R_Q-*Evolve*: LLMs Evolve Their Own Generator Programs for Self-Improvement

Anonymous authors
Paper under double-blind review

Abstract

In group-relative reinforcement learning with binary verifiable rewards, all-correct or all-incorrect response groups contribute no reward-driven policy gradient. Sustained self-improvement therefore requires problem instances that continue to yield both correct and incorrect responses as the Solver changes. We introduce R_Q-Evolve, which retains *generator programs* in an archive, each representing a task family of problem instances sharing the same structure. Each iteration, the Solver reassesses retained programs on newly sampled instances and, as the Evolver, mutates them into child programs. *Reasoning Quality* (R_Q) estimates the current usefulness of retained programs and their children by multiplying learnability, measured as verifier reward variance, by uncertainty, measured as mean token entropy. We motivate this product by deriving an upper bound on the squared norm of a centered-reward surrogate gradient in output-logit space. Fitness ranks eligible retained programs for training, while retention confines competition to cells defined by mathematical domain and output form, leaving programs in other cells available for later reassessment and mutation. With Qwen3-4B-Base and Qwen3-8B-Base, R_Q-Evolve achieves final mathematical benchmark gains of 5.24 and 6.12 percentage points over the base models. Both scales reach their highest averages late in training, and the final 8B checkpoint exceeds R-Zero's peak by 1.06 points. Classified generated instances remain broadly distributed across domains and output forms late in the 8B run.

1 Introduction

Self-generated tasks have a long history in automatic curriculum learning and open-ended learning (Schmidhuber, 2013; Sukhbaatar et al., 2017; Wang et al., 2019). Recent self-evolving reasoning systems extend this approach to large language models (LLMs), pairing a component that generates problem instances with the Solver that answers them (Zhao et al., 2026; Huang et al., 2026b). The Solver trains on those problems through reinforcement learning with verifiable rewards (RLVR) (Lambert et al., 2024), where a verifier assigns a binary reward to each sampled response. Under the centered group-relative advantage, an all-correct or all-incorrect response group contributes no reward-driven policy gradient. Such groups become increasingly common as more prompts reach accuracy 1, reducing the effective number of prompts in each batch (Yu et al., 2026). Curricula based on the Solver's current success rate therefore prioritize problems with intermediate success rates (Rutherford et al., 2024; Bae et al., 2026), but selection alone does not generate new problems. Success rates may enter or leave this range as the Solver changes, requiring a continuing supply of useful problems.

Self-evolving reasoning systems meet this need by training the component that generates problems, with a reward estimated in part from the current Solver's responses (Huang et al., 2026b; Zhao et al., 2026). Problems generated this way have been observed to repeat reasoning patterns across iterations despite differences in their statements (Li et al., 2026). Thus, continued generation does not ensure that the training supply remains structurally broad over time.

We propose R_Q-Evolve, which retains *generator programs* in an archive to keep different task families available for reassessment and mutation as the Solver changes (Figure 1). A *task family* is a set of problem instances sharing the same structure and differing in sampled values. A generator program represents a task family by mapping a random seed to a problem instance. Running a retained

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

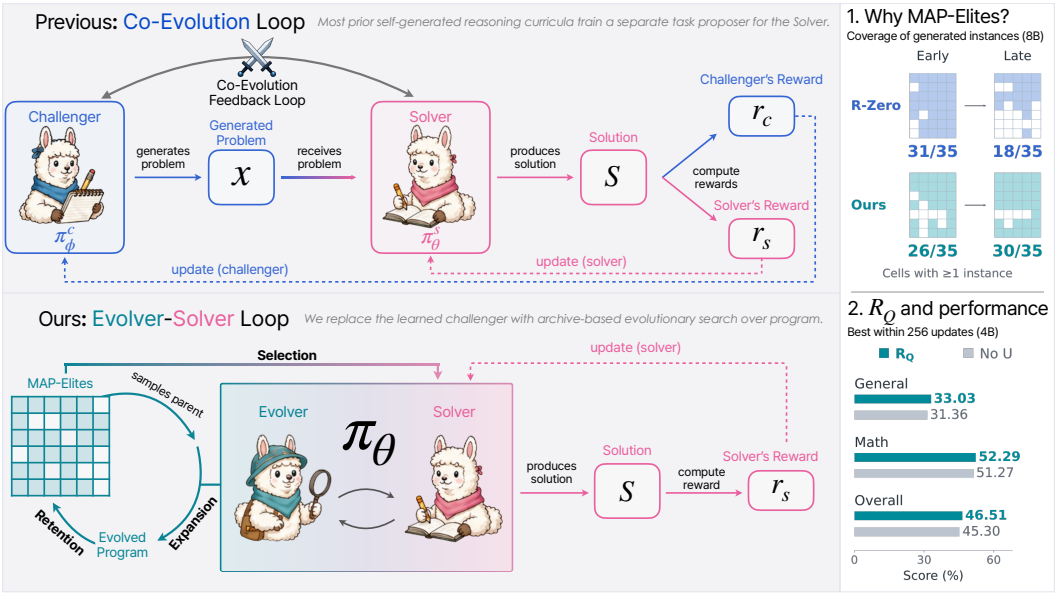


Figure 1: R_Q-Evolve uses the Solver to evolve archived generator programs that supply its training instances. **Left:** The R-Zero Challenger–Solver loop (top) is compared with the R_Q-Evolve Evolver–Solver loop (bottom). **Upper right:** distributions of generated instances across mathematical domain–output-form cells early and late in training. **Lower right:** 4B benchmark averages for R_Q-Evolve and the No U variant, which removes the token-entropy factor.

program supplies a newly sampled instance for reassessing its task family against the current Solver. The Solver also acts as the *Evolver* by mutating retained programs to explore new task families, so no separate model is trained to generate problems.

