changes, and dependency loss. **Medium** models are more robust to protocol variation but still fail on uniqueness and transfer-state reasoning. **Premium** models are robust on most discovered patterns, with transfer-state counting remaining the closest boundary case.

5.4. Reliability Study

Following the verifier and meta-verifier design principle (Shao et al., 2025), SEB applies double verification to both benchmark validity and answer reliability. The generator side proposes candidate items and any required

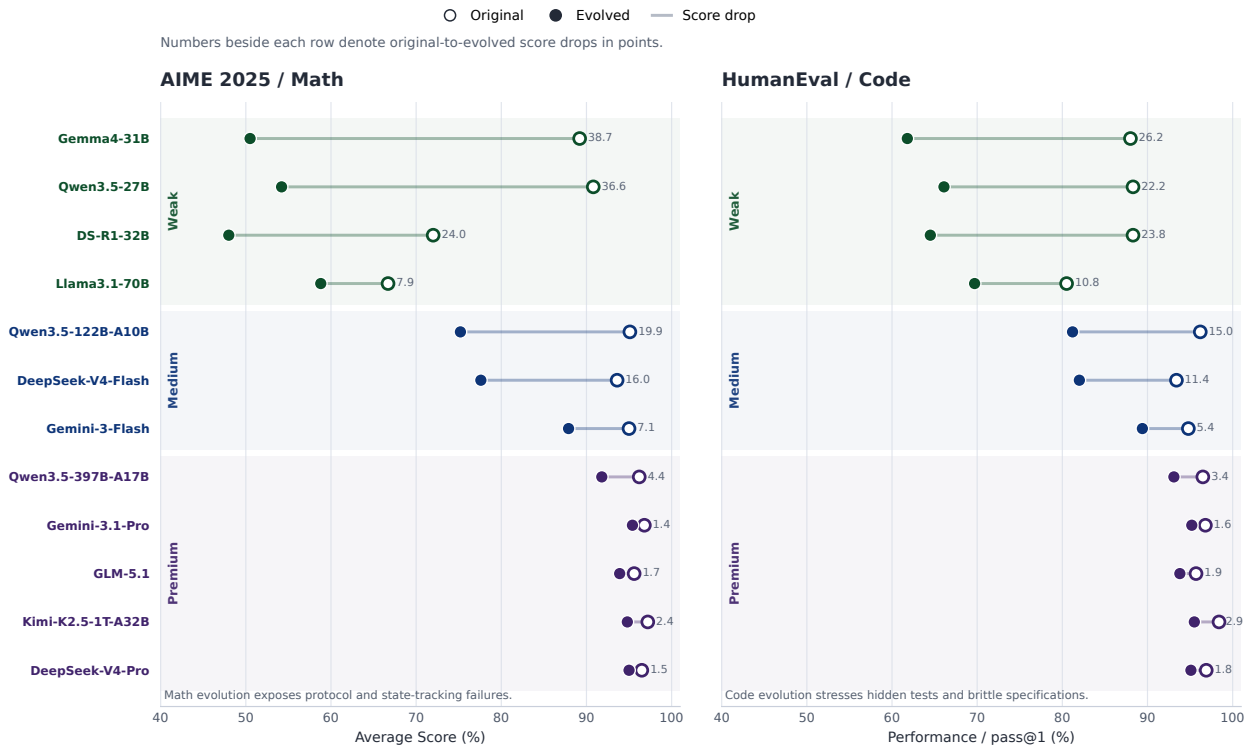

Figure 2. Expanded model-panel comparison for AIME and HumanEval before and after benchmark evolution. Colors denote model tiers, and line lengths denote original-to-evolved score drops.

Table 3. Representative AIME performance patterns discovered by SEB.

Pattern	Diagnostic manifestation	Weak	Medium	Premium
Answer forcing	outputs a number despite no valid solution	vulnerable	partly robust	robust
Uniqueness blindness	treats one witness as a unique answer	vulnerable	vulnerable	robust
Symbolic protocol fragility	solves but violates extraction format	vulnerable	robust	robust
Transfer-state counting	loses boundary statistic in local DP	vulnerable	vulnerable	almost robust
Merged carry streams	assumes recursive streams are independent	vulnerable	robust	robust
Digit-DP residues	mishandles carry interaction with modulo test	vulnerable	robust	robust

harness adaptation. Each generated item is then checked by two agent paths: the validity agent verifies domain preservation and protocol consistency, while the reliability agent verifies answer correctness. In each path, a verifier first produces a judgement, and a meta-verifier then checks both the judgement and its confidence before the item can be accepted.

IAA analysis for benchmark annotation. Since SEB includes human annotation for accepting or rejecting candidate benchmark versions, we quantify inter-annotator agreement using Krippendorff's α (Krippendorff, 2018). As shown in Table 4, three annotators independently label each AIME and HumanEval candidate, and the final audit label is obtained by majority voting. The overall agreement of 0.859 indicates that the annotation protocol is sufficiently stable despite the subjective boundary between valid performance-pattern exploitation and invalid hardening.

Verifier and meta-verifier validation. Validity verification is the primary safeguard against benchmark drift: an evolved item should be rejected if it departs from the seed benchmark's domain, topic range, difficulty envelope, or evaluation protocol. Figure 3 illustrates how this safeguard is paired with answer-reliability auditing before a candidate can enter the release. We evaluate the validity agent against the human-adjudicated binary validity labels produced by the three-annotator majority vote in Table 4. As shown in Table 5, the validity agent achieves a Pearson correlation of 0.912 with the majority-vote labels (Pearson, 1901). We further assess answer reliability on a random 20-item human audit set, where the reliability agent verifies generated answers and reference solutions with 0.950 accuracy. These two experiments prove the reliability of the meta-

Table 4. Inter-Annotator Agreement analysis for human annotation in valid judgement.

Annotation Set	Count	Krippendorff's α [95% CI]
Valid Question	78	0.872 [−0.058, +0.056]
Invalid Question	88	0.846 [−0.064, +0.061]
Overall Candidates	**166**	**0.859** [−0.043, +0.041]

Table 5. Validation of the verifier and meta-verifier pipeline on human labeled data.

