QUESTIONING THE QUESTIONS: SUSTAINING SELF-EVOLUTION IN REASONING MODELS

Anonymous authors
Paper under double-blind review

ABSTRACT

Self-evolving reasoning models learn from their own generated questions, yet repeated self-training can lead to performance collapse. In this paper, we investigate why performance deteriorates over successive rounds and how to sustain self-evolution. Our analysis identifies two recurring quality problems in self-generated questions: invalid questions and repeated variants of the same mathematical questions. First, invalid questions become more prevalent across rounds, and answer-consistency filtering further increases their proportion in training data. Second, existing question diversity controls based on lexical similarity can miss mathematically equivalent questions expressed in different ways, which leads to question diversity collapse in later training rounds. Building on these findings, we introduce R-Quest, which uses question validity and novelty feedback to guide self-evolution. We first train the solver to recognize and reject invalid questions, then use its judgments to guide questioner rewards and filter solver training data. To avoid question repetition, we use a frozen base model to compare sampled question pairs and provide novelty feedback. Empirically, our method consistently achieves the highest average performance on 12 benchmarks in mathematical reasoning, general-domain reasoning, and code generation across two model families. Additionally, R-Quest maintains stable performance gains over ten rounds of self-evolution, peaking in the final round and outperforming R-Zero by 17.32 points.

1 INTRODUCTION

Self-evolving reasoning models learn by generating and solving their own training tasks (Huang et al., 2026a; 2023; Tao et al., 2024; Tan et al., 2024; Yu et al., 2025; Xiang et al., 2026), reducing their dependence on human-curated data. Recent frameworks organize this process around questioner and solver roles: the questioner learns to propose challenging problems, while the solver learns to solve them, allowing both roles to co-evolve through reinforcement learning (Zhao et al., 2026a; Chen et al., 2026; Liu et al., 2025; Shi et al., 2026). However, continued self-evolution can reverse early gains and ultimately lead to performance collapse (Huang et al., 2025; Li et al., 2026a). In this paper, we study the practical bottlenecks to sustained self-evolution: *Why does performance deteriorate over successive rounds, and how can we address these bottlenecks to achieve sustained self-improvement?*

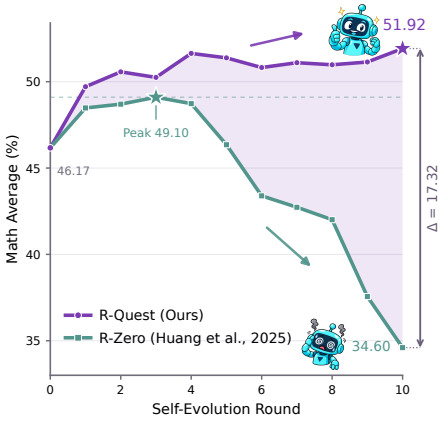


Figure 1: Average performance on seven math benchmarks. R-Quest exceeds the peak of R-Zero in every round and peaks at round ten, leading by **17.32** points.

To understand why these gains fail to persist, we examine the self-generated questions by R-Zero (Huang et al., 2025). Our analysis reveals two recurring quality issues: (1) invalid questions and (2) repeated variants of the same mathematical tasks. **First**, some generated questions are invalid (*e.g.*, due to ambiguous definitions, missing conditions or contradictory assumptions), yet the solver is not explicitly trained to reject such questions, as shown in Figure 2(a). The loop therefore lacks a mechanism to exclude or suppress invalid questions. We find that invalid questions become more

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

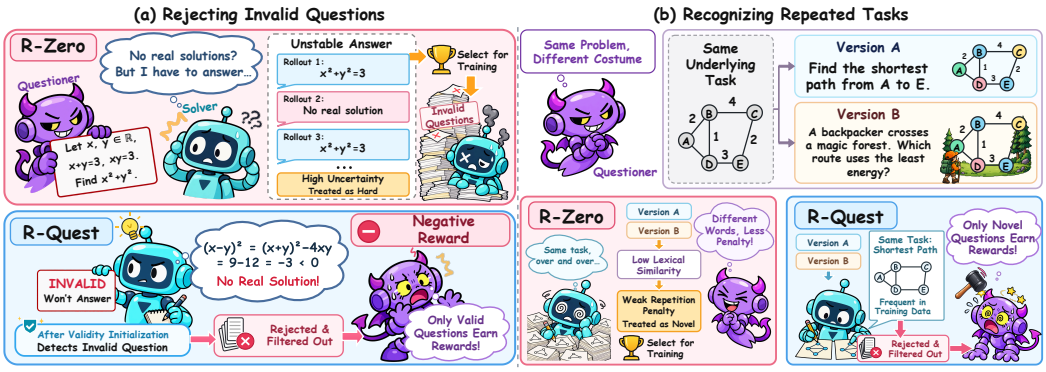


Figure 2: Two question-quality problems in self-evolution and how R-Quest addresses them: (a) R-Quest can recognize and reject invalid questions; (b) R-Quest can detect repeated mathematical tasks despite differences in expression.

prevalent across rounds and that answer-consistency filtering further increases their proportion in training data. Our comparison study further shows that repairing those invalid questions improves solver performance under matched training settings, indicating that invalid questions are a contributing factor to performance degradation and motivating explicit validity feedback during self-evolution.

Second, as shown in Figure 2(b), the questioner can repeatedly generate variants of the same underlying mathematical task by changing the wording, notation, or numerical parameters, and easily bypass existing repetition penalties. We observe substantial concentration in a few mathematical task types in R-Zero's fifth round training data. Existing diversity controls could split questions with the same task structure into separate clusters. Because repetition penalties depend on cluster size, this fragmentation can weaken the penalty on frequently repeated tasks. These findings suggest that effective diversity control must be grounded in an understanding of what a question asks and how it is solved, so that different expressions of the same task are recognized as repeated training content.

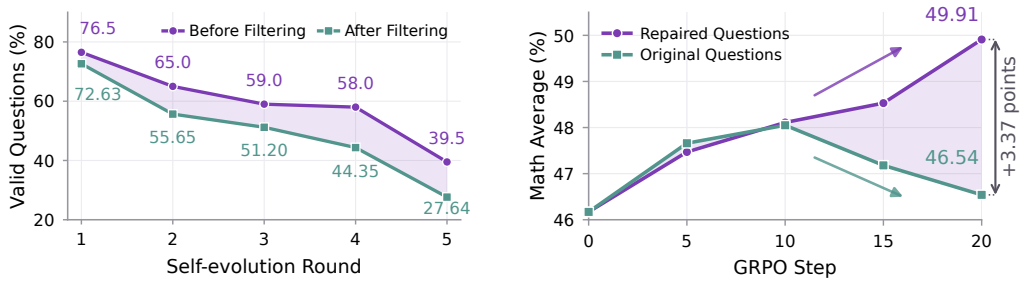
Building on these insights, we introduce R-Quest, which promotes sustained self-evolution by explicitly assessing the validity of generated questions and the novelty of their underlying mathematical tasks. We first train the solver to recognize and reject invalid questions. During self-evolution, we incorporate its validity judgments into the questioner's rewards to discourage invalid generation. We also assess task novelty through sampled pairwise comparisons, using the model's understanding of the questions to identify repeated mathematical task structures. Together, these judgments restrict the questioner's difficulty rewards to questions assessed as valid and novel. Both feedback signals are generated during subsequent self-evolution without further calls to stronger external models.

Across two model families, R-Quest achieves the highest domain-average scores among evaluated methods on 12 benchmarks spanning mathematics, code generation, and general reasoning. As shown in Figure 1, R-Quest sustains its gains over ten rounds of self-evolution, reaching its best mathematical performance in the final round and outperforming R-Zero by 17.32 points at the same iteration on Qwen3-4B-Base. After ten rounds of self-evolution, R-Quest is 5.75 points above the base model, whereas R-Zero is 11.57 points below it. These results demonstrate that assessing the validity and task novelty of self-generated questions provides a practical route to sustained self-improvement.

2 Preliminary Study

2.1 Background: The Questioner–Solver Framework

Self-evolution for mathematical reasoning commonly adopts a questioner–solver co-training framework. Following the R-Zero paradigm (Huang et al., 2025), both models share the same initialization and are trained alternately using Group Relative Policy Optimization (GRPO) (Shao et al., 2024). For each question q generated by the questioner, the frozen solver samples M responses. Let $\tilde{a}(q)$ be the majority answer and a_j the extracted answer of the j-th response; the solver's consistency is defined as $s(q) = \frac{1}{M}\sum_{j=1}^{M} \mathbb{1}[a_j = \tilde{a}(q)]$. The questioner is trained to generate novel and moderately challenging questions by rewarding uncertainty, with the reward maximized at $s(q) = 1/2$, while penalizing repetition. Questions are grouped by lexical similarity using pairwise BLEU distances,



(a) (b)

Figure 3: Question validity rate and solver performance on Qwen3-4B-Base. (a) Valid-question rates before and after R-Zero's answer-consistency filter. (b) Average math reasoning performance of the solver trained on matched original and repaired datasets.

and each receives a repetition penalty of $|C(q)|/B$, where $C(q)$ is its cluster and B is the batch size. Subsequently, the frozen questioner proposes new questions, and only those satisfying intermediate consistency ($|s(q) - 1/2| \leq \delta$) are retained to train the solver. The solver is then updated via GRPO using self-consistency pseudo-labels, assigning binary rewards $\mathbb{1}[a(y) = \tilde{a}(q)]$ to its responses.