A task family's usefulness may change with the Solver, so we reassess retained programs and score children on newly sampled instances. Reasoning Quality (R_Q), our fitness for training-batch selection, multiplies estimates of learnability (L) and uncertainty (U). Learnability is the reward variance implied by the success rate (Foster et al., 2025), and is largest when correct and incorrect responses are evenly split. The associated policy gradient also depends on the Solver's next-token distributions at the sampled tokens. We measure their uncertainty as mean token entropy. The product LU enters an upper bound on the squared norm of the centered-reward surrogate gradient in output-logit space.

Retention keeps task families available for later reassessment and mutation, but a global replacement rule based on R_Q would let the highest-scoring task families displace the rest. We therefore use MAP-Elites to retain one representative per cell (Mouret & Clune, 2015), each pairing one of seven mathematical domains with one of five output forms, such as a Boolean decision, a complete finite solution set, or a count. A child competes only with its cell's incumbent, so higher fitness in one cell does not displace programs in others.

We evaluate performance during training, generated distributions, and component contributions. R_Q-Evolve raises the final mathematical benchmark average over Qwen3-4B-Base and Qwen3-8B-Base by 5.24 and 6.12 percentage points (Table 1). Both scales reach their highest mathematical averages late in training. At 8B, the final checkpoint exceeds R-Zero's peak by 1.06 points at the same update budget (Figure 3). Classified generated instances remain broadly distributed late in the 8B run (Figure 4). Three 4B variants score lower on every benchmark aggregate, supporting the contributions of reassessment, uncertainty, and Solver-conditioned fitness (Table 2).

Our main contributions are:

1. **Evolving generator programs with the Solver.** Each archived generator program supplies newly sampled instances and remains available for mutation as the Solver changes.

2. **Scoring generator programs against the current Solver.** R_Q multiplies learnability by uncertainty, both estimated on newly sampled instances, to rank eligible programs across cells for training-batch selection. Replacement remains within domain–output-form cells to preserve coverage.
3. **Empirical support for sustained adaptation.** The observed Solver gains, late distribution breadth, and component contributions support adapting the problem supply as the Solver learns.

2 PRELIMINARIES

2.1 RLVR WITH A DYNAMIC CURRICULUM

We consider RLVR under an evolving curriculum $\mathcal{D}_t$, defined as a distribution over training problem instances. At Solver iteration t, let $\pi^{\mathrm{S}}_{\theta_t}$ denote the current Solver policy and $q^{\mathrm{S}}_{\theta_t}$ the distribution used to sample its responses under fixed decoding settings. A problem instance $(x, y^\star, v)$ comprises a problem statement x, reference answer $y^\star$, and verifier specification v. The verifier uses specification v to check response y against reference answer $y^\star$ and returns a binary reward $r_v(y, y^\star) \in \{0, 1\}$.

For a fixed curriculum $\mathcal{D}_t$, the Solver aims to maximize the expected verifier reward,

$$J_{\mathrm{solve}}(\theta; \mathcal{D}_t) := \mathbb{E}_{(x,y^\star,v)\sim\mathcal{D}_t}\left[\mathbb{E}_{y\sim q^{\mathrm{S}}_\theta(\cdot|x)}\left[r_v(y, y^\star)\right]\right]. \tag{1}$$

Solver updates use a group-relative advantage with the group mean reward as the baseline (Shao et al., 2024). Given $G \geq 2$ responses sampled independently for the same instance, with rewards $r_i = r_v(y_i, y^\star)$ and observed success rate $\hat{s}_t(x) = \frac{1}{G}\sum_{i=1}^{G} r_i$, each response is credited with the centered advantage

$$A_i = r_i - \hat{s}_t(x). \tag{2}$$

We use this centered advantage without reward-standard-deviation normalization, consistent with the learnability analysis of Foster et al. (2025). An all-correct or all-incorrect group contributes no reward-driven policy gradient because every A_i is zero (Yu et al., 2026).

For a fixed instance, let $s_t(x)$ be the probability that the verifier accepts a response sampled from $q^{\mathrm{S}}_{\theta_t}$, and let $L_t(x) = s_t(x)(1 - s_t(x))$ be its learnability, the variance of the verifier reward (Rutherford et al., 2024; Foster et al., 2025). Its unbiased estimate from the same G rollouts is the sample variance of the verifier rewards, expressed through the centered advantages,

$$\hat{L}_t(x) = \frac{1}{G-1}\sum_{i=1}^{G} A_i^2 = \frac{G}{G-1}\hat{s}_t(x)\big(1 - \hat{s}_t(x)\big). \tag{3}$$

Estimated learnability is largest at an intermediate success rate and is zero for exactly the all-correct and all-incorrect groups.