Agent Type	Sample Size	Metric	Result
validity agent	166	Pearson ρ	0.912
reliability agent	20	Accuracy	0.950

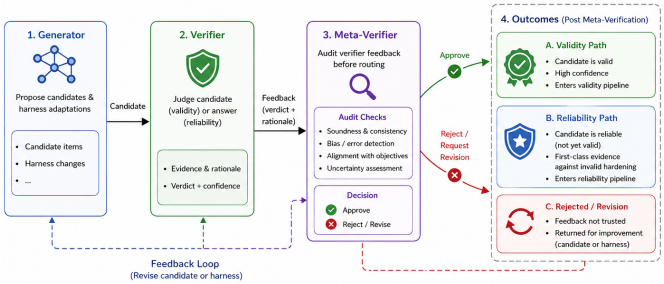

Figure 3. Meta-verification strategy for generated benchmark items. The pipeline separates candidate generation from two audit paths: validity checks preserve the seed benchmark's measurement contract, while reliability checks verify answers, tests, and scoring behavior. Meta-verifiers review both the evidence and confidence of each path before an item is approved, rejected, or returned for revision.

verification strategy, demonstrating an exceptionally high alignment with human judgment, justifying its effectiveness in the autonomous benchmark evolution framework.

6. Conclusion

We presented SEB, a multi-agent framework for reviving saturated benchmarks. Instead of making a benchmark arbitrarily harder, SEB preserves the original evaluation scope and evolves it through validated performance patterns observed in current LLMs. We instantiated SEB on two tracks, AIME 2025 for mathematical reasoning and HumanEval for function-level code generation. The resulting AIME-EVOLVE and CODE-EVOLVE releases reduce upper-tail compression and restore clearer separation while remaining within the seed benchmarks' intended domains. More broadly, SEB reframes benchmark saturation as a lifecycle problem rather than an endpoint of evaluation. When public benchmarks are repeatedly optimized and lose discriminative power, controlled evolution can preserve their original measurement scope while restoring their ability to evaluate frontier models and reveal meaningful performance differences. In this sense, benchmark revival provides a principled path to make benchmarks live again.

References

Austin, J., Odena, A., Nye, M., Bosma, M., Michalewski, H., Dohan, D., Jiang, E., Cai, C., Terry, M., Le, Q., and Sutton, C. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Chen, M., Tworek, J., Jun, H., Yuan, Q., Pinto, H. P. d. O., Kaplan, J., Edwards, H., Burda, Y., Joseph, N., Brockman, G., Ray, A., Puri, R., Krueger, G., Petrov, M., Khlaaf, H., Sastry, G., Mishkin, P., Chan, B., Gray, S., Ryder, N., Pavlov, M., Power, A., Kaiser, L., Bavarian, M., Winter, C., Tillet, P., Such, F. P., Cummings, D., Plappert, M., Chantzis, F., Barnes, E., Herbert-Voss, A., Guss, W. H., Nichol, A., Paino, A., Tezak, N., Tang, J., Babuschkin, I., Balaji, S., Jain, S., Saunders, W., Hesse, C., Carr, A. N., Leike, J., Achiam, J., Misra, V., Morikawa, E., Radford, A., Knight, M., Brundage, M., Murati, M., Mayer, K., Welinder, P., McGrew, B., Amodei, D., McCandlish, S., Sutskever, I., and Zaremba, W. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Cobbe, K., Kosaraju, V., Bavarian, M., Chen, M., Jun, H., Kaiser, L., Plappert, M., Tworek, J., Hilton, J., Nakano, R., Hesse, C., and Schulman, J. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Deng, X., Da, J., Pan, E., He, Y. Y., Ide, C., Garg, K., Lauffer, N., Park, A., Pasari, N., Rane, C., Sampath, K., Krishnan, M., Kundurthy, S., Hendryx, S., Wang, Z., Zhang, C. B. C., Jacobson, N., Liu, B., and Kenstler, B. SWE-Bench Pro: Can AI agents solve long-horizon software engineering tasks? *arXiv preprint arXiv:2509.16941*, 2025.

Guo, D., Zhou, T., Liu, D., Qian, C., Ren, Q., Shao, S., Fan, Z., Fung, Y. R., Wang, K., Zhang, L., and Shao, J. Towards self-evolving benchmarks: Synthesizing agent trajectories via test-time exploration under validate-by-reproduce paradigm. *arXiv preprint arXiv:2510.00415*, 2025.

He, C., Luo, R., Bai, Y., Hu, S., Thai, Z. L., Shen, J., Hu, J., Han, X., Huang, Y., Zhang, Y., Liu, J., Qi, L., Liu, Z., and Sun, M. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. *arXiv preprint arXiv:2402.14008*, 2024.

He, Y., Liu, J., Liu, Y., Li, Y., Cao, T., Hu, Z., Xu, X., and Hooi, B. EvoTest: Evolutionary test-time learn-

ing for self-improving agentic systems. *arXiv preprint arXiv:2510.13220*, 2025.

Hendrycks, D., Burns, C., Kadavath, S., Arora, A., Basart, S., Tang, E., Song, D., and Steinhardt, J. Measuring mathematical problem solving with the math dataset. In *NeurIPS Datasets and Benchmarks*, 2021.

Jain, N., Han, K., Gu, A., Li, W.-D., Yan, F., Zhang, T., Wang, S. I., Solar-Lezama, A., Sen, K., and Stoica, I. LiveCodeBench: Holistic and contamination free evaluation of large language models for code. *arXiv preprint arXiv:2403.07974*, 2024.

Jimenez, C. E., Yang, J., Wettig, A., Yao, S., Pei, K., Press, O., and Narasimhan, K. SWE-bench: Can language models resolve real-world GitHub issues? In *International Conference on Learning Representations*, 2024.

Kiela, D., Bartolo, M., Nie, Y., Kaushik, D., Geiger, A., Wu, Z., Vidgen, B., Prasad, G., Singh, A., Ringshia, P., Ma, Z., Thrush, T., Riedel, S., Waseem, Z., Stenetorp, P., Jia, R., Bansal, M., Potts, C., and Williams, A. Dynabench: Rethinking benchmarking in NLP. In *Conference of the North American Chapter of the Association for Computational Linguistics*, 2021.

Krippendorff, K. *Content Analysis: An Introduction to Its Methodology*. SAGE Publications, 4 edition, 2018.

Li, G., Xie, Y., Liu, Y., Dong, Z., Pan, X., Zheng, T., Choi, J., Morais, M. J., Jha, B., Mishra, S., Zhou, B., Luo, C., Cheng, M. X., and Song, D. The world won't stay still: Programmable evolution for agent benchmarks. *arXiv preprint arXiv:2603.05910*, 2026.