2.2 Questioning the Questions: Diagnosing Failures in Self-Evolution

2.2.1 Invalid Questions in Solver Training

Self-generated questions are not always well posed. Our investigation into the collapse of R-Zero (Huang et al., 2025) reveals a growing proportion of invalid generated questions. We randomly sample 200 questions from the question pool of each of the five R-Zero rounds and use GPT-5.6-Sol to assess their validity. Figure 3(a) shows that the valid-question rate falls from 76.5% in round one to 39.5% in round five, leaving most questions invalid. In addition, R-Zero would filter questions based on majority-voted answer agreement $s(q)$, retaining those between 0.3 and 0.8. We find that among the retained questions, the valid question rate is further reduced, reaching 27.64% in round five. Thus, answer-consistency filtering increases the proportion of invalid questions in solver training data.

To test whether invalid questions contribute to solver degradation, we compare GRPO training on matched original and repaired datasets using Qwen3-4B-Base. Each dataset contains 1,273 questions, including the same 766 valid questions. The remaining 507 questions are invalid in the original dataset and replaced by their repaired counterparts in the other. We use GPT-5.6-Sol to repair invalid questions while preserving the domain and minimizing changes in difficulty. Figure 3(b) shows comparable early performance followed by diverging trajectories: training on the repaired dataset yields a final average score of 49.91 across seven mathematical benchmarks, compared with 46.54 on the original dataset. These results indicate that invalid training questions are a contributing factor to solver degradation in self-evolution.

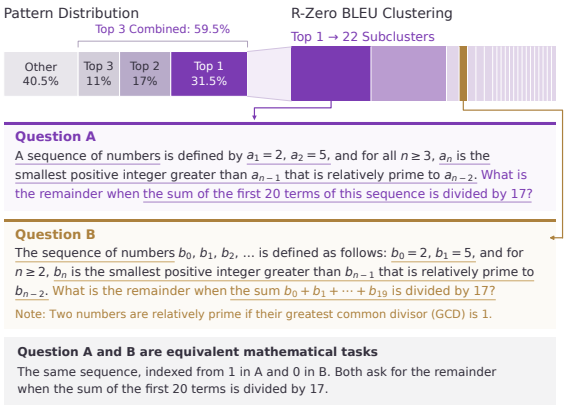


Figure 4: Repeated question distribution and BLEU fragmentation in the fifth round of R-Zero. BLEU separates the illustrated mathematically equivalent questions into multiple different clusters.

2.2.2 Repetition in Generated Questions

Generated questions can repeat the same mathematical task structure despite differences in wording, notation, or numerical parameters. Although R-Zero discourages repetition through BLEU-based clustering, lexical similarity may not capture repetition at the level of question types. In Figure 4, we show a pair of mathematically equivalent questions in two separate BLEU clusters: they share the same sequence construction and requested computation, despite differences in wording and indexing.

To quantify the prevalence of repeated question types, we use GPT-5.6-Sol to group 200 questions randomly sampled from the filtered question pool generated in R-Zero's fifth round. Questions in each group share the same mathematical task structure, and thus should be considered as "repeated" questions. The grouping protocol is detailed in Appendix B.3. As shown in Figure 4, the three largest question types account for 59.5% of the sample. Under the BLEU-based clustering procedure of R-Zero, the largest type of question is split into 22 clusters. Such fragmentation can weaken the repetition penalty (which is based on cluster size) on a frequently repeated type. These findings motivate us to assess repetition based on the underlying task structure.

3 METHOD

As shown in Figure 2, R-Quest introduces two forms of feedback to support sustained self-evolution: (1) validity feedback from a solver initialized to recognize and reject invalid questions; and (2) novelty feedback from the frozen base model, which compares sampled question pairs to identify repeated tasks. These signals guide questioner rewards during alternating questioner–solver updates.

3.1 VALIDITY-AWARE SOLVER INITIALIZATION

Our preliminary analysis in Section 2.2 examines invalid questions as a contributor to performance collapse during self-evolution. We thus introduce validity-aware solver initialization to teach the solver to recognize invalid questions before self-evolution.

We construct a dataset by deduplicating questions from the solver training sets of five R-Zero iterations and randomly sampling an equal number from each iteration. We use GPT-5.6-Sol to annotate these questions in two stages. First, the annotator attempts to solve each question and assesses whether it is valid. Second, it independently solves only the valid questions to produce reference answers. We further review these answers and discard examples whose answers cannot be reliably verified. The dataset statistics and annotation protocol are provided in Appendix A.1 and A.2, respectively.

We then train the base model with GRPO to return a mathematical answer for a valid question or `INVALID` for an invalid one. The model is instructed to attempt a solution and use `INVALID` only when it identifies a flaw in the question itself. We use the reward

$$R_{\text{init}}(q, a) = \mathbb{1}[a = a^*(q)] - \lambda\, \mathbb{1}[a = \texttt{INVALID} \wedge a^*(q) \neq \texttt{INVALID}]\,,$$

where $a^*(q)$ is the reference answer for a valid question and `INVALID` otherwise, and $\lambda > 0$ controls the penalty for rejecting valid questions to discourage excessive refusal. The full reward specification and training settings are provided in Appendix A.3. We initialize the solver with this model, bringing its ability to recognize invalid questions into the self-evolution loop.

3.2 QUESTION NOVELTY ASSESSMENT VIA SAMPLED PAIRWISE JUDGMENTS

As discussed in Section 2.2, question diversity is another dimension along which self-evolution can collapse. We use the frozen base model to perform diversity judgment throughout self-evolution. Two questions are considered the same type when they share essentially the same mathematical setup and ask for the same kind of task. Changes in numerical values or wording alone do not establish a new type, while belonging to the same broad mathematical field is insufficient to establish repetition. The full task specification is provided in Appendix B.1.

Let B denote the number of candidate questions in a questioner rollout batch during GRPO training, and let $p_{ij} \in \{0, 1\}$ indicate whether questions i and j are judged to be of the same type. These binary pairwise judgments define an exhaustive novelty gate, $\prod_{j \neq i}(1 - p_{ij})$, which equals one only when no repetition is detected: a single same-type match is sufficient for rejection. However, evaluating this gate for all candidates requires $O(B^2)$ model judgments.

To reduce this cost, we uniformly sample K other questions per candidate without replacement and reject the candidate if any reference is judged to be of the same type. This reduces the total comparison cost to $O(BK)$ and relaxes the exhaustive gate, with K controlling how strongly repeated task types are suppressed. Let p denote the fraction of the remaining questions that the judge would classify as the same type. When K is small relative to B, the probability of passing is $\Pr(\text{pass} \mid p) \approx (1 - p)^K$. For example, with $K = 8$, a same-type fraction of 10% yields a pass probability of approximately

43%, compared with only 6% at a fraction of 30%. Appendix B.2 provides the exact pass probability and a plot showing how it varies with the same-type fraction p and the comparison budget K.

3.3 Self-Evolution with Validity and Novelty Feedback

Questioner Rewards. We initialize the solver with the validity-aware model from Section 3.1, bringing its ability to recognize and reject invalid questions into self-evolution. For a generated question q, it is first instructed to assess validity while attempting a solution and return `INVALID` if it identifies a flaw in the question. We sample multiple responses and denote the fraction returning `INVALID` by $u(q)$. We define the validity gate as $g^{\text{valid}}(q) = \frac{1}{2} - u(q)$. A question is classified as invalid when $g^{\text{valid}}(q) < 0$, meaning that more than half of the responses return `INVALID`.

For questions that pass this check, we switch to a standard problem-solving instruction that asks for reasoning and a final mathematical answer, and sample a fresh set of responses. Let $s(q)$ be the proportion of responses in the largest answer cluster. We define $g^{\text{novel}}(q)$ to be 1 if q passes the sampled novelty check in Section 3.2, and 0 otherwise. The questioner reward is

$$R_{\text{question}}(q) = \begin{cases} g^{\text{valid}}(q), & g^{\text{valid}}(q) < 0, \\ g^{\text{novel}}(q) R_{\text{uncertainty}}(q), & g^{\text{valid}}(q) \geq 0, \end{cases} \quad R_{\text{uncertainty}}(q) = \min\{s(q), 1 - s(q)\}.$$

where $R_{\text{uncertainty}}(q)$ is the uncertainty reward based on answer consistency. The questioner is updated with GRPO using this reward. The prompts for validity assessment and standard problem solving are provided in the Appendix Prompts A.5 and C.2, respectively.