2.2 FROM A LEARNED CHALLENGER TO ARCHIVE-BASED EVOLUTION

To adapt $\mathcal{D}_t$, R-Zero trains a separate Challenger that samples problem proposals from $q^{\mathrm{C}}_\phi(\cdot \mid h^{\mathrm{C}}_t)$, with parameters ϕ and generation context h^{C}_t (Huang et al., 2026b). For a fixed Solver checkpoint θ_t, we write its optimization abstractly as

$$J_{\mathrm{C}}(\phi; \theta_t, h^{\mathrm{C}}_t) := \mathbb{E}_{c\sim q^{\mathrm{C}}_\phi(\cdot|h^{\mathrm{C}}_t)}\left[r_{\mathrm{C}}(c; \theta_t)\right]. \tag{4}$$

Here c is a proposal and r_{C} denotes the Challenger reward, whose Solver-conditioned component is estimated from Solver rollouts.

The Challenger adapts problem generation by updating ϕ. R$_{\mathrm{Q}}$-Evolve instead retains generator programs in an archive, and the current Solver acts as an Evolver that mutates them. At fixed θ_t, let $\mathcal{A}_{t,k}$ denote the archive at evolution step k and $h^{\mathrm{E}}_{t,k}(c)$ the mutation context for a parent c sampled from the archive at batch start. Mutation and archive updating take the form

$$c' \sim q^{\mathrm{E}}_{\theta_t}\big(\cdot \mid h^{\mathrm{E}}_{t,k}(c)\big), \quad \mathcal{A}_{t,k+1} = \Gamma(\mathcal{A}_{t,k}, c', F_{\mathrm{E},t}(c'; \theta_t)). \tag{5}$$

Here $q^{\mathrm{E}}_{\theta_t}$ is the Evolver's mutation distribution, $F_{\mathrm{E},t}$ is Solver-conditioned fitness, and Γ is the archive update. Like r_{C}, this fitness uses Solver feedback, but guides archive retention rather than a separate generator-parameter update. Section 3 instantiates this fitness as R_Q and specifies archive evolution.

3 METHOD

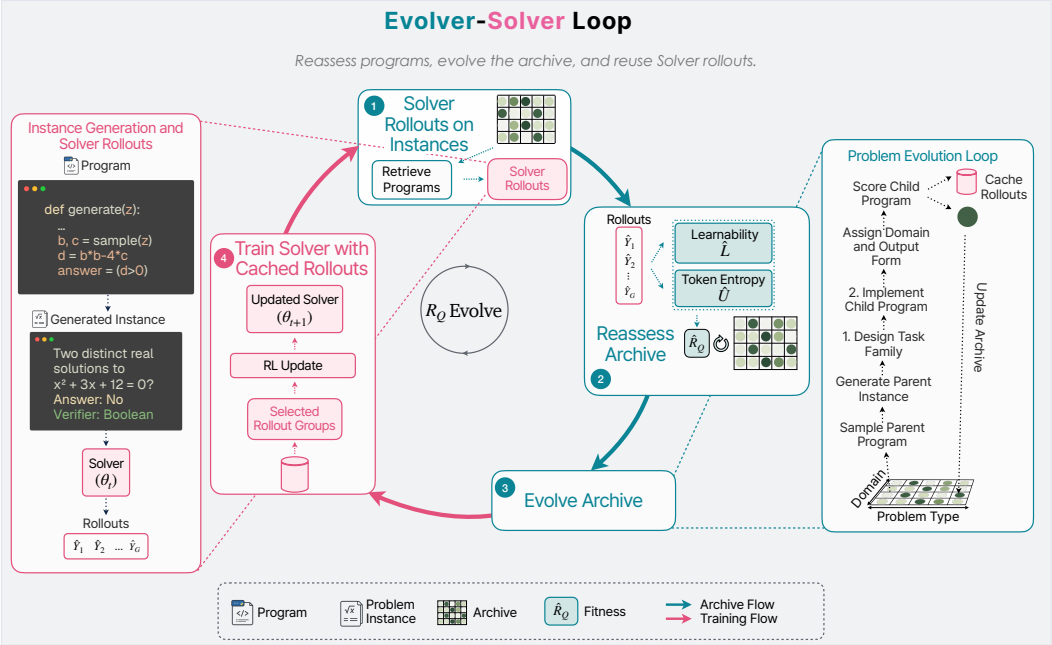


Figure 2: R_Q-Evolve reassesses generator programs on newly sampled instances and expands the archive through two-stage mutation. Rollouts from reassessment and child evaluation are cached for the Solver update. The left panel illustrates a condensed generator program for quadratic equations.

3.1 GENERATOR PROGRAMS

In Equation 5, each candidate c is a generator program P mapping a seed z to a problem instance $P(z) = (x_z, y_z^\star, v_z)$ from its task family (Figure 2, left). For each seed, the program samples values for the problem statement and computes the corresponding reference answer $y_z^\star$. Because this answer determines the Solver's verifier reward, we instruct the Evolver to implement a different calculation of the same answer as a cross-check, such as direct summation instead of a closed-form formula. Following the principle of differential testing (McKeeman, 1998), validation rejects programs whose reference answers and cross-checks disagree on validation seeds.

3.2 REASONING QUALITY

To score a generator program against the current Solver, we sample an instance $P(z)$ and estimate learnability and uncertainty from the same G rollouts from $q^{\mathrm{S}}_{\theta_t}(\cdot \mid x_z)$.

Learnability. For the fixed program and checkpoint, we abbreviate $\hat{L}_t(x_z)$ and $\hat{s}_t(x_z)$ in Equation 3 as $\hat{L}_z$ and $\hat{s}_z$. Learnability measures variation in verifier rewards, while the associated policy gradient also depends on the Solver's next-token distributions at the sampled tokens, which the success rate does not determine.