Li, T., Chiang, W.-L., Frick, E., Dunlap, L., Zhu, B., Gonzalez, J. E., and Stoica, I. From crowdsourced data to high-quality benchmarks: Arena-Hard and BenchBuilder pipeline. *arXiv preprint arXiv:2406.11939*, 2024.

Liang, P., Bommasani, R., Lee, T., Tsipras, D., Soylu, D., Yasunaga, M., Zhang, Y., Narayanan, D., Wu, Y., Kumar, A., Newman, B., Yuan, B., Yan, B., Zhang, C., Cosgrove, C., Manning, C. D., Re, C., Acosta-Navas, D., Hudson, D. A., Zelikman, E., Durmus, E., Ladhak, F., Rong, F., Ren, H., Yao, H., Wang, J., Santhanam, K., Orr, L., Zheng, L., Yuksekgonul, M., Suzgun, M., Kim, N., Guha, N., Chatterji, N., Khattab, O., Henderson, P., Huang, Q., Chi, R., Xie, S. M., Santurkar, S., Ganguli, S., Hashimoto, T., Icard, T., Zhang, T., Chaudhary, V., Wang, W., Li, X., Mai, Y., Zhang, Y., and Koreeda, Y. Holistic evaluation of language models. *arXiv preprint arXiv:2211.09110*, 2022.

Liu, J., Xia, C. S., Wang, Y., and Zhang, L. Is your code generated by ChatGPT really correct? rigorous evaluation of large language models for code generation. In *Advances in Neural Information Processing Systems*, 2023.

Luong, T., Hwang, D., Nguyen, H. H., Ghiasi, G., Chervonyi, Y., Seo, I., Kim, J., Bingham, G., Lee, J., Mishra, S., Zhai, A., Hu, C. H., Michalewski, H., Kim, J., Ahn, J., Bae, J., Song, X., Trinh, T. H., Le, Q. V., and Jung, J. Towards robust mathematical reasoning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, 2025. URL https://aclanthology.org/2025.emnlp-main.1794/.

Mathematical Association of America. American invitational mathematics examination. https://maa.org/math-competitions/american-invitational-mathematics-examination-aime/, 2026. Accessed 2026-05-03.

Pearson, K. Liii. on lines and planes of closest fit to systems of points in space. *The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science*, 2(11): 559–572, 1901.

Shao, Z., Luo, Y., Lu, C., Ren, Z. Z., Hu, J., Ye, T., Gou, Z., Ma, S., and Zhang, X. DeepSeekMath-V2: Towards self-verifiable mathematical reasoning. *arXiv preprint arXiv:2511.22570*, 2025.

Srivastava, A., Rastogi, A., Rao, A., Shoeb, A. A. M., Abid, A., Fisch, A., Brown, A. R., Santoro, A., Gupta, A., Garriga-Alonso, A., Kluska, A., Lewkowycz, A., Agarwal, A., Power, A., Ray, A., Warstadt, A., Kocurek, A. W., Safaya, A., Tazarv, A., Xiang, A., Parrish, A., Nie, A., Hussain, A., Askell, A., Dsouza, A., Rahane, A., Iyer, A. S., Andreassen, A., Santilli, A., Stuhlmuller, A., Dai, A. M., La, A. D., Lampinen, A. K., Zou, A., Jiang, A., Chen, A., Vuong, A., Gupta, A., Gottardi, A., Norelli, A., Venkatesh, A., Gholamidavoodi, A., Tabassum, A., Menezes, A., Kirubarajan, A., Mullokandov, A., Sabharwal, A., Herrick, A., Efrat, A., Erdem, A., Karakas, A., Roberts, B. R., Loe, B. S., Zoph, B., Bojanowski, B., Ozyurt, B., Hedayatnia, B., Neyshabur, B., Inden, B., Stein, B., Ekmekci, B., Lin, B. Y., Howald, B., Orinion, C., Diao, C., Dour, C., Stinson, C., Argueta, C., Ramírez, C. F., Singh, C., Rathkopf, C., Meng, C., Baral, C., Wu, C., Callison-Burch, C., Waites, C., Voigt, C., Manning, C. D., Potts, C., Ramirez, C., Rivera, C. E., Siro, C., Raffel, C., Ashcraft, C., Garbacea, C., Sileo, D., Garrette, D., Hendrycks, D., Kilman, D., Roth, D., Freeman, D., Khashabi, D., Levy, D., Gonzalez, D. M., Perszyk, D., Hernandez, D., Chen, D., Ippolito, D., Gilboa, D., Dohan, D., Drakard, D., Jurgens, D., Datta, D., Ganguli, D., Emelin, D., Kleyko, D., Yuret, D., Chen, D., Tam, D., Hupkes, D., Misra, D., Buzan, D., Mollo, D. C., Yang, D., Lee, D.-H., Schrader, D., Shutova, E., Cubuk, E. D., Segal, E., Hagerman, E., Barnes, E., Donoway,