Solver Updates. The updated questioner generates candidate training questions. We first apply the same majority-vote validity check, then sample mathematical solutions for questions that pass. Consistent with R-Zero (Huang et al., 2025), we construct the training set using answer-consistency filtering and majority-vote pseudo-labels. The solver is updated with GRPO on this dataset, mixed with a small, fixed proportion of the solver initialization data to maintain its ability to recognize and reject invalid questions. Replay examples retain the validity-aware task and reward defined in Appendix A.3. The questioner and solver are then updated alternately. Sampling settings, filtering thresholds, and the replay proportion are provided in Appendix C.2.

4 Experiments

4.1 Experimental Setup

Models and Baselines. We evaluate Qwen3-4B-Base (Yang et al., 2025) and OctoThinker-3B-Hybrid-Base (Wang et al., 2025), covering Qwen (Yang et al., 2025) and Llama (Grattafiori et al., 2024) model families. OctoThinker uses mid-training to address the limited gains from applying GRPO directly to Llama base models (Wang et al., 2025). We compare against the base models, R-Zero, and two recent methods for stabilizing self-evolution, OCNR (Lee et al., 2026) and R-Diverse (Li et al., 2026a). Baseline descriptions and implementation details are provided in Appendix D.

Evaluation. We evaluate on seven mathematical reasoning benchmarks, three general-domain reasoning benchmarks, and two code generation benchmarks. In line with baselines (Huang et al., 2025; Lee et al., 2026), we report mean@32 on AMC and both AIME benchmarks, and greedy-decoding accuracy on the remaining reasoning benchmarks. Code generation is evaluated on HumanEval+ (Chen et al., 2021) and MBPP+ (Austin et al., 2021) using EvalPlus (Liu et al., 2023), with pass@1 as the metric. For each domain, we report the checkpoint with the highest average benchmark score across evaluated rounds. Benchmark descriptions and evaluation settings are provided in Appendix E.

Training Settings. All evaluated methods alternate GRPO updates of the questioner and solver, using a per-round budget of 10,000 candidate questions, with OCNR requiring twofold oversampling by design. The solver uses the validity-aware initialization in Section 3.1, with 10% replay from the initialization dataset each round. The validity gate uses majority voting over nine responses, while the novelty gate uses $K = 8$ within-batch comparisons judged by a frozen copy of the corresponding base model. Further training details are provided in Appendices A.3 and C.2.

Table 1: Mathematical reasoning results (%) across seven benchmarks. Avg. denotes the mean across benchmarks. For each backbone, the best score in each column is shown in **bold**.

Method	Avg.	AMC	Minerva	MATH	GSM8K	Olympiad	AIME25	AIME24
Qwen3-4B-Base								
Base model	46.17	47.34	51.84	74.20	88.63	39.11	11.35	10.73
Base model (validity init.)	47.07	49.14	53.68	74.40	89.76	41.04	10.73	10.73
R-Zero	49.10	54.84	54.78	76.20	92.42	42.37	12.29	10.83
OCNR	49.57	55.94	58.09	**77.60**	92.12	43.11	7.92	12.19
R-Diverse	49.82	55.31	57.35	77.20	92.19	43.70	9.48	13.54
R-Quest (our method)	**51.64**	**57.89**	**58.82**	**77.60**	**92.57**	**46.67**	**13.44**	**14.48**
OctoThinker-3B								
Base model	24.90	25.00	23.16	41.80	69.14	11.85	0.00	3.33
Base model (validity init.)	25.22	19.84	26.84	43.40	69.90	13.63	0.00	2.92
R-Zero	28.25	25.47	**29.78**	48.80	75.89	15.11	0.10	2.60
OCNR	28.12	30.00	28.68	49.60	73.16	15.41	0.00	0.00
R-Diverse	28.61	29.84	27.21	49.20	74.53	16.15	0.00	3.33
R-Quest (our method)	**30.69**	**32.27**	29.41	**55.80**	**76.42**	**17.19**	**0.31**	**3.44**

4.2 Main Results

R-Quest outperforms self-evolution baselines in mathematical reasoning. Consistent with evaluations in previous work (Huang et al., 2025; Li et al., 2026a), we compare all methods over five rounds of self-evolution, reporting the checkpoint with the highest mathematical average for each method. As shown in Table 1, R-Quest achieves the highest average performance across seven mathematical reasoning benchmarks on both backbones, surpassing R-Zero, OCNR, and R-Diverse. It improves over Qwen3-4B-Base by 5.47 points and over OctoThinker-3B by 5.79 points. Relative to R-Zero, R-Quest yields additional gains of 2.54 and 2.44 points. Across both backbones, validity-aware initialization yields mathematical performance close to that of the base models. Most of R-Quest's gains emerge during subsequent self-evolution, suggesting that initialization supports later learning by enabling the solver to provide validity feedback. Measured from the same base models, R-Quest delivers 1.87 and 1.73 times the gains achieved by R-Zero, respectively, increasing the benefit of self-evolution across Qwen and Llama model families.

R-Quest sustains gains over extended self-evolution. We extend self-evolution to ten rounds to assess sustained gains, as shown in Figure 5. R-Zero peaks at 49.10 in round three, then declines to 34.60 by round ten, falling below the base model's score of 46.17. In contrast, R-Quest retains its gains despite minor fluctuations between rounds: every checkpoint from rounds one through ten exceeds R-Zero's best score. R-Quest ultimately reaches its highest score of 51.92 in round ten, exceeding R-Zero by 17.32 points at the same iteration. We further find that either validity or novelty feedback alone achieves a higher peak score than R-Zero. However, validity feedback is more critical for sustained gains: its removal leads to late-stage collapse, whereas the variant without novelty feedback shows a milder decline and remains above the base model.

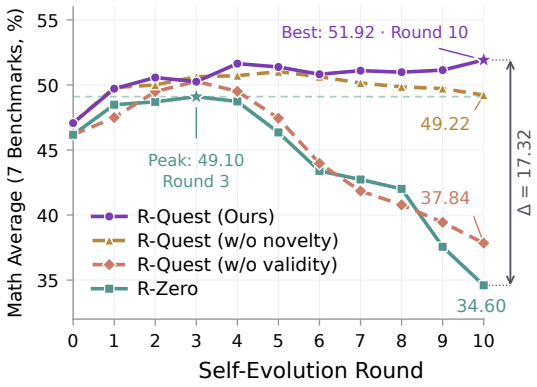


Figure 5: Average performance on seven benchmarks over ten rounds on Qwen3-4B-Base. Yellow and orange denote R-Quest without novelty or validity feedback.

Table 2: Code generation (pass@1) and general-domain reasoning results (%). Avg. denotes the mean within each domain. Best scores within each backbone are in **bold**.

Method	Code generation			General-domain reasoning			
	Avg.	HumanEval+	MBPP+	Avg.	SuperGPQA	MMLU-Pro	BBEH
Qwen3-4B-Base							
Base model	58.79	54.88	62.70	28.79	26.21	51.41	8.74
Base model (validity init.)	59.63	55.49	63.76	29.40	26.28	51.58	10.35
R-Zero	59.27	56.10	62.43	31.21	28.66	54.01	10.95
OCNR	60.28	57.32	63.23	31.56	29.05	54.82	10.82
R-Diverse	60.50	56.71	64.29	31.76	28.98	55.79	10.51
R-Quest (our method)	**62.69**	**59.76**	**65.61**	**32.75**	**29.40**	**57.36**	**11.50**
OctoThinker-3B							
Base model	13.67	14.63	12.70	8.30	10.78	12.14	1.97
Base model (validity init.)	14.77	15.24	14.29	9.04	11.42	13.53	2.17
R-Zero	14.14	15.85	12.43	14.10	13.98	22.25	6.08
OCNR	15.60	15.85	15.34	13.83	13.89	21.86	5.73
R-Diverse	14.98	16.46	13.49	14.50	14.56	22.91	6.04
R-Quest (our method)	**18.90**	**17.68**	**20.11**	**15.50**	**14.96**	**25.00**	**6.55**

R-Quest improves code generation and general-domain reasoning. Prior work has shown that mathematical reasoning training can benefit performance beyond mathematics (Huang et al., 2025). As shown in Table 2, R-Quest substantially improves code generation across both Qwen and Llama backbones. Specifically, using a mathematics-focused self-evolution curriculum, it improves average code generation scores over the base models by 3.90 points on Qwen3-4B-Base and 5.23 points on OctoThinker-3B, whereas R-Zero, OCNR, and R-Diverse yield comparatively modest gains. R-Quest also achieves the highest average performance across the three general-domain reasoning benchmarks on both backbones. Validity initialization alone yields only modest gains in both domains; most of the improvement occurs during subsequent mathematics-focused self-evolution. Reducing training on invalid questions may limit the reinforcement of solutions based on unstated assumptions, encouraging reasoning grounded in the information provided across domains. This is particularly relevant to code generation, where correct implementations must satisfy the constraints in the task.