Uncertainty. We measure the uncertainty of these next-token distributions as mean token entropy. Let $H_{\theta_t}(x, y_{<\ell})$ denote the entropy of the full-softmax Solver policy $\pi^{\mathrm{S}}_{\theta_t}$ at a generated prefix, using natural logarithms. We average over all $T(y)$ response tokens and then over the G rollouts:

$$h_t(y \mid x) = \frac{1}{T(y)} \sum_{\ell=1}^{T(y)} H_{\theta_t}(x, y_{<\ell}), \qquad \hat{U}_z = \frac{1}{G} \sum_{i=1}^{G} h_t(y_{z,i} \mid x_z). \tag{6}$$

Fitness. To motivate fitness, we examine how each response contributes to the centered-reward surrogate gradient. Fix an instance and checkpoint, and let q be the rollout distribution obtained by nucleus sampling (Holtzman et al., 2020) from the full-softmax Solver policy at temperature one. Write r for the binary verifier reward, $s = \mathbb{E}_q[r]$, $L = s(1-s)$, and $U = \mathbb{E}_q[h_t(Y \mid x)]$ for the population quantities. With an independent logit block in $\boldsymbol{\eta}$ for every possible prefix, define $g = \nabla_{\boldsymbol{\eta}} \log \pi^{\mathrm{S}}_{\boldsymbol{\eta}}(Y \mid x)$ at the current logits, holding q and s fixed. Here $r - s$ subtracts the expected reward s, whereas Equation 2 subtracts the observed group mean. The vector g points in the direction that increases the response's log-probability in output-logit space. Their product $(r-s)g$ is the response's contribution to the surrogate gradient, weighted positively for above-average reward and negatively for below-average reward. Averaging these contributions gives $\mathbb{E}_q[(r-s)g]$.

Proposition 1 (Surrogate-gradient bound) *For a finite vocabulary and response lengths $1 \le T(Y) \le T_{\max}$, with the same response tokens used for log-probability gradients and entropy, the centered-reward surrogate gradient under q satisfies*

$$\left\|\mathbb{E}_q[(r-s)g]\right\|_2^2 \le L\,\mathbb{E}_q\left[\|g\|_2^2\right] \le T_{\max} LU. \tag{7}$$

The first inequality uses the reward variance $\mathbb{E}_q[(r-s)^2] = L$, while the second bounds the expected squared norm of g by $T_{\max}U$. At fixed $T_{\max}$, the product LU controls this upper bound, whose proof appears in Appendix A.2. Among problem instances with similar success rates, higher response entropy is associated with larger length-normalized gradient norms (Wu et al., 2026), an empirical basis for using entropy as an additional selection signal. We use the estimated product $\hat{R}_Q$ as the generator program's fitness in Equation 5:

$$F_{\mathrm{E},t}(P;\theta_t) := \hat{R}_Q(P;\theta_t) = \hat{L}_z \hat{U}_z. \tag{8}$$

The fitness is zero whenever all responses to the scored instance are correct or all are incorrect. For an unfiltered group of G responses, $\hat{L}_z$ takes only $\lfloor G/2 \rfloor$ positive values, so token entropy also provides a ranking signal among programs tied at nonzero learnability. Section 4.3 evaluates the uncertainty factor empirically against No U, which uses learnability alone. Unlike the analysis of unfiltered rollouts in Appendix A, fitness estimation excludes overlong responses, while Solver updates reuse the original rollout groups.

3.3 Archive Evolution

We use a MAP-Elites archive to prevent high fitness in one domain–output-form combination from displacing representatives in others. Pairing seven mathematical domains with five output forms yields 35 cells, each holding at most one representative. Retention compares children only with their cell's incumbent, whereas training-batch selection ranks eligible retained programs across cells. Figure 2 outlines one iteration, and Appendix B gives the full algorithm.

Reassessment. Each iteration begins by reassessing every retained program against the updated Solver on an instance sampled with an unused seed and caching the rollouts. A successful measurement replaces the previous fitness, even when the new value is zero. Scoring failure leaves the previous score unchanged and supplies no cached group for that update.

Expansion and retention. Mutation parents are sampled uniformly from occupied cells regardless of fitness, making every retained program, including one with zero current fitness, eligible as a parent. Because jointly requesting a mathematical mutation and its code implementation did not reliably produce the intended changes during development, the Evolver mutates parents in two stages. Without access to the parent's source code, the first stage specifies a child task family from the parent's task family and an example instance. The second implements it as a generator program. Neither stage receives a target cell. After validation, the current Solver assigns the child's domain, and its output form is inferred from the statement and checked against the verifier. The resulting cell stays fixed when the program is later reassessed. The child is scored as in Section 3.2 and competes only in its cell, filling the cell if it is empty and otherwise replacing the incumbent only if its fitness is strictly higher. Zero fitness alone does not evict a program. In Equation 5, $q^{\mathrm{E}}_{\theta_t}$ is this two-stage mutation, and Γ comprises admission checks and this cell-wise update. Mutation, scoring, and retention repeat over batches of children until a stopping criterion holds.

Training-batch selection and Solver update. Eligible programs are those in the final archive whose scored instance yields both correct and incorrect responses ($0 < \hat{s}_z < 1$), including newly retained children. Their cached rollout groups fill the training batch in descending order of current fitness, determining the curriculum $\mathcal{D}_t$ for this iteration. If the cached groups do not fill the training batch, we sample additional instances from these programs and generate rollouts with the same Solver checkpoint. These additional rollouts fill the shortfall without changing program fitness. Extra instances of an eligible program need not individually yield both correct and incorrect responses. Appendix B specifies the fallback and replay rules when eligible cached groups do not fill the batch. The Solver update reuses the cached rollouts and their binary verifier rewards instead of regenerating responses for training. The updated Solver differs from the one that scored the archive, and the next iteration begins by reassessing the retained programs against it.

4 Experiments

We evaluate how Solver performance changes throughout training, whether generated problem instances remain broadly distributed, and which components contribute to the results.

We initialize the Solver from Qwen3-4B-Base (4B) or Qwen3-8B-Base (8B) (Yang et al., 2025) and the archive with seven hand-authored generator programs, one per mathematical domain. We compare with the base models, Absolute Zero (Zhao et al., 2026), R-Zero (Huang et al., 2026b), and G-Zero (Huang et al., 2026a) on seven mathematical and three general reasoning benchmarks. Our evaluations use code adapted from R-Zero. We describe the benchmark protocols, baseline selection, and score sources in Appendix C.1.

4.1 Solver Performance and Training Cost

Table 1: Benchmark scores in percent. R_Q-Evolve uses the final checkpoints at update 256, and our 8B R-Zero run its peak at update 96. Math, General, and Overall are unweighted means of seven, three, and ten benchmark scores. Dashes indicate unavailable scores or averages. Unmarked rows use our evaluations. † marks published scores for methods we did not run, and ‡ published scores selected over our own evaluations. Bold and underline mark the highest and second-highest scores per column and scale. MATH, Min., and Oly. denote MATH-500, Minerva, and OlympiadBench. R_Q-Evolve achieves the highest reported mathematical and overall averages at both scales.

	Mathematical reasoning								General reasoning				Overall
Method	MATH	GSM8K	AMC	$AIME_{24}$	$AIME_{25}$	Min.	Oly.	**Avg.**	SuperGPQA	MMLU-Pro	BBEH	**Avg.**	**Avg.**
Qwen3-4B-Base													
Base	74.00	89.69	48.36	11.88	<u>11.04</u>	50.37	44.00	47.05	25.33	51.20	7.94	28.16	41.38
Absolute Zero†	76.20	89.34	<u>52.45</u>	12.20	10.20	41.96	42.56	46.42	27.10	52.60	8.30	29.33	41.29
R-Zero‡	<u>79.60</u>	<u>92.12</u>	**57.27**	<u>13.40</u>	9.60	<u>52.94</u>	<u>44.59</u>	<u>49.93</u>	27.55	<u>55.47</u>	<u>10.42</u>	<u>31.15</u>	<u>44.30</u>
G-Zero†	74.80	90.22	47.81	11.15	7.50	47.06	41.10	45.66	<u>27.56</u>	54.14	–	–	–
R_Q-Evolve (ours)	**82.00**	**93.40**	51.56	**22.92**	**14.17**	**55.15**	**46.81**	**52.29**	**30.13**	**57.30**	**11.66**	**33.03**	**46.51**
Qwen3-8B-Base													
Base	73.60	92.49	51.17	<u>20.42</u>	12.50	52.94	45.04	49.74	**31.99**	57.70	8.63	32.77	44.65
Absolute Zero†	76.60	92.00	57.89	18.40	<u>18.20</u>	57.90	47.80	52.68	<u>31.89</u>	60.50	<u>10.80</u>	34.40	47.20
R-Zero	<u>81.20</u>	**93.71**	<u>63.44</u>	17.60	**19.38**	<u>58.46</u>	<u>49.78</u>	<u>54.80</u>	31.38	<u>61.53</u>	10.60	<u>34.50</u>	<u>48.71</u>
G-Zero†	76.40	93.33	49.77	12.71	10.83	48.53	44.21	47.97	31.24	59.44	–	–	–
R_Q-Evolve (ours)	**83.20**	<u>93.63</u>	**65.00**	**21.15**	14.69	**61.03**	**52.30**	**55.86**	31.37	**62.80**	**12.36**	**35.51**	**49.75**

The final checkpoints in Table 1 improve mathematical averages over the base models by 5.24 percentage points at 4B and 6.12 at 8B. The 4B run also reaches its highest mathematical average at the final checkpoint, as shown in Appendix C.2. Although trained only on mathematical problem instances, R_Q-Evolve improves general reasoning averages over the base models by 4.87 and 2.74 percentage points at 4B and 8B, respectively. At both scales, it exceeds R-Zero in general reasoning averages and outperforms G-Zero, a recent co-evolutionary method, on all nine benchmarks with reported scores (Chu et al., 2026).

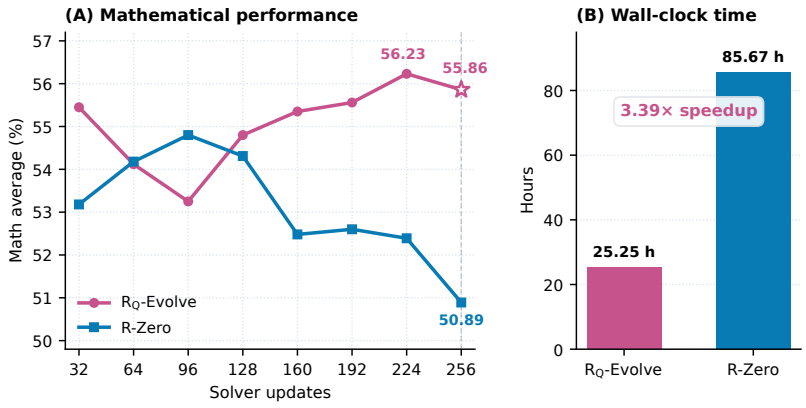


Figure 3: 8B mathematical averages over seven benchmarks (A) and total wall-clock time for 256 Solver updates on the same eight-GPU server (B). The star marks the final checkpoint. Without a separate Challenger optimization stage, R_Q-Evolve reaches its highest mathematical average late in training and completes the same update budget faster than R-Zero.

In the 8B run shown in Figure 3(A), the average reaches 55.45 by update 32, declines to 53.25 at 96, rises to its highest value of 56.23 at 224, and ends at 55.86. The R-Zero run peaks at 54.80 at update 96 and ends at 50.89. The final R_Q-Evolve checkpoint exceeds R-Zero's peak checkpoint, which Table 1 reports, by 1.06 points and R-Zero's final checkpoint by 4.97 points.

To test whether the final-checkpoint advantage over R-Zero extends to an additional benchmark and multiple domain–output-form combinations, we evaluate Omni-MATH-2, a revised version of the Olympiad-level Omni-MATH benchmark (Ballon et al., 2026). We combine the published domain labels with our output-form classification for cellwise evaluation. Under a shared evaluation protocol, the final 8B Solver achieves 33.21% accuracy on the filtered subset, compared with 28.75% for the base model and 28.82% for the final R-Zero checkpoint. In the cellwise comparison, R_Q-Evolve outperforms R-Zero in 21 cells, underperforms in 10, and ties in 4. We detail the evaluation subset, cell assignments, and cellwise results in Appendix E.

Figure 3(B) reports 25.25 hours for R_Q-Evolve versus 85.67 for R-Zero on the same eight-GPU server, a 3.39× end-to-end speedup. R-Zero additionally optimizes the Challenger with rewards whose Solver-conditioned component is estimated from Solver rollouts. R_Q-Evolve has no separate Challenger optimization stage and reuses scoring rollouts for the Solver update, with additional rollouts needed for instances sampled to fill the training batch. We provide the timing protocol and training trajectories at both model scales in Appendix C.2.

4.2 Generated Distributions and Program Evolution

Generated distributions. We measure how the domain–output-form distribution of generated instances changes as the Solver learns. The analysis includes instances from mutation candidates beyond those selected for Solver training. For generated instances from both 8B runs, `gpt-5.6-luna` assigns mathematical domains and fixed rules assign output forms, separately from archive admission. We summarize each classified distribution by its effective number of cells, the number of equally populated cells with the same entropy. Across the three sampled windows in Figure 4(A), the classified R_Q-Evolve distributions have 18.3, 21.7, and 21.6 effective cells. The R-Zero run falls from 14.1 to 2.4 effective cells, with algebra/optimization accounting for 75.6% of its final window in Figure 4(B).

We next examine how Omni-MATH-2 accuracy changes alongside these distributions, using the 8B checkpoints at updates 32, 128, and 256. Both methods improve over the shared initialization at update 32 in Figure 4(C). R_Q-Evolve maintains broadly distributed generation and retains these gains through the final checkpoint, whereas R-Zero's generation becomes concentrated and its accuracy returns close to the base-model level.

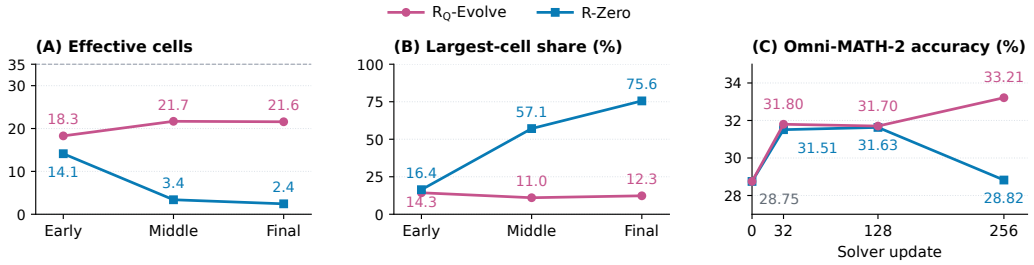


Figure 4: Generated-instance diversity across 35 domain–output-form cells and Omni-MATH-2 accuracy during the 8B runs. (A) Effective cells, $\exp(-\sum_c p_c \log p_c)$, where p_c is the cell share among classified instances in each window. (B) Largest cell share, $\max_c p_c$. (C) Omni-MATH-2 accuracy, with update 0 denoting the shared Qwen3-8B-Base initialization. The Early, Middle, and Final generation windows are paired with the Solver checkpoints at updates 32, 128, and 256 in (C), respectively. Sampling windows and evaluation details appear in Appendices C.2 and E, respectively. R_Q-Evolve maintains broadly distributed generation and retains its accuracy gains, while R-Zero becomes more concentrated and returns near its initial accuracy.

Archive dynamics. We track archive coverage and program replacement throughout training. Figure 5 in Appendix C.2 shows program replacement continuing as archive coverage stabilizes. Of the admissions that either fill an empty cell or replace its incumbent with a different program, 100% at 4B and 98.9% at 8B are replacements in the last 96 updates.

What mutation changes. In one 4B lineage, mutation changes the structure of the task family from filtering individual divisors, to transforming their values, and then to relating divisor pairs. Fixing $n = 80$ makes these structural changes directly comparable. The immediate parent of the pair-counting program has zero current fitness, with all responses to its scored instance correct, yet remains eligible as a mutation parent (Section 3.3). Its child is retained in the archive after yielding both correct and incorrect responses on its scored instance, making it eligible for training-batch selection. The condensed generator programs, answer derivations, and fitness history are presented in Appendix D.

4.3 Component Variants

To assess which components contribute to the Solver gains, we vary reassessment on newly sampled instances, the uncertainty factor, and the use of Solver-conditioned fitness at 4B, leaving the archive grid and two-stage mutation unchanged.

Table 2: Component variants at 4B. Each row uses the checkpoint with the highest mathematical average within 256 Solver updates. No Refresh fixes the sampled instance, No U omits token entropy, and Random randomizes replacement and training selection. Math, General, and Overall are unweighted means of seven, three, and ten benchmark scores, respectively (%). The full method outperforms all three variants on mathematical, general reasoning, and overall averages.

Variant	Math	General	Overall
R_Q-Evolve	**52.29**	**33.03**	**46.51**
No Refresh	51.74	31.89	45.79
No U	51.27	31.36	45.30
Random	51.00	31.66	45.20

The full method uses the same final checkpoint in Tables 1 and 2. No Refresh samples new responses to one fixed instance per program, so its lower scores support sampling new instances for both reassessment and training. The No U and Random comparisons support, respectively, including uncertainty in fitness and using fitness for retention, training priority, and the allocation of additional instances. We specify each intervention and the checkpoint-selection protocol in Appendix C.3.

5 Related Work

Learned problem generation in self-evolving reasoning. Learned problem generation adapts a proposal policy using current learner feedback (Dennis et al., 2020). Absolute Zero shares parameters between problem generation and solving (Zhao et al., 2026), whereas R-Zero trains a separate Challenger using rewards informed by the current Solver's responses (Huang et al., 2026b). For open-ended generation, G-Zero uses hints to reward task proposals and construct response preferences for Solver training (Huang et al., 2026a). To discourage repetition across iterations, R-Diverse adds a memory-based penalty to proposal training (Li et al., 2026). These methods adapt the problem supply through proposal-policy optimization. R_Q-Evolve instead mutates and retains generator programs under a fixed Solver checkpoint, avoiding separate optimization for problem generation.

Evolving problem instances versus generator programs. Evolutionary curricula retain and mutate environments or levels, reassessing them against the current learner (Wang et al., 2019; Parker-Holder et al., 2022). For RLVR, DéjàQ retains problem instances and reassesses their learnability as the Solver trains (Röpke et al., 2026). Program search retains programs and evaluates what they generate, as in FunSearch (Romera-Paredes et al., 2024), or co-evolves generator and heuristic solver programs, as in ACEvo (Duan et al., 2026). For an evolving Solver, retaining a generator program extends reassessment beyond a single retained problem instance by supplying newly sampled instances from the same task family for both scoring and training.

Solver-conditioned utility and diversity-preserving retention. Candidate utility is estimated from learning potential (Jiang et al., 2021), reward variance or intermediate success rates (Rutherford et al., 2024; Foster et al., 2025; Bae et al., 2026), or reward history combined with response entropy (Wu et al., 2026). Multiplying learnability by mean token entropy from the same scoring rollouts provides an additional ranking signal among generator programs tied at nonzero learnability. The score does not determine the scope of competition. MAP-Elites confines replacement to cells defined by chosen features (Mouret & Clune, 2015); applying this rule to domain–output-form cells prevents higher fitness in one cell from displacing programs in another. When evaluations change over time, dynamic extensions of MAP-Elites reassess the archived candidates (Gallotta et al., 2024).

6 Conclusion

R_Q-Evolve sustains the problem supply through archived generator programs that support reassessment on newly sampled instances and mutation by the Solver to explore new task families. R_Q, estimated on newly sampled instances for retained programs and their children, guides training-batch selection and retention, with replacement confined to domain–output-form cells to prevent higher fitness in one cell from displacing programs in others. Final mathematical averages improve over the base models at both scales. Classified generated instances remain broadly distributed across mathematical domains and output forms late in the 8B run. The 4B component comparisons support the contributions of reassessment on newly sampled instances, uncertainty, and Solver-conditioned replacement and training selection to Solver performance. The 8B run also achieves a $3.39\times$ speedup over R-Zero at the same Solver update budget. These results support adapting the problem supply through task families whose generator programs are reassessed and evolved as the Solver learns.

7 Limitations

Validation on a finite set of seeds does not establish correctness for all instances of a task family, even when reference answers agree with their cross-checks. The generator–verifier interface also requires computed reference answers and does not assess free-form proofs. Moreover, estimating a task family's current usefulness from one newly sampled instance makes program rankings sensitive to variation across instances and sampled responses. Mean token entropy $\hat{U}_z$ may also reflect variation in wording, even when the underlying mathematical solution is unchanged. Finally, retention is limited to the fixed 7×5 domain–output-form grid, and an incorrect domain assignment persists when a program is reassessed.

AI USE STATEMENT

We used generative AI for research methodology, implementation, evaluation, analysis, and manuscript preparation. Our framework uses Qwen3-4B-Base and Qwen3-8B-Base as the Solver and Evolver, with the Evolver producing generator programs that supply synthetic problem instances and reference answers for training. AI assisted with refining research ideas, hypotheses, and conceptual frameworks, reviewing methodology and experimental design, developing code, and analyzing and interpreting results. For evaluation, we used `gpt-5.6-luna` for post-hoc mathematical domain classification, `gpt-4o` to recheck mathematical responses that failed symbolic verification, and `gpt-5-mini-2025-08-07` for answer-equivalence checking on Omni-MATH-2. AI also assisted with manuscript drafting, revision, translation, proofreading, and readability improvements. We reviewed and, where necessary, independently verified AI-assisted artifacts, results, code, and text, and take full responsibility for this work.

REFERENCES

Sanghwan Bae, Jiwoo Hong, Min Young Lee, Hanbyul Kim, JeongYeon Nam, and Donghyun Kwak. Online difficulty filtering for reasoning oriented reinforcement learning. In *Proceedings of the 19th Conference of the European Chapter of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 700–719, 2026.

Marthe Ballon, Andres Algaba, Brecht Verbeken, and Vincent Ginis. Benchmarks saturate when the model gets smarter than the judge. *arXiv preprint arXiv:2601.19532*, 2026.

Gyouk Chu, Myeongho Jeon, and Eunho Yang. J-zero: Unified challenger–solver–judge co-evolution from zero data. *arXiv preprint arXiv:2608.26582*, 2026.

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.

Michael Dennis, Natasha Jaques, Eugene Vinitsky, Alexandre Bayen, Stuart Russell, Andrew Critch, and Sergey Levine. Emergent complexity and zero-shot transfer via unsupervised environment design. *Advances in neural information processing systems*, 33:13049–13061, 2020.

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.

Ruibo Duan, Yuxin Liu, Haoran Ye, Xinyao Dong, zhiqiang xu, and Chenglin Fan. ACEvo: Adversarial co-evolution of problem distributions and solvers for combinatorial optimization. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=wyyIZFdpXC.

Thomas Foster, Anya Sims, Johannes Forkel, Mattie Fellows, and Jakob Foerster. Learning to reason at the frontier of learnability. *arXiv preprint arXiv:2502.12272*, 2025.

Roberto Gallotta, Antonios Liapis, and Georgios N Yannakakis. Dynamic quality-diversity search. In *Proceedings of the Genetic and Evolutionary Computation Conference Companion*, pp. 463–466, 2024.

Bofei Gao, Feifan Song, Zhe Yang, Zefan Cai, Yibo Miao, Qingxiu Dong, Lei Li, Chenghao Ma, Liang Chen, Zhengyang Tang, et al. Omni-math: A universal olympiad level mathematic benchmark for large language models. In *International Conference on Learning Representations*, volume 2025, pp. 100540–100569, 2025.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 3828–3850, 2024.

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. *arXiv preprint arXiv:2103.03874*, 2021.

Ari Holtzman, Jan Buys, Li Du, Maxwell Forbes, and Yejin Choi. The curious case of neural text degeneration. In *International Conference on Learning Representations*, 2020. URL https://openreview.net/forum?id=rygGQyrFvH.

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, 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. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026b.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. In *International Conference on Machine Learning*, pp. 4940–4950. PMLR, 2021.

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.

Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023.

Hynek Kydlíček. Math-Verify: Math Verification Library. URL https://github.com/huggingface/math-verify.

Nathan Lambert, Jacob Morrison, Valentina Pyatkin, Shengyi Huang, Hamish Ivison, Faeze Brahman, Lester James V Miranda, Alisa Liu, Nouha Dziri, Shane Lyu, et al. Tulu 3: Pushing frontiers in open language model post-training. *arXiv preprint arXiv:2411.15124*, 2024.

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, Dan Zhang, Ruiqi Liu, Renrui Zhang, Zijun Yao, Junfeng Fang, Haiyun Guo, and Jinqiao Wang. R-diverse: Mitigating diversity illusion in self-play llm training. *arXiv preprint arXiv:2602.13103*, 2026.

Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *International Conference on Learning Representations*, volume 2024, pp. 39578–39601, 2024.

William M McKeeman. Differential testing for software. *Digital Technical Journal*, 10(1):100–107, 1998.

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

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. In *International Conference on Machine Learning*, pp. 17473–17498. PMLR, 2022.

Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M Pawan Kumar, Emilien Dupont, Francisco JR Ruiz, Jordan S Ellenberg, Pengming Wang, Omar Fawzi, et al. Mathematical discoveries from program search with large language models. *Nature*, 625(7995):468–475, 2024.

Willem Röpke, Samuel Coward, Andrei Lupu, Thomas Foster, Tim Rocktäschel, and Jakob Foerster. Déjàq: Open-ended evolution of diverse, learnable and verifiable problems. *arXiv preprint arXiv:2601.01931*, 2026.

Alex Rutherford, Michael Beukman, Timon Willi, Bruno Lacerda, Nick Hawes, and Jakob Foerster. No regrets: Investigating and improving regret approximations for curriculum discovery. *Advances in Neural Information Processing Systems*, 37:16071–16101, 2024.

Jürgen Schmidhuber. Powerplay: Training an increasingly general problem solver by continually searching for the simplest still unsolvable problem. *Frontiers in psychology*, 4:313, 2013.

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.

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.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play. *arXiv preprint arXiv:1703.05407*, 2017.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O Stanley. Poet: open-ended coevolution of environments and their optimized solutions. In *Proceedings of the genetic and evolutionary computation conference*, pp. 142–151, 2019.

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.

Jiahao Wu, