FrontierSmith: Synthesizing Open-Ended Coding Problems at Scale

Runyuan He[1,*] **Qiuyang Mang**[1,*,§] **Shang Zhou**[2] **Kaiyuan Liu**[3] **Hanchen Li**[1]
Huanzhi Mao[1] **Qizheng Zhang**[4] **Zerui Li** **Bo Peng**[5] **Lufeng Cheng** **Tianfu Fu**[6]
Yichuan Wang[1] **Wenhao Chai**[5] **Jingbo Shang**[2] **Alex Dimakis**[1,7]
Joseph E. Gonzalez[1] **Alvin Cheung**[1]

[1]UC Berkeley [2]UC San Diego [3]University of Washington [4]Stanford University
[5]Princeton University [6]Massachusetts Institute of Technology [7]Bespoke Labs

qmang@berkeley.edu, wenhao.chai@princeton.edu

Abstract

Many real-world coding challenges are open-ended and admit no known optimal solution. Yet, recent progress in LLM coding has focused on well-defined tasks such as feature implementation, bug fixing, and competitive programming. Open-ended coding remains a weak spot for LLMs, largely because open-ended training problems are scarce and expensive to construct. Our goal is to synthesize open-ended coding problems at scale to train stronger LLM coders. We introduce *FrontierSmith*, an automated system for iteratively evolving open-ended problems from existing closed-ended coding tasks. Starting from competitive programming problems, FrontierSmith generates candidate open-ended variants by changing the problems' goals, restricting outputs, and generalizing inputs. It then uses a quantitative idea divergence metric to select problems that elicit genuinely diverse approaches from different solvers. Agents then generate test cases and verifiers for the surviving candidates. On two open-ended coding benchmarks, training on our synthesized data yields substantial gains over the base models: Qwen3.5-9B improves by +8.82 score on FrontierCS and +306.36 (Elo-rating-based performance) on ALE-bench; Qwen3.5-27B improves by +12.12 and +309.12, respectively. The synthesized problems also make agents take more turns and use more tokens, similar to human-curated open-ended tasks, suggesting that closed-ended seeds can be a practical starting point for long-horizon coding data. Our sampled data and training scripts are publicly available at https://github.com/FrontierCS/FrontierSmith.

1 Introduction

LLMs now excel at well-defined coding tasks, reaching gold-medal performance in competitive programming [20] and solving over 80% of SWE-bench verified issues [24]. Yet these settings are mostly closed-ended: correctness is discrete and efficiently verifiable. Open-ended coding tasks, in contrast, lack tractable certificates of optimality at the target scale and score submissions by continuous quality. For example, in cloud cluster scheduling, many job-to-machine assignments are feasible, but they differ continuously in makespan, tail latency, energy use, and utilization; checking feasibility is easy, whereas certifying global optimality at production scale is intractable [34]. This difficulty is reflected in current open-ended benchmarks. On FrontierCS [41], human experts score 95.41 on algorithmic tasks, compared with 29.37 for Gemini 3.0 Pro [13]; on ALE-bench [21], frontier LLMs still trail average human participants in heuristic contests. This gap reflects a data

*Equal contribution, §Project lead

40th Conference on Neural Information Processing Systems (NeurIPS 2026).

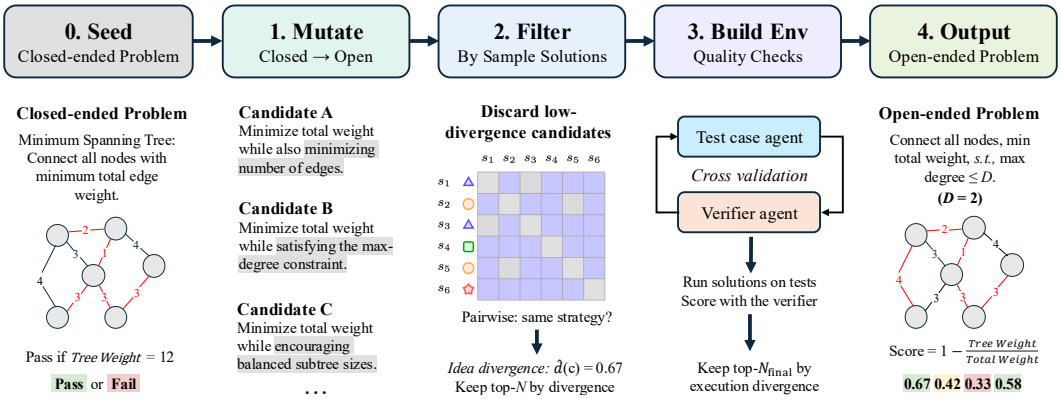


Figure 1: **FrontierSmith pipeline.** FrontierSmith converts closed-ended coding problems into open-ended ones through mutation, filtering, and automated environment construction. Candidate mugtations are ranked by idea divergence across sampled solutions, then validated by test-case and verifier agents. The resulting problems replace binary pass/fail feedback with continuous scores, enabling scalable training data synthesis for open-ended coding.

asymmetry: verified closed-ended tasks are abundant, while open-ended tasks remain scarce. We aim to automatically synthesize open-ended coding problems at scale to train stronger LLM coders.

For closed-ended tasks, the data problem is largely solved. Codeforces [10] and LeetCode [30] provide hundreds of thousands of closed-ended problems with verified solutions, enabling reinforcement learning from verifiable rewards to drive substantial gains. Open-ended coding has no equivalent. FrontierCS and ALE-bench, two of the largest open-ended coding benchmarks, contains only around 240 and 40 human-curated problems, respectively.

Constructing such problems manually is prohibitively expensive: each requires a carefully designed optimization objective, a verifier that produces continuous scores rather than binary judgments, and reliable test cases. Beyond engineering cost, expert judgment is needed to verify each problem is genuinely open-ended and not dominated by a single strategy (i.e., the core idea shared by a class of solutions, regardless of implementation).

Recent work has automated coding data synthesis, but is exclusively for closed-ended, binary-correctness settings. Given existing problem statements, several methods generate high-quality test cases at scale [16; 62; 38]. Other methods synthesize new problems across competitive programming [73], software engineering [67; 3; 56], terminal environments [12], and self-play [72; 23]. None of them transfers to open-ended problems, which need novel formulations without a known optimum, and a way to reliably score solution quality on a continuous scale.

We build *FrontierSmith*, an automated system that continuously evolves open-ended coding problems from closed tasks at scale. As shown in Figure 1, starting from closed-end competitive programming problems, which provide a large and diverse seed corpus, FrontierSmith's pipeline mutates them into open-ended variants along three axes: changing goals, restricting outputs, and generalizing inputs. Each mutation turns a problem with a known efficient solution into one where exact solutions are intractable, forcing solvers to adopt diverse strategies. We then quantify the open-endedness of the remaining candidates with a novel *idea divergence* metric, which estimates the probability that two independent solvers use different core algorithms. This estimation proceeds in two stages: first via LLM-based comparison of solution strategies, then via test-score similarity once test cases and verifiers are constructed for each candidate. These are built by separate agents systems that cross-validate each other to ensure correctness. Validated problems then join the seed pool, so each subsequent round draws from an increasingly diverse starting set.

We train Qwen3.5-9B and Qwen3.5-27B [58] on 200 synthesized problems with GRPO [53]. The 9B model gains +8.82 score on FrontierCS and +306.36 (Elo-rating-based performance) on ALE-bench; the 27B model gains +12.12 and +309.12, respectively. Against two controls, closed-ended HardTests [16] and random rewards [52], our synthetic problems lead by +5.24 / +236.40 and +7.58 /

+256.76 on FrontierCS / ALE-bench, confirming that open-ended formulations and genuine reward signals are both necessary. Ablations show the filter is critical: removing it drops performance by 2.05 points on FrontierCS. Beyond training, our idea divergence metric cleanly separates real open-ended problems from competitive programming problems, validating it as a problem-quality classifier. Finally, open-ended problems have recently shown to elicit distinctive long-horizon behavior in code agents, including more turns, tool calls, and thinking tokens [6; 36; 50; 37]; our synthetic problems match these patterns, suggesting they capture the same structure as human-curated ones.

In sum, our key findings are: (1) closed-ended problems can serve as seeds for open-ended synthesis via mutations that remove known optima; (2) idea divergence is an effective and tractable signal for selecting high-quality synthetic problems; and (3) FrontierSmith-generated problems yield training gains comparable to human-curated data, and exhibit long-horizon behavior when processed by agents.

2 Related Work

Coding data synthesis for closed tasks. Recent work synthesizes coding problems and test infrastructure under binary correctness. AutoCode [73], SWE-smith [67], BugPilot [56], Endless Terminals [12], GASP [23], and SGS [4] synthesize problems; HardTests [16], CodeContests+ [62], rStar-Coder [38], SWE-rebench [3], and R2E-Gym [22] build tests and verifiers; AgentCoder [19] and CURE [60] cross-validate code and tests across multiple agents. None produces open-ended problems, the regime FrontierSmith targets.

Open-ended evaluation and solution diversity. Open-ended coding benchmarks score solutions on a continuous quality scale: FrontierCS [41], ALE-bench [21], HeuriGym [9], KernelBench [48], RE-Bench [64], and MLE-bench [8]. NP-Engine [33] hand-crafts instance generators and rule-based verifiers for ten classical NP-hard tasks, training via RLVR with approximation-ratio rewards; its catalog is fixed and only difficulty varies, which caps the diversity of generated problems. Recent work [45; 61; 40] produces frontier results in open-ended mathematics and algorithm design. Our idea divergence metric draws on quality-diversity and novelty search [31; 43; 59; 5; 11; 49], with code-specific analogs in Lee et al. [29] and Ju et al. [25]. Unlike these works, which measure diversity of one model's outputs, we use inter-solver divergence as a problem-quality filter that admits only formulations capable of eliciting different core algorithms, a necessary but not sufficient signal for open-endedness.