E., Pavlick, E., Rodola, E., Lam, E., Chu, E., Tang, E., Erdem, E., Chang, E., Chi, E. A., Dyer, E., Jerzak, E., Kim, E., Manyasi, E. E., Zheltonozhskii, E., Xia, F., Siar, F., Martínez-Plumed, F., Happé, F., Chollet, F., Rong, F., Mishra, G., Winata, G. I., de Melo, G., Kruszewski, G., Parascandolo, G., Mariani, G., Wang, G., Jaimovitch-López, G., Betz, G., Gur-Ari, G., Galijasevic, H., Kim, H., Rashkin, H., Hajishirzi, H., Mehta, H., Bogar, H., Shevlin, H., Schütze, H., Yakura, H., Zhang, H., Wong, H., Ng, I., Noble, I., Jumelet, J., Geissinger, J., Kernion, J., Hilton, J., Lee, J., Fisac, J. F., Simon, J. B., Koppel, J., Zheng, J., Zou, J., Kocoń, J., Thompson, J., Kaplan, J., Radom, J., Sohl-Dickstein, J., Phang, J., Wei, J., Yosinski, J., Novikova, J., Bosscher, J., Marsh, J., Kim, J., Taal, J., Engel, J., Alabi, J., Xu, J., Song, J., Tang, J., Waweru, J., Burden, J., Miller, J., Balis, J. U., Berant, J., Frohberg, J., Rozen, J., Hernandez-Orallo, J., Boudeman, J., Jones, J., Tenenbaum, J., Rule, J. S., Chua, J., Kanclerz, K., Livescu, K., Krauth, K., Gopalakrishnan, K., Ignatyeva, K., Markert, K., Dhole, K. D., Gimpel, K., Omondi, K. O., Mathewson, K., Chiafullo, K., Shkaruta, K., Shridhar, K., McDonell, K., Richardson, K., Reynolds, L., Gao, L., Zhang, L., Dugan, L., Qin, L., Contreras-Ochando, L., Morency, L.-P., Moschella, L., Lam, L., Noble, L., Schmidt, L., He, L., Colón, L. O., Metz, L., Şenel, L. K., Bosma, M., Sap, M., ter Hoeve, M., Farooqi, M., Faruqui, M., Mazeika, M., Baturan, M., Marelli, M., Maru, M., Quintana, M. J. R., Tolkiehn, M., Giulianelli, M., Lewis, M., Potthast, M., Leavitt, M. L., Hagen, M., Schubert, M., Baitemirova, M., Arnaud, M., McElrath, M., Yee, M. A., Cohen, M., Gu, M., Ivanitskiy, M., Starritt, M., Strube, M., Swedrowski, M., Bevilacqua, M., Yasunaga, M., Kale, M., Cain, M., Xu, M., Suzgun, M., Tiwari, M., Bansal, M., Aminnaseri, M., Geva, M., Gheini, M., T, M. V., Peng, N., Chi, N. L., Muennighoff, N., Kuksov, N., Fiedel, N., Weller, O., Abramson, J., Ahuja, K., D'Amour, A., Hessel, J., Stillerman, J., Heath, J., Prasad, A., Aravkin, A., Freitas, A., Chandu, K. R., Tadepalli, P., Jiang, Z., Gururangan, S., Wieting, J., Kiela, D., Clark, P., Bowman, S. R., Wiegreffe, S., Gardner, M., Hsu, J., Huang, J., Xiong, C., Lamm, M., Yogatama, D., Cohen, S., Ong, D., Plank, B., Krueger, D., Hendrycks, D., Schuster, T., Gupta, S. L., Potts, C., Qiu, X., Cohen, W. W., Zhou, D., Hassabis, D., Hinton, G., LeCun, Y., Bengio, Y., Li, Y., Manning, C. D., Brown, T. B., Schulman, J., Stiennon, N., Ziegler, D. M., Wu, J., Amodei, D., Clark, J., Kaplan, J., and Sutskever, I. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

Wang, S., Long, Z., Fan, Z., Huang, X., and Wei, Z. Benchmark self-evolving: A multi-agent framework for dynamic LLM evaluation. In *Proceedings of the 31st International Conference on Computational Linguistics*, pp. 3310–3328, 2025.

Wei, T., Sachdeva, N., Coleman, B., He, Z., Bei, Y., Ning, X., Ai, M., Li, Y., He, J., Chi, E. H., Wang, C., Chen, S., Pereira, F., Kang, W.-C., and Cheng, D. Z. Evo-Memory: Benchmarking LLM agent test-time learning with self-evolving memory. *arXiv preprint arXiv:2511.20857*, 2025.

White, C., Dooley, S., Roberts, M., Pal, A., Feuer, B., Jain, S., Shwartz-Ziv, R., Jain, N., Saifullah, K., Naidu, S., Hegde, C., LeCun, Y., Goldstein, T., and Goldblum, M. LiveBench: A challenging, contamination-free LLM benchmark. *arXiv preprint arXiv:2406.19314*, 2024.

Xia, P., Zeng, K., Liu, J., Qin, C., Wu, F., Zhou, Y., Xiong, C., and Yao, H. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.

Xu, K., Mao, Y., Guan, X., and Feng, Z. Web-Bench: A LLM code benchmark based on web standards and frameworks. *arXiv preprint arXiv:2505.07473*, 2025.

Yuan, J., Jin, T., Chen, W., Liu, Z., Liu, Z., and Sun, M. SE-Bench: Benchmarking self-evolution with knowledge internalization. *arXiv preprint arXiv:2602.04811*, 2026.

Zhang, W., Wang, J., Liang, L., Zhao, Y., Wen, H., and Zhao, Z. EvoCodeBench: A human-performance benchmark for self-evolving LLM-driven coding systems. *arXiv preprint arXiv:2602.10171*, 2026.

Zheng, L., Chiang, W.-L., Sheng, Y., Zhuang, S., Wu, Z., Zhuang, Y., Lin, Z., Li, Z., Li, D., Xing, E. P., Zhang, H., Gonzalez, J. E., and Stoica, I. Judging LLM-as-a-judge with MT-Bench and chatbot arena. In *Advances in Neural Information Processing Systems*, 2023.

A. Benchmark Evolution Details

In both tracks, candidate benchmarks are generated from anonymized model-behavior traces, adapted by the harness-side agent, filtered for reliability and domain preservation, evaluated on the same model pools, and retained only when the revival gate and analysis confirmation agree. This section reports the depth-level patterns, retry behavior, candidate filtering, and audit evidence. We use *candidate leaves* to denote route-level benchmark candidates produced by one planner direction and its paired harness adaptation. A leaf may contain multiple items; the final release size is therefore not obtained by subtracting rejected leaves from generated leaves, but by integrating validated items from accepted leaves.

AIME-EVOLVE audit summary. The AIME run uses AIME 2025 as the root benchmark and advances through three accepted depths. At the first depth, the planner proposes routes around direct numerical hardening, answer forcing, hidden uniqueness assumptions, and symbolic answer formats. Direct hardening is mostly rejected at the gate because it lowers all representatives similarly and does not create adjacent-tier separation. The accepted route instead preserves the AIME-style setting but changes the failure condition: a model must reason about whether a requested answer is unique, forced by a hidden constraint, or invalid under the stated protocol. At the second depth, one same-depth replan is triggered after a batch of protocol-heavy candidates is rejected: the adapter agent can implement the changed parser, but the validity agent marks several leaves as drifting from contest-math answer conventions. The replanned directions keep the same mathematical scope while targeting transfer-state counting and coupled carry streams. At the third depth, the search concentrates on digit-DP residues and symbolic edge cases that remain solvable by careful reasoning but repeatedly break compressed solution templates observed in the anonymous traces. Protocol extensions such as exact expressions or `NO_SOLUTION` are accepted only when the adapter agent updates the prompt, parser, scorer, and equivalence checker jointly.

CODE-EVOLVE audit summary. The code run uses HumanEval as the root benchmark and advances through five accepted depths. The first accepted depth mainly modifies boundary conditions that are already implied by the original docstrings, such as empty inputs, repeated values, stable ordering, and numerical corner cases. Several leaves are rejected by the reliability agent because the generated reference implementation and hidden tests disagree on edge-case semantics. Later depths add specification constraints that require models to maintain local invariants across helper logic, stateful iteration, or composed input transformations, while still keeping the unit-test-only HumanEval harness. Two same-depth replans occur in the code track: one after all candidate leaves become repository-level tasks rather than function-level tasks, and one after hidden tests overfit to one reference implementation instead of checking the intended behavior. The accepted leaves are those for which the adapter agent can express the new requirement through the docstring, visible examples, hidden tests, and answer-extraction rule without changing the benchmark into software-engineering issue resolution.

Gate and confirmation protocol. For each leaf, the system first performs non-agentic quantitative screening using the revival indicators defined in Section 4.1. Leaves that do not improve adjacent-tier separation or that collapse premium performance below the allowed range are immediately marked invalid. This accounts for many gate rejections: some candidates are correct and in-domain, but they merely move all models downward together; others make the weak representative fail while leaving the medium representative unchanged, which does not restore a monotone tier signal. Only threshold-passing leaves are sent to the analysis agent. The analysis agent then checks whether the score change is attributable to a coherent performance pattern visible in traces, such as premature answer commitment, missing uniqueness checks, brittle symbolic parsing, state-transfer errors, or hidden-test assumption failures. If the trace evidence is inconsistent, if failures come from malformed instructions, or if the effect disappears under the adapted harness, the leaf is counted as an analysis rejection.

Failed evolution attempts. Rejected candidates are retained as audit evidence because they clarify what SEB does not treat as a valid benchmark revival. In the math track, invalid leaves include arithmetic-only variants that are harder but not more diagnostic, symbolic-answer variants whose scoring rule is underspecified, and construction problems where multiple valid answers remain after the adapter update. In the code track, invalid leaves include tasks whose hidden tests encode an unstated convention, tasks whose reference solution fails under randomized checks, and tasks whose specification expands into multi-file repository behavior. Several rejected leaves are still useful negative evidence: their failure reports are fed back to the planner agent, which avoids repeating the same drift mode at the next accepted depth. These failures motivate the final mixed-pattern release: a useful evolved benchmark should contain diverse validated probes, not a homogeneous set of adversarial variants.

Final release construction. After the accepted depths terminate, the release is built from validated items rather than from entire intermediate leaves. For each retained item, we keep the parent direction, the adapted harness files, the reference answer or solution, the validity and reliability reports,

and the analysis-agent rationale. Near-duplicate items are removed when they target the same pattern with the same surface transformation, even if both passed the gate. The resulting releases contain 26 math items and 52 code items, each tied to at least one confirmed performance pattern and scored with an explicitly declared protocol. Intermediate rollouts are kept as audit artifacts and are not used as standalone main-result benchmarks.

B. Release Artifact Manifest

Each evolved benchmark release is accompanied by a manifest that records the concrete artifacts needed to reproduce and audit the release. This manifest is not part of the evolution algorithm itself; it is the packaging layer that connects the final benchmark in Table 2 with the intermediate rollouts, harness adaptations, validation reports, and anonymized evaluation traces generated during the search. Table 7 lists the manifest fields used for both AIME-EVOLVE and CODE-EVOLVE.

C. Prompt Library

This section details the foundational prompt architecture employed by agents in SEB. Placeholders enclosed in `{ }` denote dynamic variables or reference macros that are instantiated at runtime according to the agent's role, project context, and task specifications. The operational responsibilities of each agent type, along with their corresponding prompt design, are provided below.

The library includes prompts for research agent survey, planner agent direction generation, evolution sub-agent task generation, adapter agent harness adaptation, reliability agent validation, validity agent checking, analysis agent diagnostic judging, reflection agent same-depth replanning, and answer extraction. Runtime context bundles attach the benchmark dossier, planner-visible anonymous traces, candidate manifests, adapted harness files, and validation logs, which makes evolution runs auditable.

C.1. Research Agent Prompt

The research agent prompt asks the model to survey the original benchmark and its surrounding artifacts, including the paper, arXiv versions, GitHub repository, Hugging Face datasets, README files, scoring scripts, known limitations, and public leaderboard behavior. The expected output is a structured report describing the benchmark's intended capability, task distribution, protocol, scoring rule, provenance, contamination risks, and domain boundaries. This report becomes the contract against which later evolution is judged.

Prompt for Research Agent

Role

You are the research agent for a benchmark revival run. Your responsibility is to reconstruct the measurement contract of the seed benchmark before any evolution is attempted.

Context

- seed benchmark name and version: `{seed_benchmark}`
- benchmark domain and intended task family: `{domain_summary}`
- available papers, repositories, dataset cards, READMEs, scoring scripts, issue reports, and leaderboard snapshots: `{source_bundle}`
- known model results and saturation signals, if available: `{public_results}`

Instructions

Survey the provided sources and produce a benchmark dossier. Identify the benchmark's intended capability, task distribution, difficulty range, scoring protocol, answer format, allowed tools, contamination risks, public leaderboard behavior, and known limitations. Distinguish the benchmark's core measurement target from incidental implementation details. Explicitly state what later evolution is allowed to change and what must remain fixed to avoid benchmark drift. Your response should include the following fields: `measurement_target`, `task_taxonomy`, `difficulty_envelope`, `protocol_contract`, `scoring_rule`, `answer_format`, `domain_boundaries`, `known_saturation_evidence`, `contamination_risks`, `allowed_evolution_surface`, and `forbidden_drift_cases`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{research_agent_output_template}`

C.2. Planner Agent Prompt

The planner agent prompt consumes the survey report and a double-blind observation bundle. Each evaluated system is represented by a stable anonymous identifier with item-level outcomes, parsed answers, and trace summaries, but model names and tier labels are withheld. This prevents the planner from intentionally overfitting evolution directions to a known weak, medium, or premium model, while still allowing it to discover recurring behavioral failures from the anonymous traces. It first identifies macro evolution directions, each framed as a performance-pattern hypothesis, and then dispatches these directions to evolution sub-agents. A direction may test, for example, whether models over-assume finite answers, fail to verify uniqueness, mishandle symbolic answer protocols, or drop boundary conditions in

Table 6. Audit summary for the two evolved benchmark tracks. Candidate leaves are route-level benchmark candidates generated during the search. QC rejection denotes reliability or domain-control failure; gate rejection denotes failure to pass the quantitative revival criterion; analysis rejection denotes threshold-passing candidates whose score changes were not confirmed as coherent performance-pattern exploit points. Final items are the integrated release items, not the number of valid route-level candidates.

Track	Accepted depths	Candidate leaves	QC rejected	Gate rejected	Analysis rejected	Final items
AIME-EVOLVE	3	58	12	19	9	26
CODE-EVOLVE	5	96	18	29	15	52

Table 7. Release artifact manifest fields.

Field	Recorded content
Release identity	Track name, release version, seed benchmark, accepted depth count, and final item count.
Item files	Final math problems or code tasks, reference answers or solutions, and item-level metadata.
Harness files	Prompts, answer instructions, parsers, equivalence rules, tests, and scoring scripts used for evaluation.
Pattern tags	Confirmed performance patterns associated with each retained item and its parent leaf.
Evaluation panel	Representative models, decoding settings, sample count, raw outputs, parsed answers, and correctness labels.
Quantitative gate	μ_K, δ_K, retained premium score, and the gate decision for each evaluated leaf.
Validity reports	Validity-agent and meta-verifier decisions on domain preservation, topic range, difficulty, protocol consistency, and drift risk.
Reliability reports	Reference-answer checks, solver or unit-test evidence, meta-verifier decisions, and human spot-check records.
Audit trail	Rejected leaves, rejection reason, same-depth replan events, and analysis-agent confirmation or rejection rationales.
Anonymous traces	Planner-visible model identifiers and item-level behaviors, excluding model names and tier labels.
Private mapping	Non-public audit map from anonymous identifiers to model names and tiers, used only for reproducibility by the authors.
Release notes	Differences from the seed benchmark, intended use, known caveats, and repository file organization.

counting problems.

Prompt for Planner Agent

Role

You are the planner agent for benchmark evolution. Your job is to propose evolution directions that may restore diagnostic separation while preserving the seed benchmark's measurement target.

Context

- benchmark dossier from the research agent: `{benchmark_dossier}`
- anonymized model-behavior bundle: `{blind_evaluation_traces}`
- current benchmark node and depth: `{current_node}`, `{depth}`
- valid and invalid sibling histories: `{lineage_feedback}`
- rejected candidates and reasons from prior rounds: `{audit_log}`
- target number of parallel directions: `{route_budget}`

Instructions

Inspect the anonymous item-level failures, parsed answers, format errors, execution errors, and reasoning summaries. Infer recurring behavior patterns without using model names or tier labels. Propose a small set of macro evolution directions, where each direction is a testable performance-pattern hypothesis. Each direction must specify the intended in-domain change, why it should preserve the benchmark contract, what model behavior it is expected to expose, and what evidence would make the direction invalid. Your response should contain `directions`, a JSON list in which each entry contains `direction_id`, `performance_pattern_hypothesis`, `in_domain_evolution`, `required_harness_adaptation`, `expected_separability_signal`, `domain_preservation_rationale`, and `invalidity_risks`. Do not mention model names, provider names, or tier labels.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{planner_agent_output_template}`

C.3. Evolution Sub-Agent Prompt

Each evolution sub-agent receives one direction point and generates the task side of one candidate benchmark version. The prompt requires the sub-agent to preserve the seed

domain, state the targeted performance pattern, write the evolved items, provide reference reasoning or code, and emit metadata needed by validation and scoring. Parallel sub-agents correspond to parallel leaf nodes under the same benchmark parent.

Prompt for Evolution Sub-Agent

Role

You are an evolution sub-agent assigned to one benchmark evolution direction. Your job is to generate candidate benchmark items that instantiate the assigned performance-pattern hypothesis without drifting away from the seed benchmark.

Context

- assigned direction from the planner: `{direction}`
- benchmark dossier and forbidden drift cases: `{benchmark_dossier}`
- parent benchmark examples and protocol: `{parent_benchmark}`
- item budget and diversity requirements: `{candidate_budget}`
- prior accepted and rejected evolution points: `{lineage_history}`

Instructions

Generate candidate items that preserve the seed benchmark's domain, format, and difficulty envelope. Each item must clearly target the assigned performance pattern and include sufficient metadata for validation. For math candidates, provide the problem statement, final answer, and reference reasoning. For code candidates, provide the function signature, prompt, reference implementation or oracle, public examples, and hidden-test rationale. Avoid ambiguity, unsupported protocol changes, or arbitrary difficulty escalation. Your candidate manifest should contain `candidate_id`, `parent_id`, `targeted_pattern`, `items`, `reference_evidence`, `expected_failure_mode`, `domain_preservation_notes`, `validation_requirements`, and `harness_needs`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{evolution_subagent_output_template}`

C.4. Adapter Agent Prompt

The adapter agent prompt receives the evolved task set and the original benchmark harness. It adapts the external evaluation interface required by the candidate benchmark, including prompt-engineering variants, scoring guidelines, answer instructions, parsers, scorers, and equivalence rules. Its output is not a new task distribution by itself; it is the harness-side counterpart to the evolution sub-agent's generated tasks, ensuring that any protocol change required by the leaf candidate is explicit, reproducible, and not a scoring artifact.

Prompt for Adapter Agent

Role

You are the adapter agent for a candidate benchmark. Your job is to adapt the external evaluation harness so that the evolved items can be evaluated fairly and reproducibly.

Context

- candidate manifest from the evolution sub-agent: `{candidate_manifest}`
- seed benchmark prompt, parser, scorer, and answer format: `{seed_harness}`
- required harness changes requested by the candidate: `{harness_needs}`
- protocol constraints from the benchmark dossier: `{protocol_contract}`

Instructions

Produce the minimal harness adaptation needed for the candidate. Specify prompt templates, answer instructions, parser changes, scorer changes, equivalence rules, hidden-test construction, and failure handling. Every change must be justified by the candidate's measurement goal and must not create a scoring shortcut, unfair format trap, or mismatch between instructions and evaluation. Your adapted harness should include `prompt_template`, `answer_instruction`, `parser_spec`, `scorer_spec`, `equivalence_rules`, `test_oracle` where applicable, `protocol_change_rationale`, and `artifact_risk_checks`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{adapter_agent_output_template}`

C.5. Reliability Agent Prompt

The reliability agent prompt checks ground-truth answers, reference solutions, premises, and reproducibility. These prompts audit the candidate benchmark produced by the generation side rather than proposing new evolution directions.

Prompt for Reliability Agent

Role

You are the reliability agent. Your job is to verify whether the candidate benchmark's answers, reference solutions, premises, and reproducibility evidence are correct.

Context

- candidate manifest and items: `{candidate_manifest}`
- reference answer, reasoning, implementation, or oracle: `{reference_evidence}`
- adapted harness: `{adapted_harness}`

- seed benchmark protocol constraints: `{protocol_contract}`

Instructions

Independently verify the correctness of each candidate item. For math, recompute the answer and check that all assumptions are stated. For code, inspect the reference implementation, public examples, hidden tests, and oracle behavior. Reject candidates with incorrect answers, inconsistent premises, non-reproducible evaluation behavior, brittle tests, or ambiguous scoring. Your reliability report should contain `decision` in `{reliable, unreliable, uncertain}`, `verified_items`, `failed_items`, `minimal_verification_steps`, `failure_reason`, and `recommended_fixes`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{reliability_agent_output_template}`

C.6. Validity Agent Prompt

The validity agent prompt checks topic, range, difficulty, and domain drift, ensuring that score changes are not obtained by moving outside the seed benchmark's intended scope. These prompts audit the candidate benchmark produced by the generation side rather than proposing new evolution directions.

Prompt for Validity Agent

Role

You are the validity agent. Your job is to determine whether a candidate benchmark remains a valid in-domain evolution of the seed benchmark.

Context

- benchmark dossier and domain boundaries: `{benchmark_dossier}`
- candidate manifest and evolved items: `{candidate_manifest}`
- adapted harness and protocol changes: `{adapted_harness}`
- parent benchmark examples and accepted candidates: `{lineage_context}`

Instructions

Judge whether the candidate preserves the seed benchmark's intended domain, topic range, task form, and difficulty envelope. Identify any benchmark drift, unsupported protocol change, redundant candidate, hidden ambiguity, or arbitrary hardening. Do not evaluate whether the candidate improves separation; evaluate whether it is a valid candidate for evaluation. Your validity report should contain `decision` in `{valid, invalid, redundant, uncertain}`, `domain_check`, `difficulty_check`, `protocol_check`, `redundancy_check`, `drift_risks`, and `minimal_audit_evidence`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{validity_agent_output_template}`

C.7. Analysis Agent Prompt

The analysis agent prompt is invoked only for leaves that pass the quantitative revival threshold. It determines whether a threshold-passing leaf is a valid evolution node by connecting observed score changes to a coherent exploit feature or performance pattern. It rejects leaves whose apparent gains come from ambiguity, noise, label error, scoring artifacts, or invalid difficulty escalation.

Prompt for Analysis Agent

Role

You are the analysis agent. Your job is to decide whether a threshold-passing candidate is a valid evolution node supported by a coherent performance pattern.

Context

- candidate manifest and targeted pattern: `{candidate_manifest}`
- evaluation results on the same model tiers: `{candidate_results}`
- parent benchmark results and revival indicators: `{parent_results}`
- reliability and validity reports: `{qc_reports}`
- anonymized item-level traces and parsed answers: `{behavior_traces}`

Instructions

Determine whether the observed score change is explained by the targeted performance pattern. Compare failures across items and anonymous systems. Accept the candidate only if the effect is recurring, in-domain, reliable, and not caused by parser artifacts, ambiguous wording, answer errors, random noise, or invalid difficulty escalation. State whether the candidate should become a valid node for the next depth. Your analysis report should contain `decision` in `{valid, invalid}`, `confirmed_pattern`, `evidence_from_traces`, `separability_interpretation`, `artifact_checks`, `rejection_reason` if any, and `feedback_for_planner`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{analysis_agent_output_template}`

C.8. Reflection Agent Prompt

If every leaf in a round is rejected, either by the threshold gate or by the analysis agent, the reflection agent prompt

825 826 827 828 829 830 831 832 833 834 835 836 837 838 839 840 841 842 843 844 845 846 847 848 849 850 851 852 853 854 855 856 857 858 859 860 861 862 863 864 865 866 867 868 869 870 871 872 873 874 875 876 877 878 879

receives the full same-depth record and summarizes why the attempted directions failed, enabling the planner agent to propose a new same-depth round without increasing depth.

Prompt for Reflection Agent

Role

You are the reflection agent for same-depth replanning. Your job is to explain why all candidate leaves failed and produce actionable feedback for the planner.

Context

- current parent benchmark node: `{current_node}`
- all attempted directions in the failed round: `{attempted_directions}`
- QC reports, gate decisions, and analysis reports: `{round_audit_log}`
- benchmark dossier and forbidden drift cases: `{benchmark_dossier}`

Instructions

Summarize why the round failed without proposing new items. Identify whether failures were caused by reliability problems, validity drift, weak separability gain, retained saturation, redundancy, scoring artifacts, or incoherent pattern hypotheses. Convert these findings into planning constraints for another same-depth round. Your reflection report should contain `failure_taxonomy`, `invalid_direction_patterns`, `promising_partial_signals`, `constraints_for_next_plan`, and `directions_to_avoid`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{reflection_agent_output_template}`

C.9. Answer Extraction Prompt

The answer extraction prompt standardizes model outputs before scoring. It is not allowed to solve the task or repair an incorrect answer; it only extracts the answer implied by the model output under the declared protocol.

Prompt for Answer Extraction

Role

You are the answer extraction component for benchmark scoring. Your job is to extract the candidate answer from a model response according to the declared benchmark protocol.

Context

- raw model response: `{model_output}`
- benchmark answer instruction: `{answer_instruction}`
- parser specification and allowed formats: `{parser_spec}`
- task metadata and expected answer type: `{task_metadata}`

Instructions

Extract only the answer explicitly provided by the model. Do not infer a corrected answer from reasoning if the final answer is missing or contradictory. If the response violates the declared answer format, return a parse failure with the shortest explanation. Preserve enough evidence for auditability by quoting the relevant answer span. Your extraction record should contain `parse_status` in `{parsed, parse_failure, ambiguous}`, `extracted_answer`, `answer_span`, `format_violation` if any, and `notes`.

Output

Please respond strictly in the following reference JSON format, and do not include any other content: `{answer_extraction_output_template}`

D. Labeling Instructions

D.1. Task Overview

Annotators review candidate benchmark items produced during the evolution process and determine whether each candidate should be accepted as a valid in-domain evolution of the seed benchmark. For each annotation task, annotators are given: (i) the seed benchmark specification and scoring protocol, (ii) the candidate item or task together with its proposed reference answer or solution, (iii) the adapted harness information when the candidate changes prompts, answer instructions, parsers, tests, or equivalence rules, and (iv) any already accepted candidates from the same evolution track.

The goal is to judge whether the candidate preserves the intended measurement target of the seed benchmark, whether the reference answer or solution is reliable, and whether the candidate exposes a meaningful performance pattern rather than an invalid hardening artifact. Annotators may inspect the candidate reasoning, execute or replay the relevant evaluation path when applicable, and record the shortest verification trace needed for later audit.

D.2. Materials Provided to Annotators

Annotators base their judgment only on the materials provided for the current task:

- **Seed Benchmark Specification:** The intended domain, difficulty range, task format, scoring rule, answer protocol, and known constraints of AIME 2025 or HumanEval.
- **Candidate Benchmark Item:** The evolved math problem or function-level programming task generated by the evolution route.
- **Reference Answer or Solution:** The proposed ground-

truth answer, reference reasoning, reference implementation, or unit-test oracle.

- **Harness Adaptation:** Any prompt, parser, scorer, equivalence rule, answer instruction, or hidden-test change introduced by the adapter agent.
- **Accepted Candidate Records:** The current list of already accepted candidates from the same track, used to identify duplicates or redundant variants.

If a candidate omits details that are necessary for verification, annotators may inspect nearby reasoning traces, recompute the answer, run the code tests, or refine the verification sequence. The final label must be grounded in behavior or correctness evidence that the annotator actually verified.

D.3. Definition of a Valid Candidate

A candidate should be labeled as **valid** only if all of the following conditions hold:

- **In-Domain:** The candidate remains within the seed benchmark's intended topic range, task format, and difficulty envelope.
- **Protocol-Consistent:** Any harness adaptation is explicitly declared and does not create a scoring artifact or hidden mismatch between instructions and evaluation.
- **Answer-Reliable:** The reference answer, reference reasoning, reference implementation, or unit-test oracle is correct under the stated task.
- **Diagnostic:** The candidate targets a coherent performance pattern rather than lowering scores through ambiguity, invalid assumptions, or unrelated difficulty escalation.

Do **not** mark a candidate as valid in the following situations:

- The task drifts into a different domain, such as olympiad-level mathematics for AIME or repository-level software engineering for HumanEval.
- The answer or reference solution is incorrect, underspecified, or cannot be independently verified.
- The observed score change is caused by ambiguous wording, parser failure, unsupported answer formats, or inconsistent scoring rules.
- The candidate is merely harder but does not correspond to a recurring model performance pattern.
- The evidence is too incomplete or ambiguous to support a confident decision.

D.4. Validity and Reliability Criteria

Validity judgement. The validity label captures whether a candidate should be admitted as an in-domain evolved benchmark item. A valid candidate preserves the original measurement scope and passes topic, range, difficulty, and protocol checks. An invalid candidate is rejected because it causes benchmark drift, depends on a scoring artifact, duplicates an already accepted candidate, or lacks sufficient evidence.

Reliability judgement. The reliability label captures whether the candidate's answer evidence is trustworthy. For math items, annotators verify the final answer and the reference reasoning. For code items, annotators inspect the reference implementation and test oracle, and execute or reason through the proposed tests when needed. A candidate with an unreliable answer or oracle cannot be accepted even if the task itself appears in-domain.

Duplicate or redundant candidates. If a candidate tests the same underlying pattern as an already accepted item under materially the same verification condition, annotators mark it as redundant and record the matched candidate ID. Superficial wording differences are not sufficient to make a candidate distinct.

Uncertain cases. If annotators cannot verify correctness or domain preservation after reviewing the provided materials, they mark the candidate as uncertain. Uncertain candidates are not used in the final release.

D.5. Required Output Format

Output Format

validity: {valid | invalid | redundant | uncertain}
answer_reliability: {reliable | unreliable | N/A}
matched_candidate_id: {candidate ID or empty}
minimal_verification_steps:
1. First verified action, computation, or test.
2. Second verified action, computation, or test.
3. Continue until the candidate judgement is auditable.

explanation: short but specific justification grounded in the seed specification, candidate behavior, and verification evidence.

D.6. Worked Example

The following example illustrates the annotation format for a HumanEval-style candidate.

Seed Track: HumanEval

Candidate Summary: The evolved task preserves the original function-level programming format but adds a hidden edge case involving empty input and repeated boundary

values.

Example Output

validity: valid
answer_reliability: reliable
matched_candidate_id: empty
minimal_verification_steps:
1. Read the seed HumanEval task format and confirm that the evolved item remains a single-function programming task.
2. Inspect the new edge-case specification and confirm that it is stated in the prompt rather than hidden only in the tests.
3. Execute the reference solution on the public examples and the proposed hidden edge cases.
4. Confirm that the candidate targets brittle input-assumption behavior rather than repository-level software engineering.
explanation: The candidate remains within the HumanEval-style function-level format, the added edge case is explicitly specified, and the reference solution passes the proposed oracle. The candidate is therefore a valid in-domain evolution with reliable answer evidence.

D.7. Important Considerations

- **Judge against the seed benchmark contract.** Do not accept a candidate merely because it lowers model scores; it must preserve the intended measurement target.
- **Separate validity from answer reliability.** A task can be in-domain but still rejected if its reference answer, reference implementation, or oracle is unreliable.
- **Record minimal verification steps.** The verification trace should be concise but sufficient for another expert to reproduce the judgement.
- **Treat protocol changes carefully.** Prompt, parser, scorer, equivalence-rule, or hidden-test changes must be explicit and justified by the candidate's measurement goal.
- **Do not guess.** If the evidence is insufficient, use uncertain rather than forcing a definitive label.

E. LLM Usage Statement

We utilized large language models solely for language polishing, including correcting grammatical errors and suggesting alternative vocabulary. These models did not contribute to the research design, analysis, or conclusions. The authors assume full responsibility for the integrity and content of this paper.