4.3 Analysis

4.3.1 Component Ablations

Validity and novelty feedback both improve performance. As shown in Table 3, removing either validity or novelty feedback lowers performance across all three domains. We also compare with a frozen validity judge setting, with novelty feedback enabled. This variant underperforms the full R-Quest across all three domains, supporting that continuous validity-training through self-evolution is essential.

Table 3: Component ablations on Qwen3-4B-Base. Scores are averaged within each domain.

Variant	Math	Code	General
R-Quest	**51.64**	**62.69**	**32.75**
w/o novelty	51.02	61.98	31.80
w/o validity	50.27	61.28	31.53
w/o both (R-Zero)	49.10	59.27	31.21
Frozen validity judge	49.93	61.85	32.39

4.3.2 Validity Assessment During Self-Evolution

Self-evolution strengthens validity assessment in R-Quest. We evaluate each round's solver validity assessment on the same 297 questions held out from the training process, as described in Appendix A.1. Using Prompt A.5, each solver is instructed to answer valid questions and reject invalid ones, with one greedy response generated per question. We report balanced accuracy as

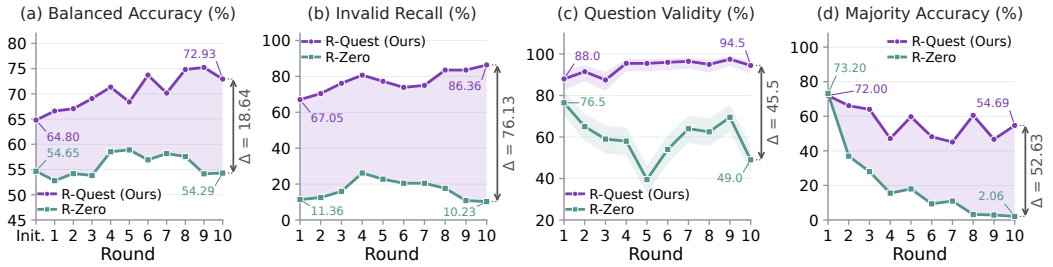


Figure 6: Validity assessment, question and answer quality during self-evolution on Qwen3-4B-Base. Panels (a) and (b) show R-Quest's improved validity assessment on held-out questions. Panels (c) and (d) show that R-Quest maintains high question validity and better preserves majority-answer accuracy on valid questions. Bands in (c) show pointwise 95% Wilson confidence intervals.

$\frac{1}{2}(\mathrm{Recall}_{\mathrm{valid}} + \mathrm{Recall}_{\mathrm{invalid}})$, where the two terms are the fractions of valid questions accepted and invalid questions rejected, respectively. R-Quest's balanced accuracy increases from 64.80% after initialization to 72.93% in round ten (Figure 6(a)). Since a solver that accepts every question achieves 50% balanced accuracy without detecting any invalid questions, we additionally report invalid recall to assess this capability directly. Figure 6(b) shows that R-Quest achieves an invalid recall of 86.36% in round ten, whereas recall for R-Zero remains between 10.23% and 26.14% across all ten rounds. Thus, the validity assessment capability of R-Quest continues to improve through self-evolution.

R-Quest generates more valid questions. We next examine whether stronger solver validity assessment is accompanied by improved question quality. For each method and round, we randomly sample 200 questions from the unfiltered pool generated by their questioner. We query GPT-5.6-Sol to assess each question's validity and compute the validity rate. Figure 6(c) shows that R-Quest increases this rate from 88.0% in round one to 94.5–97.5% throughout rounds four to ten. In contrast, the rate for R-Zero falls from 76.5% to 49.0%. This indicates that R-Quest steers question generation toward valid problems as self-evolution proceeds.

R-Quest solver derives more reliable answers. Question generation and answer supervision are coupled across rounds: generated questions shape solver training, and the updated solver supplies pseudo-labels for subsequent rounds. Among the 200 questions sampled for each method and round, we evaluate majority-answer accuracy only on those judged valid, since invalid questions do not provide well-defined mathematical targets for solver training. We query GPT-5.6-Sol independently to derive and verify reference answers, and compare the majority-voted answers with these references. We report strict accuracy over questions with verified references, giving no credit to missing answers or unresolved judgments. Figure 6(d) shows that the two methods begin with comparable accuracy, but by round ten, R-Quest retains 54.69% accuracy, whereas R-Zero falls to just 2.06%. This shows that the deterioration in R-Zero extends from question generation to answer supervision. By maintaining a predominantly valid question distribution and more reliable pseudo-labels, R-Quest preserves the quality of self-generated supervision that supports sustainable self-evolution.

4.3.3 Question Diversity During Self-Evolution

R-Quest limits the concentration of recurring exercise patterns. Following the analysis in Section 2.2.2, we uniformly sample 200 solver training question records without replacement for each method and round over the first five rounds. GPT-5.6-Sol sequentially groups them by their underlying mathematical construction and requested task. We measure concentration using the top-five share, the fraction of questions in the five largest groups, averaged over three random orderings of the same sample. Appendix B.3 provides the grouping procedure and task specification. As shown in Figure 7, the top-five share for R-Zero rises from 17.0% to 69.0% between rounds one and five. R-Quest with the default $K = 8$ remains less concentrated throughout, reaching 24.0% in round five while maintaining higher performance.

Overly strict diversity control can limit mathematical gains. We further examine the strength of novelty control in Figure 7 by increasing K from its default value of 8 to 16, with other settings unchanged. A larger K increases the probability that questions from recurring exercise patterns are rejected by the novelty gate (Appendix Figure 8). This further lowers the top-five share, most notably from 30.0% to 16.8% in round four. However, performance remains below that of $K = 8$ throughout, suggesting that overly strict diversity control may suppress more challenging variants within recurring exercise patterns. Effective novelty control should therefore balance exploration of new exercise patterns with progress toward harder problems within existing ones.

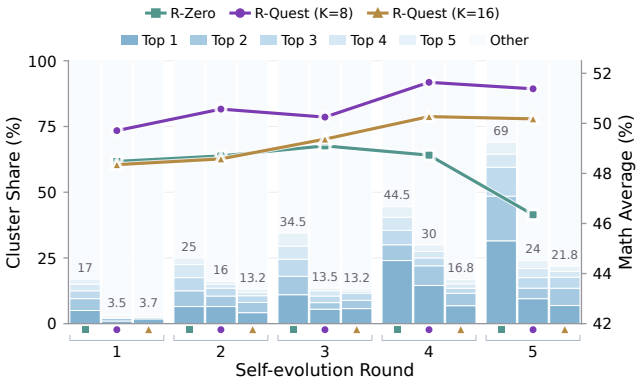


Figure 7: Question type clusters (bars, left axis) and mathematical performance (lines, right axis) on Qwen3-4B-Base. Numbers above the bars denote top-five group proportion.

5 Related Work

Self-improving Reasoning Models. Label-free reinforcement learning uses model confidence, answer agreement, and entropy as training signals (Zhao et al., 2026b; Zuo et al., 2026; Agarwal et al., 2026). Complementary work learns to generate training tasks through self-play. Absolute Zero validates tasks and solutions through code execution (Zhao et al., 2026a), while R-Zero (Huang et al., 2025) and Self-Questioning Language Models (Chen et al., 2026) train question generators and solvers using model-derived feedback. Related frameworks ground learning in games, document corpora, vision tasks, and tool environments (Liu et al., 2026; 2025; Yue et al., 2026; Huang et al., 2026c;b; Li et al., 2026b; He et al., 2025). Our work examines the conditions under which self-generated questions support sustained improvement, focusing on question validity and task novelty.

Question Validity and Diversity in Self-evolution. For validity, prior work studies question validity in synthetic mathematical data (Liang et al., 2026; Shen et al., 2026; Yang et al., 2026). For self-evolution, VHG (Lai et al., 2026) relies on a stronger external model (GPT-5.4) to verify general mathematical problems during self-play. We teach the solver to reject invalid questions through mid-training, avoiding calls to stronger external models during subsequent self-evolution. For diversity, recent methods assess novelty using distances between solver representations or code-embedding similarity (Lee et al., 2026; Li et al., 2026a). R-Diverse targets repetition in underlying reasoning procedures, but relies on an additional code generator (Qwen2.5-Coder-7B (Hui et al., 2024)) and code encoder (Jina-Code-Embeddings-1.5B (Kryvosheieva et al., 2025)) during self-evolution. This departs from the goal of self-contained self-evolution, where the model generates its own training feedback. We instead reduce novelty assessment to an atomic judgment of whether two questions are variations of the same mathematical task, allowing a frozen copy of the same base model to provide novelty feedback with only a few random within-batch comparisons.

6 Conclusion

We identified invalid questions and repeated mathematical tasks as two bottlenecks to sustained self-evolution. R-Quest addresses these bottlenecks by enabling the solver to recognize and reject invalid questions and using validity and task-level novelty feedback to guide question generation. Across two model families and 12 benchmarks, R-Quest achieves the highest domain-average scores among evaluated methods in mathematics, code generation, and general reasoning. It further sustains gains over ten rounds on Qwen3-4B-Base, reaching its best mathematical performance in the final round. Further analyses show that R-Quest strengthens validity assessment, limits task repetition, and maintains more reliable majority-vote supervision across rounds. These findings highlight the ability to assess self-generated questions as a key component of sustained self-improvement.

AI USE STATEMENT

Generative AI was used in the research workflow for synthetic data generation, model-based annotation and verification, and evaluation, and is also integral to the proposed self-evolution method as described in the paper. It was additionally used for limited language editing. AI-assisted research outputs were reviewed using the validation and consistency-checking procedures described in the paper. The authors reviewed the final manuscript and take responsibility for its contents.

REPRODUCIBILITY STATEMENT

We provide the information required to reproduce the main training and evaluation pipeline in the paper and appendices. Appendix A describes construction of the validity-aware initialization dataset, the annotation and reference-answer review procedures, the complete prompts, reward function, and mid-training hyperparameters. Appendix B specifies the semantic novelty judgment and sampling procedure. Appendix C provides the question-generation and problem-solving prompts, question-evaluation procedure, solver-data construction, replay strategy, sampling parameters, and GRPO training settings. Appendix D documents the implementations and configurations of the R-Zero, OCNR, and R-Diverse baselines, and Appendix E specifies the evaluation benchmarks, decoding procedures, answer judging, and reported metrics.

REFERENCES

Shivam Agarwal, Zimin Zhang, Lifan Yuan, Jiawei Han, and Hao Peng. The unreasonable effectiveness of entropy minimization in llm reasoning. *Advances in Neural Information Processing Systems*, 38:107150–107180, 2026.

Jacob Austin, Augustus Odena, Maxwell Nye, Maarten Bosma, Henryk Michalewski, David Dohan, Ellen Jiang, Carrie Cai, Michael Terry, Quoc Le, et al. Program synthesis with large language models. *arXiv preprint arXiv:2108.07732*, 2021.

Lili Chen, Mihir Prabhudesai, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. Self-questioning language models. In *Workshop on Multi-Agent Learning and Its Opportunities in the Era of Generative AI*, 2026. URL https://openreview.net/forum?id=8OOlhDmpXL.

Mark Chen, Jerry Tworek, Heewoo Jun, Qiming Yuan, Henrique Ponde De Oliveira Pinto, Jared Kaplan, Harri Edwards, Yuri Burda, Nicholas Joseph, Greg Brockman, et al. Evaluating large language models trained on code. *arXiv preprint arXiv:2107.03374*, 2021.

Karl Cobbe, Vineet Kosaraju, Mohammad Bavarian, Mark Chen, Heewoo Jun, Lukasz Kaiser, Matthias Plappert, Jerry Tworek, Jacob Hilton, Reiichiro Nakano, et al. Training verifiers to solve math word problems. *arXiv preprint arXiv:2110.14168*, 2021.

Xeron Du, Yifan Yao, Kaijing Ma, Bingli Wang, Tianyu Zheng, Minghao Liu, Yiming Liang, Xiaolong Jin, Zhenlin Wei, Chujie Zheng, et al. Supergpqa: Scaling llm evaluation across 285 graduate disciplines. *Advances in Neural Information Processing Systems*, 38, 2026.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, Jie Liu, Lei Qi, Zhiyuan Liu, and Maosong Sun. OlympiadBench: A challenging benchmark for promoting AGI with olympiad-level bilingual multimodal scientific problems. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.211. URL https://aclanthology.org/2024.acl-long.211/.

Yicheng He, Chengsong Huang, Zongxia Li, Jiaxin Huang, and Yonghui Yang. Visplay: Self-evolving vision-language models from images. *arXiv preprint arXiv:2511.15661*, 2025.

Dan Hendrycks, Collin Burns, Saurav Kadavath, Akul Arora, Steven Basart, Eric Tang, Dawn Song, and Jacob Steinhardt. Measuring mathematical problem solving with the MATH dataset. In *Thirty-fifth Conference on Neural Information Processing Systems Datasets and Benchmarks Track (Round 2)*, 2021. URL https://openreview.net/forum?id=7Bywt2mQsCe.

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

Chengsong Huang, Haolin Liu, Tong Zheng, Runpeng Dai, Langlin Huang, Jinyuan Li, Zongxia Li, Zhepei Wei, Yu Meng, and Jiaxin Huang. G-zero: Self-play for open-ended generation from zero data. *arXiv preprint arXiv:2605.09959*, 2026a.

Chengsong Huang, Zifeng Wang, Rujun Han, Jun Yan, Yanfei Chen, Zoey CuiZhu, Ke Jiang, Peng Xia, Han Yu, Yufan Zhuang, et al. Envharness: Awakening static worlds for agent learning. *arXiv preprint arXiv:2608.19880*, 2026b.

Jiaxin Huang, Shixiang Gu, Le Hou, Yuexin Wu, Xuezhi Wang, Hongkun Yu, and Jiawei Han. Large language models can self-improve. In *Proceedings of the 2023 Conference on Empirical Methods in Natural Language Processing*, pp. 1051–1068, December 2023. doi: 10.18653/v1/2023.emnlp-main.67. URL https://aclanthology.org/2023.emnlp-main.67/.

Siyuan Huang, Pengyu Cheng, Haotian Liu, Tao Chen, Yihao Liu, Jingwei Ni, Shijie Zhou, Ziyi Yang, Gangwei Jiang, Mengyu Zhou, et al. Skill self-play: Pushing the frontier of llm capability with co-evolving skills. *arXiv preprint arXiv:2607.22529*, 2026c.

Binyuan Hui, Jian Yang, Zeyu Cui, Jiaxi Yang, Dayiheng Liu, Lei Zhang, Tianyu Liu, Jiajun Zhang, Bowen Yu, Keming Lu, et al. Qwen2. 5-coder technical report. *arXiv preprint arXiv:2409.12186*, 2024.

Mehran Kazemi, Bahare Fatemi, Hritik Bansal, John Palowitch, Chrysovalantis Anastasiou, Sanket Vaibhav Mehta, Lalit K Jain, Virginia Aglietti, Disha Jindal, Yuanzhu Peter Chen, et al. Big-bench extra hard. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 26473–26501, 2025.

Daria Kryvosheieva, Saba Sturua, Michael Günther, Scott Martens, and Han Xiao. Efficient code embeddings from code generation models. In *NeurIPS 2025 Fourth Workshop on Deep Learning for Code*, 2025. URL https://openreview.net/forum?id=smR7RPdWZs.

Yuhang Lai, Jiazhan Feng, Yee Whye Teh, and Ning Miao. Verifier-backed hard problem generation for mathematical reasoning. *arXiv preprint arXiv:2605.06660*, 2026.

Seungyoo Lee, Giung Nam, Hyungi Lee, and Juho Lee. OCNR: Stabilizing self-play by mitigating iteration-collapse with one-class novelty rewards. In *Forty-third International Conference on Machine Learning*, 2026. URL https://openreview.net/forum?id=70lhVsTZ62.

Aitor Lewkowycz, Anders Andreassen, David Dohan, Ethan Dyer, Henryk Michalewski, Vinay Ramasesh, Ambrose Slone, Cem Anil, Imanol Schlag, Theo Gutman-Solo, et al. Solving quantitative reasoning problems with language models. *Advances in neural information processing systems*, 35:3843–3857, 2022.

Gengsheng Li, Jinghan He, Shijie Wang, Ruiqi Liu, Renrui Zhang, Zijun Yao, Junfeng Fang, Haiyun Guo, Dan Zhang, and Jinqiao Wang. R-diverse: Mitigating diversity illusion in self-play LLM training. In *Forty-third International Conference on Machine Learning*, 2026a. URL https://openreview.net/forum?id=DZiuKVvrJW.

Zongxia Li, Hongyang Du, Chengsong Huang, Xiyang Wu, Lantao Yu, Yicheng He, Jing Xie, Xiaomin Wu, Zhichao Liu, Jiarui Zhang, et al. Mm-zero: Self-evolving multi-model vision language models from zero data. *arXiv preprint arXiv:2603.09206*, 2026b.

Hao Liang, Meiyi Qiang, Yuying Li, Zefeng He, Xiaochen Ma, Ruichuan An, Yongzhen Guo, Zhengzhou Zhu, Bin Cui, and Wentao Zhang. Mathdebugger: Detecting and diagnosing errors in synthetic mathematical data, 2026. URL https://arxiv.org/abs/2502.19058.

Bo Liu, Chuanyang Jin, Seungone Kim, Weizhe Yuan, Wenting Zhao, Ilia Kulikov, Xian Li, Sainbayar Sukhbaatar, Jack Lanchantin, and Jason Weston. Spice: Self-play in corpus environments improves reasoning. *arXiv preprint arXiv:2510.24684*, 2025.

Bo Liu, Simon Yu, Zichen Liu, Leon Guertler, Penghui Qi, Daniel Balcells, Mickel Liu, Cheston Tan, Weiyan Shi, Min Lin, et al. Spiral: Self-play on zero-sum games incentivizes reasoning via multi-agent multi-turn reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 35407–35434, 2026.

Jiawei Liu, Chunqiu Steven Xia, Yuyao Wang, and Lingming Zhang. Is your code generated by chatgpt really correct? rigorous evaluation of large language models for code generation. *Advances in neural information processing systems*, 36:21558–21572, 2023.

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.

Chengyu Shen, Zhen Hao Wong, Runming He, Hao Liang, Meiyi Qiang, Zimo Meng, Zhengyang Zhao, Bohan Zeng, Zhengzhou Zhu, Bin Cui, et al. Let's verify math questions step by step. In *Proceedings of the 32nd ACM SIGKDD Conference on Knowledge Discovery and Data Mining V. 1*, pp. 2770–2781, 2026.

Yucheng Shi, Zhenwen Liang, Kishan Panaganti, Dian Yu, Wenhao Yu, and Haitao Mi. Learning to build the environment: Self-evolving reasoning rl via verifiable environment synthesis. *arXiv preprint arXiv:2605.14392*, 2026.

Zhen Tan, Dawei Li, Song Wang, Alimohammad Beigi, Bohan Jiang, Amrita Bhattacharjee, Mansooreh Karami, Jundong Li, Lu Cheng, and Huan Liu. Large language models for data annotation and synthesis: A survey. In *Proceedings of the 2024 conference on empirical methods in natural language processing*, pp. 930–957, 2024.

Zhengwei Tao, Ting-En Lin, Xiancai Chen, Hangyu Li, Yuchuan Wu, Yongbin Li, Zhi Jin, Fei Huang, Dacheng Tao, and Jingren Zhou. A survey on self-evolution of large language models. *arXiv preprint arXiv:2404.14387*, 2024.

Yubo Wang, Xueguang Ma, Ge Zhang, Yuansheng Ni, Abhranil Chandra, Shiguang Guo, Weiming Ren, Aaran Arulraj, Xuan He, Ziyan Jiang, et al. Mmlu-pro: A more robust and challenging multi-task language understanding benchmark. *Advances in Neural Information Processing Systems*, 37: 95266–95290, 2024.

Zengzhi Wang, Fan Zhou, Xuefeng Li, and Pengfei Liu. Octothinker: Mid-training incentivizes reinforcement learning scaling. *arXiv preprint arXiv:2506.20512*, 2025.

Zhishang Xiang, Chengyi Yang, Zerui Chen, Zhimin Wei, Yunbo Tang, Zongpei Teng, Zexi Peng, Zongxia Li, Chengsong Huang, Yicheng He, Chang Yang, Xinrun Wang, Xiao Huang, Qinggang Zhang, and Jinsong Su. A systematic survey of self-evolving agents: From model-centric to environment-driven co-evolution. *TechRxiv*, 2026(0227), 2026. doi: 10.36227/techrxiv.177203250.05832634/v2. URL https://www.techrxiv.org/doi/abs/10.36227/techrxiv.177203250.05832634/v2.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Chengyi Yang, Zhishang Xiang, Yunbo Tang, Zongpei Teng, Chengsong Huang, Fei Long, Yuhan Liu, and Jinsong Su. Ttcs: Test-time curriculum synthesis for self-evolving. *arXiv preprint arXiv:2601.22628*, 2026.

Wenhao Yu, Zhenwen Liang, Chengsong Huang, Kishan Panaganti, Tianqing Fang, Haitao Mi, and Dong Yu. Guided self-evolving llms with minimal human supervision. *arXiv preprint arXiv:2512.02472*, 2025.

Zhenrui Yue, Kartikeya Upasani, Xianjun Yang, Suyu Ge, Shaoliang Nie, Yuning Mao, Zhe Liu, and Dong Wang. Dr. zero: Self-evolving search agents without training data. *arXiv preprint arXiv:2601.07055*, 2026.

Andrew Zhao, Yiran Wu, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data. *Advances in Neural Information Processing Systems*, 38:105816–105879, 2026a.

Xuandong Zhao, Zhewei Kang, Aosong Feng, Sergey Levine, and Dawn Song. Learning to reason without external rewards. In *International Conference on Learning Representations*, volume 2026, pp. 2548–2581, 2026b.

Yuxin Zuo, Kaiyan Zhang, Li Sheng, Shang Qu, Ganqu Cui, Xuekai Zhu, Haozhan Li, Xinwei Long, Ermo Hua, Biqing Qi, et al. Ttrl: Test-time reinforcement learning. *Advances in Neural Information Processing Systems*, 38:131459–131483, 2026.

A Validity-Aware Mid-Training Details

A.1 Dataset Construction

We collect questions from the solver training sets of the first five R-Zero (Huang et al., 2025) rounds using Qwen3-4B-Base (Yang et al., 2025). After deduplication within and across iterations, we randomly sample 460 questions per iteration, assigning 400 to training and 60 to validation. Annotation and answer review yield a final dataset of 2,240 questions, with statistics reported in Table 4. Table 5 reports the distribution of error types among invalid questions.

Table 4: Dataset statistics for validity-aware solver initialization.

Split	Valid	Invalid	Total
Training	802	1,141	1,943
Validation	121	176	297
Total	923	1,317	2,240

Table 5: Error types among the 1,317 invalid questions in the training and validation splits.

Error type	Count	Percentage
Contradiction	380	28.9%
No solution	362	27.5%
Ambiguity	315	23.9%
Missing information	181	13.7%
Poorly formed questions	72	5.5%
Other	7	0.5%
Total	1,317	100%

A.2 Annotation and Answer Review

We use `gpt-5.6-Sol` to annotate the questions in two stages. Both stages use only the question, without the original R-Zero pseudo-label. First, the model attempts to solve each question and assesses whether it is valid using Prompt A.1. Category A is mapped to `VALID`, and B–F to `INVALID`.

Prompt A.1: Validity annotation

You are a careful reference judge for generated mathematical and reasoning questions. You see only the question. Restate its goal, inspect every definition and condition, check for contradictions and multiple reasonable interpretations, and genuinely attempt a solution. Then classify:

A = self-contained, valid, solvable, and uniquely gradable;
B = meaningful but open-ended or not precisely gradable;
C = missing conditions, multiple answers, or key ambiguity;
D = contradictory or has no solution;
E = malformed, undefined, garbled, or unintelligible;
F = you cannot judge reliably.

Use A only when all strict requirements hold. A question can have multiple solutions and still be A when it explicitly asks for the complete solution set and that set is objectively gradable. Judge only the question as written. Keep the reasoning summary concise but concrete. For A, `invalid_type` must be `null`; for B-F choose the single closest `invalid_type`.

Second, a separate call independently solves the valid questions to produce reference answers using Prompt A.2.

Prompt A.2: Reference answers

You are the answer-verification pass for a mathematical reasoning dataset. The validity pass has already classified this question as A (valid). Solve it completely from scratch.
Do not copy or trust an answer hint that may appear in the question. Check every stated condition, substitute or otherwise verify the result, and confirm that the final answer answers exactly what was asked.

Prompt A.2: Reference answers (continued)

`canonical_final_answer` must be a concise answer suitable for exact or mathematical-equivalence rewarding, without reasoning or a boxed wrapper.
Set `answer_verified=true` only when you have high confidence that the answer is correct and complete. If reliable verification is not possible, set it `false` and explain the uncertainty; never invent an answer to satisfy the schema.

We then review the reference answers and exclude questions whose answers cannot be reliably verified. Prompts A.3 and A.4 provide the review instructions, with the latter used for questions requiring further review. Questions flagged for `HUMAN_REVIEW` were manually reviewed and retained only when their validity and reference answers could be verified; unresolved cases were excluded.

Prompt A.3: Answer review

You are an independent mathematical answer consistency auditor. You receive an opaque question `ID`, the question, a Pass 1 derived answer (which may be missing), and a Pass 2 canonical answer. Neither answer is authoritative.
Read the question carefully and identify exactly what it asks. Independently do the mathematical checks needed to decide whether the canonical answer is correct, complete, and responsive. Decide whether an ambiguity in the question could change the answer. Compare the two supplied answers by mathematical meaning, not merely by string equality: different strings can be equivalent, and equal strings are not proof of correctness. If reliable verification is not possible, say so explicitly. Keep `reasoning_summary` concise and concrete; do not provide a long proof.
Relation meanings:

- `AGREE`: same answer and meaning.
- `EQUIVALENT`: different presentation but mathematically equivalent.
- `CONFLICT`: incompatible answers.
- `NOT_COMPARABLE`: their forms or scopes do not permit a reliable direct comparison.
- `MISSING`: the derived answer is absent.

Set `needs_deep_review` whenever there is a correctness concern, material ambiguity, conflict, or insufficient basis for a high-confidence screening decision.

Prompt A.4: Further answer review

You are the independent deep reviewer for a mathematical answer dataset. You receive only an opaque question `ID`, the question, a prior derived answer (possibly missing), and a canonical answer (possibly missing). None is authoritative.
Solve the question independently from first principles. Verify what the question actually asks, whether it is clear and valid, and whether your independently derived answer is complete and responsive. Then assess the canonical answer and compare your answer separately with the derived and canonical answers by mathematical meaning, not string equality. Do not assume agreement means correctness. If ambiguity can change the answer, or reliable verification is not possible, say so.
Recommend `KEEP_CANONICAL` only when the canonical answer is independently confirmed. Recommend `REPLACE_CANONICAL` only when the canonical answer is wrong or missing and your replacement is reliably verified. Recommend `EXCLUDE` for a materially ambiguous or invalid question. Recommend `HUMAN_REVIEW` for unresolved conflicts or uncertainty. Keep `reasoning_summary` concise but include the decisive mathematical check. When recommending `REPLACE_CANONICAL`, set `replacement_answer` to exactly the `independently_derived_answer`; otherwise set `replacement_answer` to `null`.

A.3 Reward and Training Settings

We train Qwen3-4B-Base (Yang et al., 2025) and OctoThinker-3B-Hybrid-Base (Wang et al., 2025) with GRPO to return a mathematical answer for a valid question or `INVALID` for an invalid one, using Prompt A.5.

Table 6: Training settings for validity-aware solver initialization.

Setting	Value
Training steps	15
Questions per rollout batch	512
Responses per question	8
Actor global batch size	128
Learning rate	10^{-6}
KL coefficient	10^{-2}
Sampling temperature	1.0
Top-p	0.99
Maximum prompt tokens	2,048
Maximum response tokens	4,096

Prompt A.5: Solver training

Question: *[Question]*
Solve the problem carefully and reason step by step. While solving, check whether the problem itself is well-defined, internally consistent, and solvable as stated.
If the problem is valid, solve it normally. If, while solving, you determine that the problem itself is invalid because of missing information, contradiction, ambiguity, impossibility, malformed conditions, or lack of objective gradability, use `INVALID` as the final answer.
Do not output `INVALID` merely because the problem is difficult, because you are uncertain, or because you fail to solve it.
First give your reasoning enclosed within `<think>` and `</think>` tags. Then put exactly one final answer inside `\boxed{}`. For a valid problem, that box must contain the mathematical answer; for an invalid problem, it must contain `INVALID`.

Rewards are computed from the final output in `\boxed{}`. Rejection means returning `INVALID` as the final answer. Mathematical correctness is evaluated against the reviewed reference answer using the `mathruler` equivalence grader used in R-Zero. For responses with a non-empty final answer box, we apply the reward in Section 3.1 with $\lambda = 0.5$:

$$r = \begin{cases} 1, & \text{correct answer or correct rejection,} \\ -0.5, & \text{rejection of a valid question,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

Responses with a missing or empty answer box receive a reward of -0.1.

We sample eight responses per question and compute GRPO advantages within each group, with a KL penalty relative to the base model. The checkpoint at step 15 initializes the solver for self-evolution. Table 6 summarizes the training settings.

B SEMANTIC NOVELTY ASSESSMENT

B.1 TASK SPECIFICATION

The frozen base model compares a candidate question (Question A) with a sampled reference (Question B) using the task specification in Prompt B.1.

Prompt B.1: Semantic comparison

You are judging whether two generated math problems are repetitions of the same recurring exercise pattern.
Choose SAME_TYPE when the two problems have essentially the same distinctive mathematical setup and ask the same kind of task, so that they feel like variations of the same exercise.
Differences in constants, coefficients, variables, formulas, bounds, or other local details do not by themselves make the problems different.
Do not choose SAME_TYPE merely because the problems share a broad topic, similar wording, presentation style.
If the common pattern is not clear and specific, choose DIFFERENT.
Briefly compare the exercise pattern of the two problems, then end with exactly
`\boxed{SAME_TYPE}`
or
`\boxed{DIFFERENT}`.
Question A:
`{question_a}`
Question B:
`{question_b}`
Analysis:

B.2 EFFECT OF RANDOM SAMPLING

Consider a candidate among the B successfully parsed questions in a Questioner rollout batch. Each batch initially contains 2,048 generated candidates. Among the other $B-1$ questions, let m be classified as the same exercise type by the judge, with $p = m/(B-1)$. The candidate passes the novelty check only if none of its K references is classified as the same type. Uniform sampling without replacement gives

$$P_{\text{pass}}(B, m, K) = \frac{\binom{B-1-m}{K}}{\binom{B-1}{K}}, \qquad K \le B-1, \tag{2}$$

where $\binom{n}{K} = 0$ for $n < K$. When K is small relative to B, the same-type fraction changes little across successive draws, yielding

$$P_{\text{pass}}(B, m, K) \approx (1-p)^K. \tag{3}$$

Figure 8 compares the default $K = 8$ with the additional $K = 16$ setting. Increasing the comparison budget makes same-type matches more likely, raising the probability of rejection. For example, when a recurring exercise pattern accounts for 5% of the other candidates in the batch, increasing K from 8 to 16 lowers the approximate pass probability from 66.3% to 44.0%. For a more frequent pattern accounting for 20% of the other candidates, the pass probability drops from 16.8% to just 2.8%.

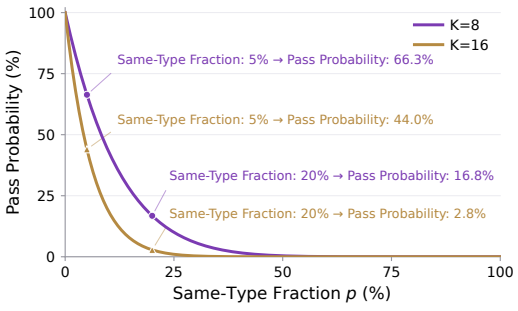


Figure 8: Approximate pass probability $(1-p)^K$ for the default comparison budget $K = 8$ and the additional setting $K = 16$.

Thus, $K = 8$ already provides substantial suppression: candidates from a pattern with a 20% share are rejected about five sixths of the time. Increasing K to 16 raises this rejection probability to 97.2%, leaving very few candidates of that type eligible for the uncertainty reward. This

helps interpret the results in Section 4.3.3, where $K = 16$ lowers the top-five share but yields worse mathematical performance than $K = 8$. Overly strict novelty control may suppress not only repeated exercises but also more challenging variants within the same pattern, limiting opportunities for further learning. The default budget therefore balances suppression of recurring patterns with room for continued exploration within them.

B.3 Exercise-Pattern Concentration Analysis

We analyze the sampled training records without additional filtering or deduplication. Only question text is supplied to the evaluator; method names, round indices, solver responses, and group sizes are withheld. All methods and rounds use the same evaluator and prompt.

For each ordering, grouping starts with an empty inventory. Questions are processed one at a time using Prompt B.2. Each existing group is represented by a short pattern description and the question that established it. A question either joins an existing group or establishes a new group with its own description and representative. Existing descriptions and representatives remain fixed throughout the pass. Each question is assigned to exactly one group.

We repeat this procedure from an empty inventory under three random orderings of the same 200 records. For each pass, we sum the sizes of the five largest groups and divide by 200, or sum all groups if fewer than five are formed. We report the mean top-five share across the three orderings.

Prompt B.2: Incremental exercise-pattern grouping

You are grouping mathematical questions by recurring exercise pattern. Given a new question and an inventory of existing groups, assign it to an existing group or establish a new one.

Exercise-pattern definition
Questions share a pattern when they have essentially the same distinctive mathematical setup and ask for the same kind of task, making them variations of the same exercise.
Differences in wording, notation, numerical values, or other local details do not by themselves establish a new pattern.
Changes to defining conditions or the requested task warrant a separate group when they change the underlying exercise.
A shared broad topic or generic solution technique is not sufficient.

Assignment
Compare the new question with the description and representative question of each existing group.
Choose the best-matching group only when the shared pattern is clear and specific, without broadening the group's definition.
Otherwise, create a new group and briefly describe its mathematical setup and requested task. If the inventory is empty, create a new group.
Classify the task as stated rather than repairing it. Apparent flaws alone are not a shared exercise pattern.

Output
Return a JSON object with the following fields:
`group_id`: the matching group ID, or `NEW`.
`pattern`: a concise description for a new group; otherwise, an empty string.
`reason`: a brief explanation of the match or the need for a new group.

Existing groups
Each entry contains an ID, pattern description, and representative question.
`{group_inventory}`

New question
`{question}`

C Self-Evolution Settings

C.1 Task Specifications

We use the question-generation and mathematical problem-solving instructions from R-Zero (Huang et al., 2025), provided in Prompts C.1 and C.2, respectively. Validity checks use Prompt A.5, and semantic comparisons use Prompt B.1.

Prompt C.1: Question generation

System
You are an expert competition-math problem setter.
FIRST, in your private scratch-pad, think step-by-step to design a brand-new, non-trivial problem. The problem could come from any field of mathematics, including but not limited to algebra, geometry, number theory, combinatorics, prealgebra, probability, statistics, and calculus. Aim for a difficulty such that fewer than 30 % of advanced high-school students could solve it. Avoid re-using textbook clichés or famous contest problems.
THEN, without revealing any of your private thoughts, output **exactly** the following two blocks:
<question>
{The full problem statement on one or more lines}
</question>
\boxed{final_answer}
Do NOT output anything else—no explanations, no extra markup.
User
Generate one new, challenging reasoning question now. Remember to format the output exactly as instructed.

Prompt C.2: Mathematical problem solving

System
Please reason step by step, and put your final answer within \boxed{}.
User
[Question]

C.2 Sampling and Training Settings

Questioner Rewards. To evaluate each generated question, the frozen solver first produces nine validity responses. A question is rejected if more than half of the outputs are INVALID. For questions that pass, we sample ten fresh mathematical solutions to compute answer consistency. These signals, together with the novelty judgment, determine the questioner reward defined in Section 3.3. Questions that fail the required output format receive a reward of -1.

Solver Data Construction. After updating the questioner, we generate candidate training questions and apply the same evaluation procedure: nine validity responses, followed by ten mathematical solutions for questions that pass. We retain questions with answer consistency $s(q) \in [0.3, 0.8]$ and a non-empty majority-vote answer, which serves as the pseudo-label.

Solver Training and Replay. Replay examples are sampled from the training split of the mid-training dataset to constitute 10% of the mixed solver dataset. They retain the validity-aware task and reward from Appendix A.3. On newly generated questions, the solver uses the standard mathematical problem-solving task and the R-Zero training reward: 0.9 for agreement with the pseudo-label under mathematical equivalence checking, plus 0.1 for placing reasoning inside <think> and </think> tags followed by a boxed answer.

Shared Settings. For question evaluation and GRPO rollouts, we use a sampling temperature of 1.0 and a maximum response length of 4,096 tokens. Both the questioner and solver are trained with a learning rate of 10^{-6}, a KL coefficient of 10^{-2}, and a maximum prompt length of 2,048 tokens. Table 7 reports question evaluation settings, while Table 8 reports GRPO training settings. Responses per input in the latter refer to training rollouts.

Table 7: Question evaluation settings.

Setting	Value
Validity responses per question	9
Math solutions per accepted question	10
Top-p	1.0
Top-k	40
Consistency filter for solver data	$[0.3, 0.8]$

Table 8: GRPO training settings.

Setting	Questioner	Solver
Steps per iteration	5	15
Inputs per rollout batch	512	512
Responses per input	4	5
Actor global batch size	4	128
Top-p	0.99	0.99

D BASELINE DETAILS

Base Models. We evaluate Qwen3-4B-Base (Yang et al., 2025) and OctoThinker-3B (Wang et al., 2025) without additional training to establish their performance before self-evolution.

R-Zero. We use the released implementation of R-Zero (Huang et al., 2025), which alternates reinforcement learning updates of a questioner and a solver. The questioner reward combines solver uncertainty with a BLEU-based penalty for within-batch repetition. Generated questions are assigned pseudo-labels through majority voting and used for subsequent solver training.

OCNR. We implement OCNR (Lee et al., 2026) from its published description within the R-Zero training framework. OCNR augments the questioner reward with a novelty score from a one-class detector trained to distinguish questions used for solver training from questions withheld from training. Following OCNR's twofold overgeneration scheme, we generate 20,000 candidate questions per round, twice our standard budget. The detector maintains one learnable center in the solver representation space. Using values reported in the paper, we set the novelty reward coefficient to 0.3, the distance temperature $\tau = 0.5$, and the EMA update weight $\alpha = 0.05$. Question representations are obtained by mean-pooling the solver's final-layer hidden states. The detector receives one full-batch SGD update per round with a learning rate of 7×10^{-3}.

R-Diverse. We implement R-Diverse (Li et al., 2026a) from its published description within the R-Zero training framework. It penalizes repetition based on how questions are solved, both within the current batch and across rounds. Following the published configuration, we use Qwen2.5-Coder-7B (Hui et al., 2024) with greedy decoding to represent questions as solution programs and Jina-Code-Embeddings-1.5B (Kryvosheieva et al., 2025) to measure their similarity within the current batch and against earlier questions. For within-batch repetition, we retain R-Zero's average-linkage clustering with a cosine-distance threshold of 0.5. Following the paper, we set the coefficients of the within-batch and cross-iteration repetition penalties to 1. The memory penalty combines maximum and mean cosine similarities with equal weights and thresholds of 0.5 and 0.25, respectively. From the second round onward, historical examples are uniformly sampled to constitute 30% of the solver training data.

E EVALUATION BENCHMARKS

Although our self-evolution process focuses on mathematical questions, we evaluate both mathematical reasoning and its generalization to other domains. Our evaluation covers mathematical reasoning, general-domain reasoning, and code generation.

Mathematical Reasoning. We evaluate on seven benchmarks: AMC, Minerva (Lewkowycz et al., 2022), MATH-500 (Hendrycks et al., 2021), GSM8K (Cobbe et al., 2021), OlympiadBench (He et al., 2024), AIME24, and AIME25. We use Qwen3-32B (Yang et al., 2025) to judge the equivalence of model answers and reference answers. Consistent with R-Zero (Huang et al., 2025), we report mean@32 on AMC and both AIME benchmarks, and accuracy under greedy decoding on the remaining benchmarks.

General Domain Reasoning. We evaluate generalization beyond mathematics on three benchmarks:

- **MMLU-Pro** (Wang et al., 2024): Multiple-choice questions across 14 academic domains, requiring both subject knowledge and reasoning.
- **SuperGPQA** (Du et al., 2026): Graduate-level multiple-choice questions across 285 disciplines, assessing specialized knowledge and reasoning.
- **BBEH** (Kazemi et al., 2025): Challenging tasks covering logical deduction, spatial reasoning, and multi-step computation.

We follow the R-Zero evaluation protocol and report exact-match accuracy under greedy decoding.

Code Generation. We evaluate code generation on two Python programming benchmarks:

- **HumanEval+** (Chen et al., 2021; Liu et al., 2023): Completing Python functions given their signatures and natural-language specifications.
- **MBPP+** (Austin et al., 2021; Liu et al., 2023): Generating short Python programs from natural-language descriptions of basic programming tasks.

We use greedy decoding and report pass@1 on the full EvalPlus test suites.

Table 9: Per-benchmark results for the gate ablations on Qwen3-4B-Base.

Variant	Mathematical reasoning							
	Avg.	**AMC**	**Minerva**	**MATH**	**GSM8K**	**Olympiad**	**AIME25**	**AIME24**
R-Quest	51.64	57.89	58.82	77.60	92.57	46.67	13.44	14.48
w/o novelty	51.02	55.94	56.62	78.00	91.81	45.63	15.42	13.75
w/o validity	50.27	56.48	58.82	77.00	91.74	41.19	13.54	13.12
w/o both (R-Zero)	49.10	54.84	54.78	76.20	92.42	42.37	12.29	10.83
Frozen validity judge	49.93	58.13	56.99	78.20	92.12	43.11	8.02	12.92

Variant	Code generation			General-domain reasoning			
	Avg.	**HumanEval+**	**MBPP+**	**Avg.**	**SuperGPQA**	**MMLU-Pro**	**BBEH**
R-Quest	62.69	59.76	65.61	32.75	29.40	57.36	11.50
w/o novelty	61.98	59.15	64.81	31.80	29.21	55.96	10.24
w/o validity	61.28	58.54	64.02	31.53	28.43	55.39	10.77
w/o both (R-Zero)	59.27	56.10	62.43	31.21	28.66	54.01	10.95
Frozen validity judge	61.85	59.15	64.55	32.39	29.41	56.25	11.50

F Additional Results

F.1 Detailed Ablation Results

Table 9 reports the individual benchmark scores underlying the domain averages in Table 3. Removing either validity or novelty feedback lowers performance on both code generation benchmarks and all three general reasoning benchmarks. The effects vary across mathematical benchmarks, with removing validity feedback producing the largest drop on OlympiadBench. Although individual ablations achieve higher scores on some benchmarks, the full R-Quest achieves the highest average performance in all three domains.