Mutation-based synthesis and iterative self-play. Evolving prompts or programs by mutation is established: WizardLM [66], WizardCoder [39], EvoEval [65], and Auto Evol-Instruct [71] mutate seed prompts; FunSearch [51], AlphaEvolve [45], ELM [32], and EoH [35] mutate programs inside evolutionary loops. Iterative methods and infrastructure support continual improvement: STaR [70], ReST-EM [55], V-STaR [17], Self-Rewarding LM [69], R-Zero [18], EVA [68], Absolute Zero [72], and Reef [7]. FrontierSmith differs: we mutate problem formulations rather than solutions or instructions, and admit them by inter-solver divergence rather than learning gain, self-consistency, or solver pass rate. These signals reward solver-side progress or confidence rather than the open-endedness of the problem itself.

3 Method

3.1 Mutation Problem Formulation

Existing work on synthesizing closed-ended coding tasks focuses on constructing test environments rather than problem formulations, because closed-ended formulations can be drawn from existing repositories [73; 16; 62] and domain-specific datasets [67; 56]. Open-ended problems have no such source; the formulation itself is the central challenge.

Our key insight is that closed-ended problems can serve as seeds for synthesizing open-ended ones. Targeted mutations to their formulations can remove known optima while preserving a meaningful way to evaluate submission quality.

Algorithm 1: FrontierSmith: Open-Ended Problem Synthesis

Input: Seed pool $\mathcal{S}$, CP corpus $\mathcal{D}$, LLM $\mathcal{M}$, sample size B, solutions per candidate n, budgets N_{div}, N_{final}
Output: Validated open-ended problem set $\mathcal{P}$

$\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{D}$; ▷ add CP corpus to seed pool
$\mathcal{B} \leftarrow$ sample B problems from $\mathcal{S}$;
$\mathcal{C} \leftarrow \texttt{Mutate}(\mathcal{B}, \mathcal{M})$; ▷ mutate goals, outputs, inputs
$\mathcal{C} \leftarrow \texttt{CoarseFilter}(\mathcal{C}, \mathcal{M})$; ▷ remove non-open-ended
foreach *candidate* $c \in \mathcal{C}$ **do**
  $\{s_1, \ldots, s_n\} \leftarrow$ draw n solutions for c from $\mathcal{M}$;
  $\hat{d}(c) \leftarrow \frac{1}{\binom{n}{2}} \sum_{i<j} \texttt{LLM-as-a-Judge}(s_i, s_j)$; ▷ idea divergence
end
$\mathcal{C} \leftarrow \text{top}_{N_{\text{div}}}\big(\mathcal{C},\ \hat{d}(\cdot)\big)$; ▷ keep highest divergence
foreach *candidate* $c \in \mathcal{C}$ **do**
  $(\mathcal{T}_c, \mathcal{V}_c) \leftarrow \texttt{BuildAndValidate}(c, \mathcal{M})$; ▷ test cases + verifier with cross-validation
  Update $\hat{d}(c)$ using score vectors $\mathbf{q}_i = (\mathcal{V}_c(s_i, t_1), \ldots, \mathcal{V}_c(s_i, t_m))$;
end
$\mathcal{P} \leftarrow \text{top}_{N_{\text{final}}}\big(\mathcal{C},\ \hat{d}(\cdot)\big)$; ▷ re-rank with verified divergence
$\mathcal{S} \leftarrow \mathcal{S} \cup \mathcal{P}$; ▷ expand seed pool for next round
return $\mathcal{P}$

Concretely, we represent a problem formulation as a tuple $(\mathcal{O}, \mathcal{C}_I, \mathcal{C}_O)$ where $\mathcal{O}$ is the computational goal, $\mathcal{C}_I$ the admissible problem instances, and $\mathcal{C}_O$ the constraints on valid program outputs. In this representation, $\mathcal{O}$ can take various forms: a required output, a decision criterion, a property to satisfy, or a quantity to optimize. We define three mutation types that transform a closed-ended formulation into an open-ended one:

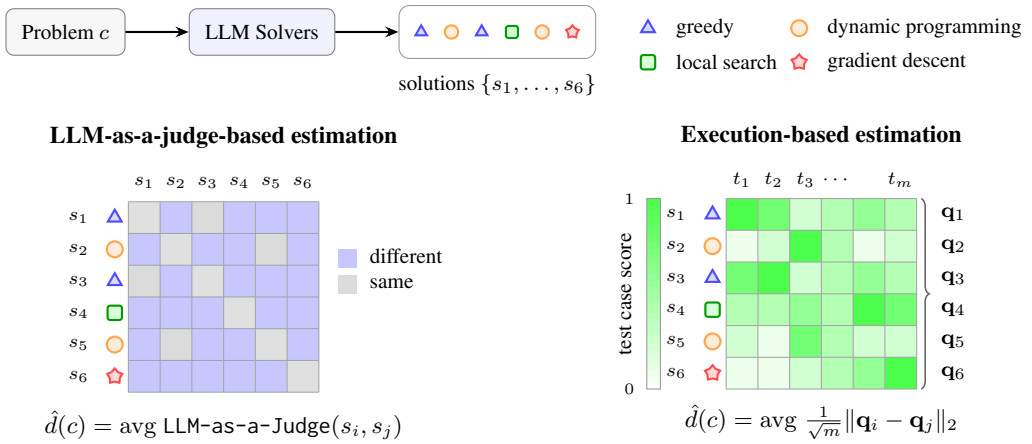
1. **Changing goals** ($\mathcal{O} \to \mathcal{O}'$): replace a decision or exact-answer goal with an optimization-oriented goal that admits graded performance. For example, 2-SAT [2] decides whether a Boolean formula with two-literal clauses is satisfiable; a mutation of this goal produces Min-True 2-SAT [14], which keeps the same input and output but asks for a satisfying assignment that minimizes the number of true variables.
2. **Restricting outputs** ($\mathcal{C}_O \to \mathcal{C}'_O$): add or tighten constraints on valid outputs while keeping the underlying goal fixed. For example, the minimum spanning tree problem [28] on a weighted graph asks for a tree connecting all vertices with minimum total edge weight, which admits a greedy solution. Adding per-vertex degree bounds yields the NP-hard degree-constrained spanning tree [44]. At scale, exact solutions become infeasible, and different approximation strategies yield solutions of varying quality.
3. **Generalizing inputs** ($\mathcal{C}_I \to \mathcal{C}'_I$): relax structural assumptions on the input domain while keeping the goal and output constraints fixed. For example, the maximum independent set asks for the largest vertex set with no two vertices sharing an edge. On bipartite graphs, it is polynomial via Kőnig's theorem [27], but on arbitrary graphs it becomes one of Karp's original NP-complete problems [26], similarly making exact solutions infeasible at scale.

In all three mutation types, the resulting problem has no efficient method to certify its optimum at scale and admits a continuous quality measure: both conditions for open-endedness. We implement each mutation by prompting an LLM $\mathcal{M}$ with the seed problem and the desired mutation type; multiple types can apply simultaneously to a single candidate. Given only the problem statement, $\mathcal{M}$ first extracts the original formulation $(\mathcal{O}, \mathcal{C}_I, \mathcal{C}_O)$ and then produces an open-ended candidate $(\mathcal{O}', \mathcal{C}'_I, \mathcal{C}'_O)$.

3.2 Problem Filtering

Mutation produces a broad but noisy pool; we apply two filters in sequence to retain only high-quality open-ended candidates.

Coarse LLM-as-a-judge filter. The first filter is a coarse LLM-as-a-judge check. We prompt $\mathcal{M}$ to check three conditions: the problem defines an optimization objective with no known optimum;



LLM-based estimation. An LLM judge labels each solution pair as same- or different-strategy. $\hat{d}(c)$ is the fraction of pairs judged different. Blue cells: different-strategy; gray cells: same-strategy.

Execution-grounded estimation. Each solution is run on the test cases $t_1, \ldots, t_m$ and scored by the verifier, yielding a score vector $\mathbf{q}_i$. $\hat{d}(c)$ is the average pairwise distance between score vectors.

Figure 2: Estimating idea divergence $\hat{d}(c)$, the probability that two independently sampled LLM-generated solutions to candidate problem c use different algorithmic strategies. We draw 6 solutions $s_1, \ldots, s_6$ from an LLM solver and estimate $\hat{d}(c)$ in two complementary ways. The shape attached to each s_i denotes its underlying algorithmic strategy.

multiple distinct strategies are plausible; and a scoring function can meaningfully rank submissions. Candidates failing any condition are discarded.

Idea divergence filter. The coarse filter removes closed-ended candidates but does not measure solution diversity. If one strategy dominates, the problem effectively degenerates to a closed-ended task. Well-designed open-ended settings exhibit genuine solution diversity. The AtCoder Heuristic Contest [21] and the database join-ordering problem [57] illustrate this: top performers explore fundamentally different ideas rather than refine one dominant approach. This diversity also strengthens RL training: under GRPO [53], varied strategies that yield meaningfully different rewards produce a stronger gradient signal than samples following the same heuristic [63]. We therefore introduce an *idea divergence* filter that directly quantifies this diversity.

Definition. For a candidate problem $c = (\mathcal{O}, \mathcal{C}_I, \mathcal{C}_O)$, we define idea divergence as the probability that two independently generated solutions use different algorithmic strategies (Figure 2):

$$d(c) \coloneqq \mathbb{P}_{s_i, s_j \sim \text{Solver}(c)}\big[\text{strategy}(s_i) \neq \text{strategy}(s_j)\big]. \tag{1}$$

$\text{Solver}(c)$ is the distribution over LLM-generated solutions for c, and $\text{strategy}(\cdot)$ maps each solution to its core algorithmic idea. Computing $d(c)$ exactly is intractable; we provide two complementary estimates below.

LLM-based estimate. We draw n independent solutions $s_1, \ldots, s_n \sim \text{Solver}(c)$. An LLM-as-a-judge labels each pair (s_i, s_j) as same- or different-strategy:

$$\hat{d}(c) \coloneqq \frac{1}{\binom{n}{2}} \sum_{i<j} \texttt{LLM-as-a-Judge}\big(s_i, s_j\big). \tag{2}$$

where each judge call returns 1 if the strategies differ and 0 otherwise. A naive implementation requires $O(n^2)$ judge calls. We batch the calls instead, scoring all pairs in a small group per query and averaging across groups.

Execution-grounded estimate. After testing infrastructure (§3.3), we complement with an execution-based estimate. Given test cases $\mathcal{T}_c = \{t_1, \ldots, t_m\}$ and verifier $\mathcal{V}_c$ for c, the score

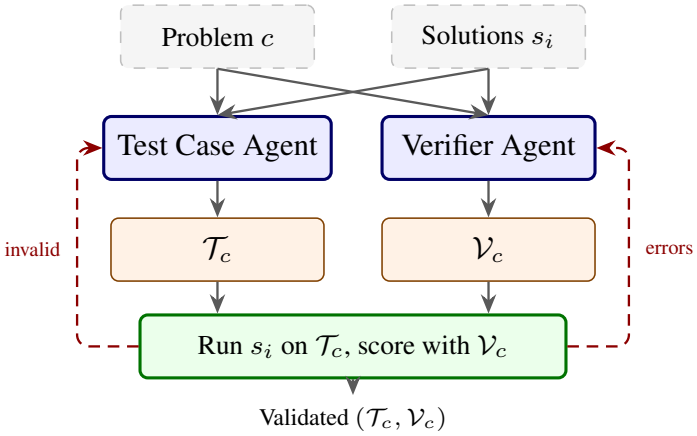


Figure 3: Test case and verifier synthesis. A test case agent produces inputs $\mathcal{T}_c$; a verifier agent produces a scoring program $\mathcal{V}_c$. Sampled solutions are run on both; each agent uses the other's output to validate its own (dashed), iterating until consistent.

vector $\mathbf{q}_i = (\mathcal{V}_c(s_i, t_1), \ldots, \mathcal{V}_c(s_i, t_m))$ records s_i's per-test-case performance, and we estimate divergence as

$$\hat{d}(c) \coloneqq \frac{1}{\binom{n}{2}} \sum_{i<j} \frac{1}{\sqrt{m}} \|\mathbf{q}_i - \mathbf{q}_j\|_2 \,. \tag{3}$$

Selection. The two estimates above are complementary. The LLM-based estimate captures semantic differences in algorithmic strategy, e.g., greedy vs. dynamic programming. The execution-based estimate captures behavioral differences across test cases, e.g., two similar solutions that trade speed against accuracy. Together they form a two-stage funnel (Algorithm 1): the LLM-based estimate is applied first since it requires no test infrastructure, retaining the top N_{div} candidates. After testing infrastructure (§3.3), the execution-based estimate refines the ranking, and the top N_{final} are selected as the final problem set $\mathcal{P}$.

3.3 Testing Infrastructure

Test case generation. For each surviving candidate c, a test case agent generates a set of inputs $\mathcal{T}_c$ (Figure 3). Closed-ended test cases target corner cases and boundary conditions; open-ended test cases must stress-test different algorithmic strategies. Following prior work on test synthesis [73; 62], we prompt the agent to write test-case generator programs that produce inputs of varying size and structure, such as sparse versus dense graphs or uniform versus skewed distributions. The test-case agent also receives the solutions sampled by the divergence estimate of §3.2, allowing it to craft adversarial inputs that expose where specific strategies break down.

Verifier generation. Each candidate's objective $\mathcal{O}$ orders solutions but yields no bounded score. A separate verifier agent translates $\mathcal{O}$ into a scoring program $\mathcal{V}_c$ that returns a normalized score in $[0, 1]$. One common approach is to normalize against a baseline solution s^*: $\mathcal{V}_c$ returns 0 when s fails to improve on s^* and a value in $[0, 1)$ otherwise, scaling with the size of the improvement. The baseline need not be strong; even a greedy heuristic or random valid solution suffices. Note that we can ensure $\mathcal{O}(\cdot, t) > 0$ by adding a constant offset. We assume $\mathcal{O}(\cdot, t) > 0$. Letting $\sigma_{\mathcal{O}} = +1$ for maximization objectives and -1 for minimization,

$$\mathcal{V}_c(s, t) = \max\left(0,\ \sigma_{\mathcal{O}} \cdot \frac{\mathcal{O}(s, t) - \mathcal{O}(s^*, t)}{\max\{\mathcal{O}(s, t),\ \mathcal{O}(s^*, t)\}}\right). \tag{4}$$

Solutions that crash, time out, or produce unparseable output receive a score of 0.

Cross-validation protocol. The test case agent and the verifier agent use each other's output for self-validation. The test case agent runs the sampled solutions through the verifier: if a solution crashes or produces unparseable output on a test input, that input is invalid, and the test-case agent revises it. The verifier agent scores the sampled solutions from §3.2 on the generated test cases: if

scores collapse to a narrow band or contradict the relative quality of solutions, the scoring program is flawed, and the agent revises it. Each agent iterates until the other's output no longer exposes errors in its own; candidates that fail to converge within a fixed number of rounds are discarded. In our experiments, 10% of candidates that enter this stage produce a validated $(\mathcal{T}_c, \mathcal{V}_c)$ pair, which then feeds the execution-based idea-divergence re-ranking of §3.2.

4 Experiments

4.1 Experimental Setup

Benchmarks. We evaluate on two open-ended coding benchmarks. FrontierCS [41] contains 240 open-ended problems across two tracks. We use the 172 algorithmic problems, which require only local computation; the remaining 68 problems require cloud infrastructure (e.g., GPUs, external services) and are excluded. Submissions are scored on a continuous 0–100 scale. ALE-bench [21] draws tasks from AtCoder Heuristic Contests and evaluates submissions using performance-based ratings. We use the ALE-bench-lite subset (10 tasks) for evaluation.

Baselines. We compare five configurations: (1) Base: the pretrained model without RL; (2) FrontierCS: RL on 172 human-curated FrontierCS algorithmic problems; (3) ALE-bench: RL on 40 ALE-bench tasks, with rewards computed from each problem's public test set. (4) HardTests: RL on 200 problems randomly sampled from the 47,136 closed-ended competitive programming problems from HardTests [16], trained with binary rewards (1 if all test cases pass, 0 otherwise), following the standard competitive programming setup [10; 20]; (5) Random Reward: RL on 172 FrontierCS algorithmic problems with the raw rewards drawn from $\mathcal{U}[0, 100]$. Following Shao et al. [52], this is a spurious reward control: it tests whether gains require task-specific reward signals or arise from RL dynamics under uninformative rewards.

Problem synthesis. We run Algorithm 1 for 4 iterations. Each iteration samples $B = 1{,}000$ seed problems from the pool (initialized with HardTests [16]) with $N_{\text{div}} = 100$ and $N_{\text{final}} = 50$, yielding 50 synthetic problems. We use GPT-5.4 Thinking [46] with medium thinking effort for problem formulation mutation (§3.1), the coarse LLM-as-a-judge filter, the LLM-based divergence estimate (§3.2), and Claude Sonnet 4.6 [1] with default thinking effort for sampling solutions ($n = 10$ per candidate), test case generation, and verifier generation (§3.3). The generated problems follow the FrontierCS [41] format, and we reuse its evaluation sandbox to execute code and run verifiers.

Training configurations. We use veRL [54] with GRPO, learning rate is set to 10^{-6} for Qwen3.5-9B and 5×10^{-7} for Qwen3.5-27B, rollout batch size 8, and group size 8. We train for 100 steps and evaluate every 10 steps. Qwen3.5-9B trains on 8 A100 GPUs for 1.5 days with maximum response length 16,000 tokens; Qwen3.5-27B trains on 32 H200 GPUs for 1.5 days with maximum response length 32,000 tokens. For the 27B model, we only report Base, FrontierCS, HardTests, and FrontierSmith due to our computing budget.

4.2 Main Results

Table 1 reports the main results. FrontierSmith achieves competitive or superior performance to human-curated training data on both benchmarks, and the gains hold across model sizes (9B and 27B), suggesting that our synthesis pipeline scales with model capacity.

FrontierSmith vs. human-curated open-ended data. On Qwen3.5-9B, FrontierSmith achieves 10.62 Avg@5 on FrontierCS, close to 11.17 from human-curated FrontierCS problems, a gap of only 0.55 points. On ALE-bench, FrontierSmith achieves the highest Best@5 (782.30) across all configurations, and its Avg@5 (633.58) surpasses the FrontierCS-trained baseline (558.49), trailing only the ALE-bench-trained model (657.40) which benefits from in-domain data. On Qwen3.5-27B, FrontierSmith outperforms the human-curated baseline on both benchmarks: FrontierCS (19.82 vs. 13.98) and ALE-bench (661.64 vs. 543.80), improving from a base of 7.70 and 352.52. These results suggest that automated synthesis can largely substitute for expensive human curation, with strong cross-benchmark generalization.

	Qwen3.5-9B				Qwen3.5-27B			
	FrontierCS		ALE-bench		FrontierCS		ALE-bench	
Training Data	Avg@5	Best@5	Avg@5	Best@5	Avg@5	Best@5	Avg@5	Best@5
Base	1.80	5.00	327.22	327.22	7.70	13.91	352.52	498.00
FrontierCS (172)	**11.17**	**16.29**	558.49	762.28	13.98	21.92	543.80	843.80
ALE-bench (40)	10.64	15.70	**657.40**	750.15	-	-	-	-
FrontierSmith (200)	10.62	15.73	633.58	**782.30**	**19.82**	**29.38**	**661.64**	**938.10**
HardTests (200)	5.38	11.19	397.18	512.83	11.20	18.37	529.12	922.50
Random Reward (172)	3.04	6.93	376.82	466.73	-	-	-	-

Table 1: Performance on FrontierCS and ALE-bench. We train for 100 steps and evaluate every 10 steps; reported numbers are the best checkpoint for each metric. Avg@5 and Best@5: average and maximum score over 5 samples per problem. Parentheses: number of training problems. **Bold**: best; underline: second best. Red cells: training data overlaps with the evaluation benchmark.

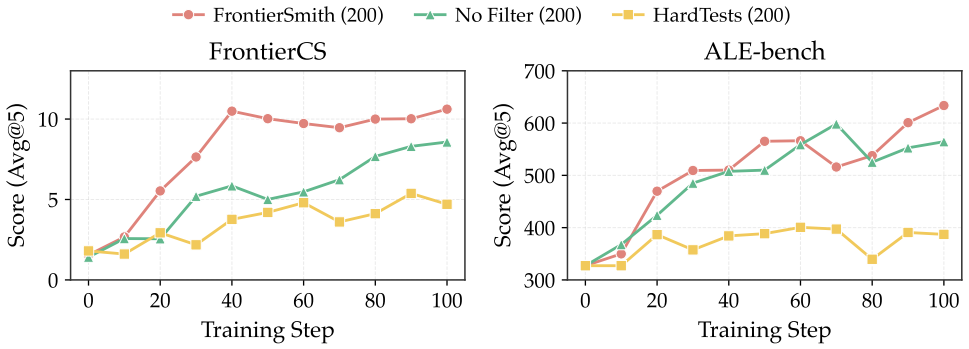


Figure 4: Test performance over training steps on FrontierCS and ALE-bench (Qwen3.5-9B, 200 problems each). FrontierSmith and HardTests are the same runs as Table 1; "No Filter" skips both filters. FrontierSmith consistently outperforms the no-filter ablation and the HardTests baseline.

FrontierSmith vs. closed-ended data. HardTests problems serve as the seed corpus for FrontierSmith's mutations, yet training directly on them yields only 5.38 on FrontierCS and 397.18 on ALE-bench for the 9B model, well below FrontierSmith's 10.62 and 633.58. The same pattern holds for the 27B model (11.20 vs. 19.82 on FrontierCS). This demonstrates that mutation transforms closed-ended problems into more effective open-ended training data.

Random reward control. The random reward baseline scores 3.04 on FrontierCS and 376.82 on ALE-bench, which is similar to the untrained base (1.80 and 327.22). This rules out problem-format exposure as the source of gains, consistent with Shao et al. [52].

4.3 Filter Analysis

Figure 4 shows test performance over training steps for three 200-problem configurations: FrontierSmith and HardTests are the same runs as in Table 1; No Filter skips both filters and directly builds test infrastructure for 200 mutated problems. On FrontierCS, FrontierSmith reaches 10.62 Avg@5, compared with 8.57 for the no-filter variant. The same trend holds on ALE-bench, where FrontierSmith achieves 633.6 versus 564.4. These results suggest that idea-divergence filtering improves both in-domain performance and cross-benchmark generalization. Both filtered and unfiltered synthetic data outperform the closed-ended HardTests baseline (5.38 and 397.2). We next validate the two filters individually.

Coarse filter validation. To validate the coarse LLM-as-a-judge filter (§3.2), we apply it to 100 closed-ended competitive programming problems from HardTests [16] using GPT-5.4 Thinking [46]

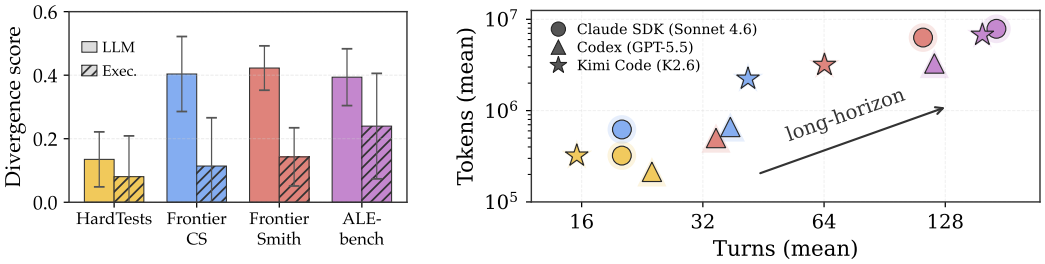


Figure 5: Comparison of four problem sources. Color encodes the problem source in both panels. **Left:** Idea divergence cleanly separates open-ended from closed-ended problems. **Right:** Points show source-agent geometric means of turns and tokens; FrontierSmith and human-curated open-ended problems elicit increasing horizon agent behavior.

as the judge. Since all problems in this set are closed-ended, any retained candidate is a false positive. The filter rejects 91 and retains 9, yielding a false-positive rate of 9%. Conversely, applying the filter to 100 FrontierCS problems rejects 19, a false-negative rate of 19%. This is acceptable because the filter is designed to preserve high-quality open-ended problems rather than maximizing recall.

Idea divergence as a classifier. To test whether idea divergence separates open-ended from closed-ended problems, we sample 10 tasks from each of four sources: FrontierCS (human-curated), FrontierSmith (synthetic), HardTests (closed-ended, sampled from the hardest difficulty tier in human competitive programming contests), and ALE-bench. For each task, we sample n=10 solutions using Claude Sonnet 4.6 and compute the two divergence estimates (§3.2).

The left panel of Figure 5 shows the results. Using the LLM-based estimate, the three open-ended sources (FrontierCS, FrontierSmith, and ALE-bench) all score around 0.4, roughly 3× higher than HardTests (0.14). FrontierSmith scores slightly higher than the human-curated FrontierCS (0.42 vs. 0.40), suggesting that our filtering pipeline selects problems with genuine solution diversity comparable to the human-curated ones. The execution-grounded estimate shows a similar trend: HardTests remains lowest (0.08), while FrontierCS, FrontierSmith, and ALE-bench score 0.11, 0.14, and 0.24, respectively. Although the margin between HardTests and the open-ended sources is smaller, the separation remains clear.

4.4 Long-Horizon Code Agent Behavior

Open-ended problems often lead to extended interactions: agents benefit from iterating on solutions, running tests, and refining strategies over many turns. Using the same 40 tasks from §4.3, we run three code agents via the Harbor evaluation framework [15]: Claude SDK with Sonnet 4.6 [1], Codex with GPT-5.5 [47], and Kimi Code with K2.6 [42]. The right panel of Figure 5 plots geometric-mean turns and tokens for each source-agent pair. ALE-bench drives all three agents into the long-horizon regime, where every pair exceeds 100 turns and 3×10^6 tokens. FrontierSmith exhibits the same long-horizon behavior, with Claude SDK reaching 113 turns and 6.3×10^6 tokens, on par with ALE-bench. HardTests and FrontierCS stay in the short-horizon regime.

5 Discussion

Synthesis cost analysis. FrontierSmith synthesizes each open-ended problem for approximately $16.1 in LLM API costs. Filtering dominates at 45% of the per-problem cost, driven by sampling n=10 solutions per candidate and pairwise LLM-as-a-judge calls; mutation accounts for 9% and test infrastructure generation for 46%. By model, Claude Sonnet 4.6 accounts for 87% (primarily solution sampling) and GPT-5.4 Thinking for 13% (mutation and judging). This is orders of magnitude cheaper than manual curation by domain experts.

Human expert evaluation. To assess whether synthesized problems match human-curated quality, we conduct a blind ranking study. We sample 10 groups, each containing one problem from FrontierCS (human-curated), FrontierSmith (synthetic), and HardTests (closed-ended), presented in random order.

Source	Mean Rank	Std
FrontierSmith	**1.60**	0.85
FrontierCS	1.72	0.60
HardTests	2.44	0.85

Table 2: Human expert ranking of open-endedness (lower is better).

Experts rank the three problems in each group by open-ended quality using the rubric in Appendix A; ties are allowed. Each group is evaluated by 2–3 experts, yielding 25 rankings per source. In this evaluation, experts were asked to inspect each problem statement and reason about several plausible solutions, but were not required to implement them. Table 2 reports the results. FrontierSmith problems receive the best mean rank (1.60), slightly ahead of human-curated FrontierCS (1.72) and well ahead of closed-ended HardTests (2.44). This confirms that our synthesis pipeline produces problems perceived by experts as comparable in quality to human-curated open-ended problems.

Limitations. Our current pipeline operates entirely within self-contained algorithmic environments: each problem takes a text input and produces a program output, with no external dependencies. This excludes repo-level open-ended tasks that require complex environment setup, such as system optimization with cloud infrastructure, GPU kernel tuning, or multi-file software engineering. Extending FrontierSmith to these settings would require generating not only problem formulations and verifiers but also reproducible execution environments. Additionally, due to compute budget constraints, our RL training is limited to 100 steps of single-turn GRPO. We do not explore agentic RL and leave this as future work, where the policy interacts with the environment over multiple turns, which is a natural next step given the long-horizon nature of the problems we synthesize.

6 Conclusion

We introduced FrontierSmith, an automated pipeline that transforms closed-ended coding problems into open-ended ones using targeted mutations and idea-divergence filtering. Training Qwen3.5-9B and 27B on FrontierSmith-generated problems achieves performance competitive with training on human-curated open-ended data, while substantially outperforming training on closed-ended problems and generalizing strongly across benchmarks. Our analysis further shows that idea divergence serves as a reliable signal of open-endedness, and that synthesized problems elicit long-horizon agent behavior comparable to human-curated tasks, with agents spending substantially more turns and tokens than on closed-ended problems. Together, these findings show that open-ended coding data can be synthesized at scale without expensive expert curation. FrontierSmith therefore provides a scalable source of training data for RL on long-horizon, open-ended coding tasks.

Responsible Use Statement

This work synthesizes open-ended coding problems for model training. Although this can improve coding agents, stronger agents may also enable insecure or harmful software. We recommend filtering tasks, sandboxing execution, auditing verifiers and data, and requiring human review for sensitive deployments. Our experiments use self-contained benchmarks without access to production systems or sensitive data.

Acknowledgments and Disclosure of Funding

This research has been supported by NSF awards IIS-1955488, IIS-2027575, DOE awards DE-SC0016260, AC02-05CH11231, a Bakar Fellowship, and DARPA Agreement No. HR00112590131. We thank Youliang Yuan, Ruyi Ji, Zhifei Li, the Harbor team, the Modal team, and the Laude Institute. This work received no third-party funding. The authors declare no competing interests.

References

[1] Anthropic. Introducing claude sonnet 4.6. https://www.anthropic.com/news/claude-sonnet-4-6, 2026. Accessed: 2026-05-03.

[2] Bengt Aspvall, Michael F Plass, and Robert Endre Tarjan. A linear-time algorithm for testing the truth of certain quantified boolean formulas. *Information processing letters*, 8(3):121–123, 1979.

[3] Ibragim Badertdinov, Alexander Golubev, Maksim Nekrashevich, Anton Shevtsov, Simon Karasik, Andrei Andriushchenko, Maria Trofimova, Daria Litvintseva, and Boris Yangel. SWE-rebench: An automated pipeline for task collection and decontaminated evaluation of software engineering agents. *arXiv preprint arXiv:2505.20411*, 2025.

[4] Luke Bailey, Kaiyue Wen, Kefan Dong, Tatsunori Hashimoto, and Tengyu Ma. Scaling self-play with self-guidance. *arXiv preprint arXiv:2604.20209*, 2026.

[5] Herbie Bradley, Andrew Dai, Hannah Teufel, Jenny Zhang, Koen Oostermeijer, Marco Bellagente, Jeff Clune, Kenneth Stanley, Grégory Schott, and Joel Lehman. Quality-diversity through AI feedback. In *International Conference on Learning Representations*, 2024.

[6] Mert Cemri, Shubham Agrawal, Akshat Gupta, Shu Liu, Audrey Cheng, Qiuyang Mang, Ashwin Naren, Lutfi Eren Erdogan, Koushik Sen, Matei Zaharia, et al. Adaevolve: Adaptive llm driven zeroth-order optimization. *arXiv preprint arXiv:2602.20133*, 2026.

[7] Wenhao Chai, Shuangrui Ding, Hao He, Haoze He, Chonghe Jiang, Nan Jiang, Xuan Jiang, Xiaochen Li, Paul Liang, Bo Liu, Boyuan Long, Qiuyang Mang, Zhenting Qi, Ao Qu, Mingruo Qu, Zhaokai Wang, Xuezhi Yan, Hanfei Yu, Haofei Yu, Simon Yu, Han Zheng, Kaichen Zhou, Zijian Zhou, Jiacheng Zhu, and Dingyi Zhuang. Reef: Continual learning infrastructure for self-improving agents. https://github.com/Human-Agent-Society/reef, 2026. Accessed: 2026-09-07.

[8] Jun Shern Chan et al. MLE-bench: Evaluating machine learning agents on machine learning engineering. *arXiv preprint arXiv:2410.07095*, 2024.

[9] Hongzheng Chen et al. HeuriGym: An agentic benchmark for LLM-crafted heuristics in combinatorial optimization. *arXiv preprint arXiv:2506.07972*, 2025.

[10] Codeforces. Codeforces: Programming competitions and contests. https://codeforces.com, 2026.

[11] Maxence Faldor, Jenny Zhang, Antoine Cully, and Jeff Clune. OMNI-EPIC: Open-endedness via models of human notions of interestingness with environments programmed in code. In *International Conference on Learning Representations*, 2025.

[12] Kanishk Gandhi, Shivam Garg, Noah D. Goodman, and Dimitris Papailiopoulos. Endless terminals: Scaling RL environments for terminal agents. *arXiv preprint arXiv:2601.16443*, 2026.

[13] Google DeepMind. Gemini 3: Introducing the latest gemini ai model from google. https://blog.google/products/gemini/gemini-3/, 2025. Accessed: 2026-05-05.

[14] Dan Gusfield and Leonard Pitt. A bounded approximation for the minimum cost 2-sat problem. *Algorithmica*, 8(1):103–117, 1992.

[15] Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. https://github.com/harbor-framework/harbor, January 2026.

[16] Zhongmou He, Yee Man Choi, Kexun Zhang, Jiabao Ji, Junting Zhou, Dejia Xu, Ivan Bercovich, Aidan Zhang, and Lei Li. HardTests: Synthesizing high-quality test cases for LLM coding. *arXiv preprint arXiv:2505.24098*, 2025.

[17] Arian Hosseini, Xingdi Yuan, Nikolay Malkin, Aaron Courville, Alessandro Sordoni, and Rishabh Agarwal. V-STaR: Training verifiers for self-taught reasoners. *arXiv preprint arXiv:2402.06457*, 2024.

[18] Chengsong Huang et al. R-Zero: Self-evolving reasoning LLM from zero data. In *International Conference on Learning Representations*, 2026.

[19] Dong Huang, Qingwen Bu, Jie M. Zhang, Michael Luck, and Heming Cui. AgentCoder: Multi-agent-based code generation with iterative testing and optimisation. *arXiv preprint arXiv:2312.13010*, 2023.

[20] ICPC Foundation. International Collegiate Programming Contest. `https://icpc.global`, 2026.

[21] Yuki Imajuku, Kohki Horie, Yoichi Iwata, Kensho Aoki, Naohiro Takahashi, and Takuya Akiba. ALE-Bench: A benchmark for long-horizon objective-driven algorithm engineering. *arXiv preprint arXiv:2506.09050*, 2025.

[22] Naman Jain, Jaskirat Singh, Manish Shetty, Tianjun Zheng, Koushik Sen, and Ion Stoica. R2E-Gym: Procedural environments and hybrid verifiers for scaling open-weights SWE agents. *arXiv preprint arXiv:2504.07164*, 2025.

[23] Swadesh Jana, Cansu Sancaktar, Tomáš Daniš, Georg Martius, Antonio Orvieto, and Pavel Kolev. GASP: Guided asymmetric self-play for coding LLMs. *arXiv preprint arXiv:2603.15957*, 2026.

[24] Carlos E. Jimenez, John Yang, Alexander Wettig, Shunyu Yao, Kexin Pei, Ofir Press, and Karthik R. Narasimhan. SWE-bench: Can language models resolve real-world GitHub issues? In *The Twelfth International Conference on Learning Representations*, 2024.

[25] Feng Ju et al. Reasoning path divergence: A new metric and curation strategy to unlock LLM diverse thinking. *arXiv preprint arXiv:2510.26122*, 2025.

[26] Richard M Karp. Reducibility among combinatorial problems. In *50 Years of Integer Programming 1958-2008: from the Early Years to the State-of-the-Art*, pages 219–241. Springer, 2009.

[27] Dénes König. Theory of finite and infinite graphs. In *Theory of Finite and Infinite Graphs*, pages 45–421. Springer, 1990.

[28] Joseph B Kruskal. On the shortest spanning subtree of a graph and the traveling salesman problem. *Proceedings of the American Mathematical society*, 7(1):48–50, 1956.

[29] Seonghyeon Lee et al. How diversely can language models solve problems? Exploring the algorithmic diversity of model-generated code. *arXiv preprint arXiv:2503.00691*, 2025.

[30] LeetCode. LeetCode: Online coding platform. `https://leetcode.com`, 2026.

[31] Joel Lehman and Kenneth O. Stanley. Abandoning objectives: Evolution through the search for novelty alone. *Evolutionary Computation*, 19(2):189–223, 2011.

[32] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O. Stanley. Evolution through large models. *arXiv preprint arXiv:2206.08896*, 2022.

[33] Xiaozhe Li et al. NP-Engine: Empowering optimization reasoning in LLMs with verifiable synthetic NP problems. *arXiv preprint arXiv:2510.16476*, 2025.

[34] Zhifei Li, Tian Xia, Ziming Mao, Zihan Zhou, Ethan J Jackson, Jamison Kerney, Zhanghao Wu, Pratik Mishra, Yi Xu, Yifan Qiao, et al. Skynomad: On using multi-region spot instances to minimize ai batch job cost. *arXiv preprint arXiv:2601.06520*, 2026.

[35] Fei Liu, Xialiang Tong, Mingxuan Yuan, Xi Lin, Fu Luo, Zhenkun Wang, Zhichao Lu, and Qingfu Zhang. Evolution of heuristics: Towards efficient automatic algorithm design using large language models. In *International Conference on Machine Learning*, 2024.

[36] Shu Liu, Shubham Agarwal, Monishwaran Maheswaran, Mert Cemri, Zhifei Li, Qiuyang Mang, Ashwin Naren, Ethan Boneh, Audrey Cheng, Melissa Z Pan, et al. Evox: Meta-evolution for automated discovery. *arXiv preprint arXiv:2602.23413*, 2026.

[37] Tengxiao Liu, Yuqing Yang, Xi Ye, and Danqi Chen. Can coding agents optimize algorithms autonomously? `https://tengxiaoliu.github.io/autoevolver/`, 2026. Accessed: 2026-05-06.

[38] Yifei Liu, Li Lyna Zhang, Yi Zhu, Bingcheng Dong, Xudong Zhou, Ning Shang, Fan Yang, and Mao Yang. rstar-coder: Scaling competitive code reasoning with a large-scale verified dataset. *arXiv preprint arXiv:2505.21297*, 2025.

[39] Ziyang Luo, Can Xu, Pu Zhao, Qingfeng Sun, Xiubo Geng, Wenxiang Hu, Chongyang Tao, Jing Ma, Qingwei Lin, and Daxin Jiang. WizardCoder: Empowering code large language models with Evol-Instruct. In *International Conference on Learning Representations*, 2024.

[40] Monishwaran Maheswaran, Leon Lakhani, Zhongzhu Zhou, Shijia Yang, Junxiong Wang, Coleman Hooper, Yuezhou Hu, Rishabh Tiwari, Jue Wang, Harman Singh, et al. Squeeze evolve: Unified multi-model orchestration for verifier-free evolution. *arXiv preprint arXiv:2604.07725*, 2026.

[41] Qiuyang Mang, Wenhao Chai, Zhifei Li, Huanzhi Mao, Shang Zhou, Alexander Du, Hanchen Li, Shu Liu, Edwin Chen, Yichuan Wang, Xieting Chu, Zerui Cheng, Yuan Xu, Tian Xia, Zirui Wang, Tianneng Shi, Jianzhu Yao, Yilong Zhao, Qizheng Zhang, Charlie Ruan, Zeyu Shen, Kaiyuan Liu, Runyuan He, Dong Xing, Zerui Li, Zirong Zeng, Yige Jiang, Lufeng Cheng, Ziyi Zhao, Youran Sun, Wesley Zheng, Meiyuwang Zhang, Ruyi Ji, Xuechang Tu, Zihan Zheng, Zexing Chen, Kangyang Zhou, Zhaozi Wang, Jingbang Chen, Aleksandra Korolova, Peter Henderson, Pramod Viswanath, Vijay Ganesh, Saining Xie, Zhuang Liu, Dawn Song, Sewon Min, Ion Stoica, Joseph E. Gonzalez, Jingbo Shang, and Alvin Cheung. FrontierCS: Evolving challenges for evolving intelligence. *arXiv preprint arXiv:2512.15699*, 2025.

[42] Moonshot AI. Kimi k2.6: Scaling agent orchestration with multimodal integration. `https://www.kimi.com/ai-models/kimi-k2-6`, 2026. Accessed: 2026-05-05.

[43] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909*, 2015.

[44] Subhash C Narula and Cesar A Ho. Degree-constrained minimum spanning tree. *Computers & Operations Research*, 7(4):239–249, 1980.

[45] Alexander Novikov et al. AlphaEvolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*, 2025.

[46] OpenAI. Introducing gpt-5.4. `https://openai.com/index/introducing-gpt-5-4/`, 2026. Accessed: 2026-04-26.

[47] OpenAI. Introducing gpt-5.5. `https://openai.com/index/introducing-gpt-5-5/`, 2026. Accessed: 2026-05-05.

[48] Anne Ouyang et al. KernelBench: Can LLMs write efficient GPU kernels? *arXiv preprint arXiv:2502.10517*, 2025.

[49] Julien Pourcel, Cédric Colas, Gaia Molinaro, Pierre-Yves Oudeyer, and Laetitia Teodorescu. ACES: Generating a diversity of challenging programming puzzles with autotelic generative models. *arXiv preprint arXiv:2310.10692*, 2024.

[50] Ao Qu, Han Zheng, Zijian Zhou, Yihao Yan, Yihong Tang, Shao Yong Ong, Fenglu Hong, Kaichen Zhou, Chonghe Jiang, Minwei Kong, et al. Coral: Towards autonomous multi-agent evolution for open-ended discovery. *arXiv preprint arXiv:2604.01658*, 2026.

[51] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi. Mathematical discoveries from program search with large language models. *Nature*, 625:468–475, 2024.

[52] Rulin Shao, Shuyue Stella Li, Rui Xin, Scott Geng, Yiping Wang, Sewoong Oh, Simon Shaolei Du, Nathan Lambert, Sewon Min, Ranjay Krishna, Yulia Tsvetkov, Hannaneh Hajishirzi, Pang Wei Koh, and Luke Zettlemoyer. Spurious rewards: Rethinking training signals in RLVR. *arXiv preprint arXiv:2506.10947*, 2025.

[53] Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

[54] Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

[55] Avi Singh, John D. Co-Reyes, Rishabh Agarwal, et al. Beyond human data: Scaling self-training for problem-solving with language models. *Transactions on Machine Learning Research*, 2024.

[56] Atharv Sonwane, Isadora White, Hyunji Lee, Matheus Pereira, Lucas Caccia, Minseon Kim, Zhengyan Shi, Chinmay Singh, Alessandro Sordoni, Marc-Alexandre Côté, and Xingdi Yuan. BugPilot: Complex bug generation for efficient learning of SWE skills. *arXiv preprint arXiv:2510.19898*, 2025.

[57] Michael Steinbrunn, Guido Moerkotte, and Alfons Kemper. Heuristic and randomized optimization for the join ordering problem. *The VLDB Journal*, 6(3):191–208, 1997.

[58] Qwen Team. Qwen3.5: Accelerating productivity with native multimodal agents, February 2026. URL `https://qwen.ai/blog?id=qwen3.5`.

[59] Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (POET): Endlessly generating increasingly complex and diverse learning environments and their solutions. *arXiv preprint arXiv:1901.01753*, 2019.

[60] Yinjie Wang et al. CURE: Co-evolving LLM coder and unit tester via reinforcement learning. *arXiv preprint arXiv:2506.03136*, 2025.

[61] Yiping Wang, Shao-Rong Su, Zhiyuan Zeng, Eva Xu, Liliang Ren, Xinyu Yang, Zeyi Huang, Xuehai He, Luyao Ma, Baolin Peng, et al. Thetaevolve: Test-time learning on open problems. *arXiv preprint arXiv:2511.23473*, 2025.

[62] Zihan Wang, Siyao Liu, Yang Sun, Hongyan Li, and Kai Shen. CodeContests+: High-quality test case generation for competitive programming. In *Findings of the Association for Computational Linguistics: EMNLP 2025*, pages 5576–5600, 2025.

[63] Zihan Wang, Chi Gui, Xing Jin, Qineng Wang, Licheng Liu, Kangrui Wang, Shiqi Chen, Linjie Li, Zhengyuan Yang, Pingyue Zhang, Yiping Lu, Jiajun Wu, Li Fei-Fei, Lijuan Wang, Yejin Choi, and Manling Li. RAGEN-2: Reasoning collapse in agentic RL. *arXiv preprint arXiv:2604.06268*, 2026.

[64] Hjalmar Wijk et al. RE-Bench: Evaluating frontier AI R&D capabilities of language model agents. *arXiv preprint arXiv:2411.15114*, 2024.

[65] Chunqiu Steven Xia, Yinlin Deng, and Lingming Zhang. Top leaderboard ranking = top coding proficiency, always? EvoEval: Evolving coding benchmarks via LLM. *arXiv preprint arXiv:2403.19114*, 2024.

[66] Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, and Daxin Jiang. WizardLM: Empowering large language models to follow complex instructions. *arXiv preprint arXiv:2304.12244*, 2023.

[67] John Yang, Kilian Lieret, Carlos E. Jimenez, Alexander Wettig, Kabir Khandpur, Yanzhe Zhang, Binyuan Hui, Ofir Press, Ludwig Schmidt, and Diyi Yang. SWE-smith: Scaling data for software engineering agents. *arXiv preprint arXiv:2504.21798*, 2025.

[68] Ziyu Ye et al. Evolving alignment via asymmetric self-play. *arXiv preprint arXiv:2411.00062*, 2024.

[69] Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. *arXiv preprint arXiv:2401.10020*, 2024.

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

[71] Weihao Zeng, Can Xu, Yingxiu Zhao, Jian-Guang Lou, and Weizhu Chen. Automatic instruction evolving for large language models. In *Empirical Methods in Natural Language Processing*, 2024.

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

[73] Shang Zhou, Zihan Zheng, Kaiyuan Liu, Zeyu Shen, Zerui Cheng, Zexing Chen, Hansen He, Jianzhu Yao, Huanzhi Mao, Qiuyang Mang, Tianfu Fu, Beichen Li, Dongruixuan Li, Wenhao Chai, Zhuang Liu, Aleksandra Korolova, Peter Henderson, Natasha Jaques, Pramod Viswanath, Saining Xie, and Jingbo Shang. AutoCode: LLMs as problem setters for competitive programming. *arXiv preprint arXiv:2510.12803*, 2025.

A Human Expert Grading Policy

We grade each synthesized open-ended problem on three dimensions, listed in priority order. Unlike an exact-output problem, an open-ended problem is not judged on whether one clever optimal solution exists. It is judged on whether the scoring formula spreads many distinct strategies apart, and whether the score ceiling resists easy saturation.

Dimension 1: Score discrimination and saturation risk. The scoring formula must separate weak, medium, strong, and excellent submissions.

- *Excellent*: scores cover a wide range; strong solutions gain ground steadily; reaching the top requires combining several nontrivial ideas.
- *Good*: most levels separate cleanly, but top solutions cluster.
- *Weak*: many reasonable solutions tie at similar scores, or one simple known heuristic already saturates the top.
- *Failing*: scores are effectively binary, the cap is hit too easily, or the problem admits an efficient exact solution.

Red flags: a low-effort baseline already sits near the cap; one obvious heuristic gets close to optimal; the score formula compresses real differences into a flat region near the top.

Dimension 2: Solution diversity. The problem must support several substantially different solution families.

- *Excellent*: many approaches stay viable: greedy construction, local search, randomized improvement, DP on reduced instances, clustering, approximation, exact subroutines, portfolio methods, and instance-specific strategies.
- *Good*: several distinct routes exist, but most strong solutions share one backbone.
- *Weak*: contestants are pushed toward one obvious method, with only parameter tuning to differentiate.
- *Failing*: the task is a disguised exact problem or a one-trick heuristic.

Dimension 3: Novelty and naturalness. The problem must read as a coherent optimization task on its own, not a scoring wrapper attached to a standard exact problem.

- *Excellent*: the objective has a standalone motivation, and admits tradeoffs that push different strategies in different directions.
- *Good*: the setting is recognizable, but the specific combination of constraints or objectives is fresh.
- *Weak*: the problem reuses a known open-ended problem format with cosmetic changes.
- *Failing*: the statement is contrived, or the scoring is pasted onto a standard exact problem rather than arising from it.

A strong open-ended problem rewards both algorithmic insight and engineering judgment, and does not reduce to "implement the intended heuristic."

B Example Synthesized Problems

Below are two example open-ended problems produced by FrontierSmith, each mutated from a different closed-ended seed.

B.1 Concat Factory Compression Challenge

Polycarp's toy language has only two commands:

1. **Literal creation** (`L s`): Creates a new variable whose value is the lowercase string `s` (length 1–8).

2. **Concatenation** (`C a b`): Creates a new variable whose value is the concatenation of the values of previously created variables `a` and `b`.

Variables are numbered automatically in creation order, starting from 1. For each test case, you are given a list of target strings. Your task is to output a valid program that constructs every target string in some variable, while keeping the program as small as possible.

Input

The first line contains an integer T (number of test cases). For each test case: the first line contains q (number of targets), followed by q lines each containing one target string.

Output

For each test case, output: (1) an integer m (number of commands); (2) m lines, each `L s` or `C a b`; (3) one line with q integers $r_1, \ldots, r_q$, where r_i is the index of a variable whose value equals the i-th target.

Feasibility

A solution is feasible if $1 \le m \le 5000$, every command has valid syntax, every literal has length 1–8 with only lowercase letters, every `C a b` references previously created variables, every variable's value has length at most the maximum target length, and for every i, $\texttt{value}[r_i] = t_i$.

Objective

For a feasible solution, the cost is

$$C = m + \sum_{\text{literal commands } L\ s} |s|.$$

The baseline cost B for each test case is computed by splitting each target into blocks of length at most 8 without reuse:

$$B = \sum_{\text{targets } t} \left(|t| + 2 \left\lceil \frac{|t|}{8} \right\rceil - 1 \right).$$

The test-case score is $S = 10^6 \cdot \min(2,\ B/C)$. The final score is the mean of S over all test cases.

Constraints

$1 \le T \le 20$, $1 \le q \le 200$, $1 \le |t_i| \le 400$, sum of target lengths per test case $\le 40{,}000$. Time limit: 3 s. Memory limit: 512 MB.

Example

Input: 3 targets `haha`, `ahaha`, `hahahaha`.

Program: `L ha`, `L a`, `C 1 1`, `C 2 3`, `C 3 3`. Variables: 1=`ha`, 2=`a`, 3=`haha`, 4=`ahaha`, 5=`hahahaha`. Targets matched by variables 3, 4, 5.

Cost $C = 5 + (2 + 1) = 8$. Baseline $B = 5 + 6 + 9 = 20$. Score $= 10^6 \cdot \min(2, 20/8) = 2 \times 10^6$ (cap).

B.2 Yae Village Defense Network

The ancient village of Yae has n houses, numbered $1 \ldots n$. House 1 is the Grand Shrine, where Sakura starts and ends every day. There are m candidate bidirectional roads; road i connects houses u_i, v_i with build cost c_i and travel time d_i.

Before day 1, Sakura may build a subset of roads with total cost $\le B$. Over D days, Collapse beasts attack: attack j targets house h_j, is active on days $[a_j, b_j]$, and causes damage w_j if not cleared. On each day t, Sakura starts at house 1, walks along built roads (total travel time $\le L_t$), and must return to house 1. Visiting a house clears all active attacks there.

The task is to choose which roads to build and one patrol per day to **minimize total damage** from uncleared attacks.

Input

One instance: n, m, D, A (number of attacks), B (budget), followed by m road descriptions (u_i, v_i, c_i, d_i), daily time limits $L_1, \ldots, L_D$, and A attack descriptions (h_j, a_j, b_j, w_j).

Output

Line 1: k built road indices. Lines 2 to $D + 1$: for each day, a sequence of built road indices forming a walk from house 1 back to house 1.

Feasibility

Built roads must have total cost $\leq B$; each daily patrol must use only built roads, follow valid adjacencies, start and end at house 1, and respect the daily time limit L_t.

Objective

Minimize Damage $= \sum_{j \text{ not cleared}} w_j$. The baseline builds no roads and stays at house 1 every day, clearing only attacks at house 1. For baseline damage B_{dmg} and your damage D_{your}:

$$\text{Score} = \lfloor 10^6 \cdot \min(10,\ (B_{\text{dmg}} + 1)/(D_{\text{your}} + 1)) \rfloor.$$

Final score is the mean over all test files.

Constraints

$2 \leq n \leq 400$, $1 \leq m \leq 3000$, $1 \leq D \leq 60$, $1 \leq A \leq 4000$, $1 \leq c_i, d_i \leq 10^6$, $1 \leq B \leq 10^9$, $0 \leq L_t \leq 10^9$, $1 \leq w_j \leq 10^{12}$. Time limit: 3 s. Memory limit: 256 MB.

Example

4 houses, 4 candidate roads, 3 days, 4 attacks, budget 4. Building roads 1–2 and 2–3 (cost 4) and patrolling: day 1 visits house 2 (clears attack, energy 5), day 2 visits house 3 (clears attack, energy 8), day 3 stays home (clears house-1 attack, energy 4). The attack at house 4 is missed: total damage $= 6$, baseline damage $= 19$, score $= \lfloor 10^6 \cdot (19{+}1)/(6{+}1) \rfloor = 2{,}857{,}142$.

NeurIPS Paper Checklist

1. **Claims**

   Question: Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope?

   Answer: [Yes]

   Justification: All three key findings stated in the introduction are supported by Table 1 (§4.2), Figure 4 (§4.3), and Figures 5–6 (§4.3–4.4). Limitations (e.g., single-seed runs, 9B FrontierCS gap) are acknowledged in the text.

   Guidelines:

   - The answer [N/A] means that the abstract and introduction do not include the claims made in the paper.
   - The abstract and/or introduction should clearly state the claims made, including the contributions made in the paper and important assumptions and limitations. A [No] or [N/A] answer to this question will not be perceived well by the reviewers.
   - The claims made should match theoretical and experimental results, and reflect how much the results can be expected to generalize to other settings.
   - It is fine to include aspirational goals as motivation as long as it is clear that these goals are not attained by the paper.

2. **Limitations**

   Question: Does the paper discuss the limitations of the work performed by the authors?

   Answer: [Yes]

   Justification: §5 discusses two main limitations: restriction to self-contained algorithmic environments (no repo-level tasks), and limited RL training scale (no agentic RL).

   Guidelines:

   - The answer [N/A] means that the paper has no limitation while the answer [No] means that the paper has limitations, but those are not discussed in the paper.
   - The authors are encouraged to create a separate "Limitations" section in their paper.
   - The paper should point out any strong assumptions and how robust the results are to violations of these assumptions (e.g., independence assumptions, noiseless settings, model well-specification, asymptotic approximations only holding locally). The authors should reflect on how these assumptions might be violated in practice and what the implications would be.
   - The authors should reflect on the scope of the claims made, e.g., if the approach was only tested on a few datasets or with a few runs. In general, empirical results often depend on implicit assumptions, which should be articulated.
   - The authors should reflect on the factors that influence the performance of the approach. For example, a facial recognition algorithm may perform poorly when image resolution is low or images are taken in low lighting. Or a speech-to-text system might not be used reliably to provide closed captions for online lectures because it fails to handle technical jargon.
   - The authors should discuss the computational efficiency of the proposed algorithms and how they scale with dataset size.
   - If applicable, the authors should discuss possible limitations of their approach to address problems of privacy and fairness.
   - While the authors might fear that complete honesty about limitations might be used by reviewers as grounds for rejection, a worse outcome might be that reviewers discover limitations that aren't acknowledged in the paper. The authors should use their best judgment and recognize that individual actions in favor of transparency play an important role in developing norms that preserve the integrity of the community. Reviewers will be specifically instructed to not penalize honesty concerning limitations.

3. **Theory assumptions and proofs**

   Question: For each theoretical result, does the paper provide the full set of assumptions and a complete (and correct) proof?

Answer: [N/A]

Justification: The paper is primarily empirical and does not include formal theorems or proofs.

Guidelines:

- The answer [N/A] means that the paper does not include theoretical results.
- All the theorems, formulas, and proofs in the paper should be numbered and cross-referenced.
- All assumptions should be clearly stated or referenced in the statement of any theorems.
- The proofs can either appear in the main paper or the supplemental material, but if they appear in the supplemental material, the authors are encouraged to provide a short proof sketch to provide intuition.
- Inversely, any informal proof provided in the core of the paper should be complemented by formal proofs provided in appendix or supplemental material.
- Theorems and Lemmas that the proof relies upon should be properly referenced.

4. **Experimental result reproducibility**

   Question: Does the paper fully disclose all the information needed to reproduce the main experimental results of the paper to the extent that it affects the main claims and/or conclusions of the paper (regardless of whether the code and data are provided or not)?

   Answer: [Yes]

   Justification: §4.1 specifies all pipeline parameters, model choices, RL hyperparameters, and GPU configurations. Algorithm 1 describes the full synthesis pipeline.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - If the paper includes experiments, a [No] answer to this question will not be perceived well by the reviewers: Making the paper reproducible is important, regardless of whether the code and data are provided or not.
   - If the contribution is a dataset and/or model, the authors should describe the steps taken to make their results reproducible or verifiable.
   - Depending on the contribution, reproducibility can be accomplished in various ways. For example, if the contribution is a novel architecture, describing the architecture fully might suffice, or if the contribution is a specific model and empirical evaluation, it may be necessary to either make it possible for others to replicate the model with the same dataset, or provide access to the model. In general. releasing code and data is often one good way to accomplish this, but reproducibility can also be provided via detailed instructions for how to replicate the results, access to a hosted model (e.g., in the case of a large language model), releasing of a model checkpoint, or other means that are appropriate to the research performed.
   - While NeurIPS does not require releasing code, the conference does require all submissions to provide some reasonable avenue for reproducibility, which may depend on the nature of the contribution. For example
     (a) If the contribution is primarily a new algorithm, the paper should make it clear how to reproduce that algorithm.
     (b) If the contribution is primarily a new model architecture, the paper should describe the architecture clearly and fully.
     (c) If the contribution is a new model (e.g., a large language model), then there should either be a way to access this model for reproducing the results or a way to reproduce the model (e.g., with an open-source dataset or instructions for how to construct the dataset).
     (d) We recognize that reproducibility may be tricky in some cases, in which case authors are welcome to describe the particular way they provide for reproducibility. In the case of closed-source models, it may be that access to the model is limited in some way (e.g., to registered users), but it should be possible for other researchers to have some path to reproducing or verifying the results.

5. **Open access to data and code**

Question: Does the paper provide open access to the data and code, with sufficient instructions to faithfully reproduce the main experimental results, as described in supplemental material?

Answer: [No]

Justification: We plan to release the trained model weights and a subset of synthesized problems. The synthesis pipeline code is not released at this time.

Guidelines:

- The answer [N/A] means that paper does not include experiments requiring code.
- Please see the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- While we encourage the release of code and data, we understand that this might not be possible, so [No] is an acceptable answer. Papers cannot be rejected simply for not including code, unless this is central to the contribution (e.g., for a new open-source benchmark).
- The instructions should contain the exact command and environment needed to run to reproduce the results. See the NeurIPS code and data submission guidelines (`https://neurips.cc/public/guides/CodeSubmissionPolicy`) for more details.
- The authors should provide instructions on data access and preparation, including how to access the raw data, preprocessed data, intermediate data, and generated data, etc.
- The authors should provide scripts to reproduce all experimental results for the new proposed method and baselines. If only a subset of experiments are reproducible, they should state which ones are omitted from the script and why.
- At submission time, to preserve anonymity, the authors should release anonymized versions (if applicable).
- Providing as much information as possible in supplemental material (appended to the paper) is recommended, but including URLs to data and code is permitted.

6. **Experimental setting/details**

Question: Does the paper specify all the training and test details (e.g., data splits, hyperparameters, how they were chosen, type of optimizer) necessary to understand the results?

Answer: [Yes]

Justification: §4.1 details benchmarks, baselines, synthesis pipeline parameters, model choices, RL optimizer (GRPO), learning rates, batch sizes, and GPU configurations.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The experimental setting should be presented in the core of the paper to a level of detail that is necessary to appreciate the results and make sense of them.
- The full details can be provided either with the code, in appendix, or as supplemental material.

7. **Experiment statistical significance**

Question: Does the paper report error bars suitably and correctly defined or other appropriate information about the statistical significance of the experiments?

Answer: [No]

Justification: RL training results (Table 1) are single runs due to compute constraints; no multi-seed error bars are provided. Figure 5(a) reports standard deviations for idea divergence scores.

Guidelines:

- The answer [N/A] means that the paper does not include experiments.
- The authors should answer [Yes] if the results are accompanied by error bars, confidence intervals, or statistical significance tests, at least for the experiments that support the main claims of the paper.

- The factors of variability that the error bars are capturing should be clearly stated (for example, train/test split, initialization, random drawing of some parameter, or overall run with given experimental conditions).
- The method for calculating the error bars should be explained (closed form formula, call to a library function, bootstrap, etc.)
- The assumptions made should be given (e.g., Normally distributed errors).
- It should be clear whether the error bar is the standard deviation or the standard error of the mean.
- It is OK to report 1-sigma error bars, but one should state it. The authors should preferably report a 2-sigma error bar than state that they have a 96% CI, if the hypothesis of Normality of errors is not verified.
- For asymmetric distributions, the authors should be careful not to show in tables or figures symmetric error bars that would yield results that are out of range (e.g., negative error rates).
- If error bars are reported in tables or plots, the authors should explain in the text how they were calculated and reference the corresponding figures or tables in the text.

8. **Experiments compute resources**

   Question: For each experiment, does the paper provide sufficient information on the computer resources (type of compute workers, memory, time of execution) needed to reproduce the experiments?

   Answer: [Yes]

   Justification: §4.1 reports GPU types and training time (9B: 8×A100, 1.5 days; 27B: 32×H200, 1.5 days). §5 reports LLM API costs for synthesis.

   Guidelines:

   - The answer [N/A] means that the paper does not include experiments.
   - The paper should indicate the type of compute workers CPU or GPU, internal cluster, or cloud provider, including relevant memory and storage.
   - The paper should provide the amount of compute required for each of the individual experimental runs as well as estimate the total compute.
   - The paper should disclose whether the full research project required more compute than the experiments reported in the paper (e.g., preliminary or failed experiments that didn't make it into the paper).

9. **Code of ethics**

   Question: Does the research conducted in the paper conform, in every respect, with the NeurIPS Code of Ethics `https://neurips.cc/public/EthicsGuidelines`?

   Answer: [Yes]

   Justification: We have reviewed the NeurIPS Code of Ethics and our research conforms with it.

   Guidelines:

   - The answer [N/A] means that the authors have not reviewed the NeurIPS Code of Ethics.
   - If the authors answer [No], they should explain the special circumstances that require a deviation from the Code of Ethics.
   - The authors should make sure to preserve anonymity (e.g., if there is a special consideration due to laws or regulations in their jurisdiction).

10. **Broader impacts**

    Question: Does the paper discuss both potential positive societal impacts and negative societal impacts of the work performed?

    Answer: [Yes]

    Justification: §6 discusses both positive impacts (democratizing RL training data, accelerating scientific computing) and potential negative impacts (lowering barriers for exploit generation).

Guidelines:

- The answer [N/A] means that there is no societal impact of the work performed.
- If the authors answer [N/A] or [No], they should explain why their work has no societal impact or why the paper does not address societal impact.
- Examples of negative societal impacts include potential malicious or unintended uses (e.g., disinformation, generating fake profiles, surveillance), fairness considerations (e.g., deployment of technologies that could make decisions that unfairly impact specific groups), privacy considerations, and security considerations.
- The conference expects that many papers will be foundational research and not tied to particular applications, let alone deployments. However, if there is a direct path to any negative applications, the authors should point it out. For example, it is legitimate to point out that an improvement in the quality of generative models could be used to generate Deepfakes for disinformation. On the other hand, it is not needed to point out that a generic algorithm for optimizing neural networks could enable people to train models that generate Deepfakes faster.
- The authors should consider possible harms that could arise when the technology is being used as intended and functioning correctly, harms that could arise when the technology is being used as intended but gives incorrect results, and harms following from (intentional or unintentional) misuse of the technology.
- If there are negative societal impacts, the authors could also discuss possible mitigation strategies (e.g., gated release of models, providing defenses in addition to attacks, mechanisms for monitoring misuse, mechanisms to monitor how a system learns from feedback over time, improving the efficiency and accessibility of ML).

11. **Safeguards**

    Question: Does the paper describe safeguards that have been put in place for responsible release of data or models that have a high risk for misuse (e.g., pre-trained language models, image generators, or scraped datasets)?

    Answer: [N/A]

    Justification: Our work synthesizes algorithmic coding problems and fine-tunes existing open-weight models; it does not pose high risks for misuse.

    Guidelines:

    - The answer [N/A] means that the paper poses no such risks.
    - Released models that have a high risk for misuse or dual-use should be released with necessary safeguards to allow for controlled use of the model, for example by requiring that users adhere to usage guidelines or restrictions to access the model or implementing safety filters.
    - Datasets that have been scraped from the Internet could pose safety risks. The authors should describe how they avoided releasing unsafe images.
    - We recognize that providing effective safeguards is challenging, and many papers do not require this, but we encourage authors to take this into account and make a best faith effort.

12. **Licenses for existing assets**

    Question: Are the creators or original owners of assets (e.g., code, data, models), used in the paper, properly credited and are the license and terms of use explicitly mentioned and properly respected?

    Answer: [Yes]

    Justification: All benchmarks, models, and tools are cited. FrontierCS, ALE-bench, HardTests, Qwen3.5, veRL, and all LLM APIs are properly credited.

    Guidelines:

    - The answer [N/A] means that the paper does not use existing assets.
    - The authors should cite the original paper that produced the code package or dataset.
    - The authors should state which version of the asset is used and, if possible, include a URL.

- The name of the license (e.g., CC-BY 4.0) should be included for each asset.
- For scraped data from a particular source (e.g., website), the copyright and terms of service of that source should be provided.
- If assets are released, the license, copyright information, and terms of use in the package should be provided. For popular datasets, `paperswithcode.com/datasets` has curated licenses for some datasets. Their licensing guide can help determine the license of a dataset.
- For existing datasets that are re-packaged, both the original license and the license of the derived asset (if it has changed) should be provided.
- If this information is not available online, the authors are encouraged to reach out to the asset's creators.

13. **New assets**

    Question: Are new assets introduced in the paper well documented and is the documentation provided alongside the assets?

    Answer: [Yes]

    Justification: We plan to release trained model weights and a subset of synthesized problems. Example problems are documented in Appendix B.

    Guidelines:

    - The answer [N/A] means that the paper does not release new assets.
    - Researchers should communicate the details of the dataset/code/model as part of their submissions via structured templates. This includes details about training, license, limitations, etc.
    - The paper should discuss whether and how consent was obtained from people whose asset is used.
    - At submission time, remember to anonymize your assets (if applicable). You can either create an anonymized URL or include an anonymized zip file.

14. **Crowdsourcing and research with human subjects**

    Question: For crowdsourcing experiments and research with human subjects, does the paper include the full text of instructions given to participants and screenshots, if applicable, as well as details about compensation (if any)?

    Answer: [Yes]

    Justification: The human expert evaluation rubric is provided in Appendix A. The study setup (blind ranking, 10 groups, 2–3 experts per group) is described in §5.

    Guidelines:

    - The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
    - Including this information in the supplemental material is fine, but if the main contribution of the paper involves human subjects, then as much detail as possible should be included in the main paper.
    - According to the NeurIPS Code of Ethics, workers involved in data collection, curation, or other labor should be paid at least the minimum wage in the country of the data collector.

15. **Institutional review board (IRB) approvals or equivalent for research with human subjects**

    Question: Does the paper describe potential risks incurred by study participants, whether such risks were disclosed to the subjects, and whether Institutional Review Board (IRB) approvals (or an equivalent approval/review based on the requirements of your country or institution) were obtained?

    Answer: [N/A]

    Justification: The human evaluation involved expert co-authors and collaborators ranking problem quality; no external human subjects were recruited.

    Guidelines:

- The answer [N/A] means that the paper does not involve crowdsourcing nor research with human subjects.
- Depending on the country in which research is conducted, IRB approval (or equivalent) may be required for any human subjects research. If you obtained IRB approval, you should clearly state this in the paper.
- We recognize that the procedures for this may vary significantly between institutions and locations, and we expect authors to adhere to the NeurIPS Code of Ethics and the guidelines for their institution.
- For initial submissions, do not include any information that would break anonymity (if applicable), such as the institution conducting the review.

16. **Declaration of LLM usage**

    Question: Does the paper describe the usage of LLMs if it is an important, original, or non-standard component of the core methods in this research? Note that if the LLM is used only for writing, editing, or formatting purposes and does *not* impact the core methodology, scientific rigor, or originality of the research, declaration is not required.

    Answer: [Yes]

    Justification: LLMs are core to our method: GPT-5.4 Thinking for mutation and filtering, Claude Sonnet 4.6 for solution sampling and test infrastructure generation (§4.1). The trained models (Qwen3.5) are also LLMs.

    Guidelines:

    - The answer [N/A] means that the core method development in this research does not involve LLMs as any important, original, or non-standard components.
    - Please refer to our LLM policy in the NeurIPS handbook for what should or should not be described.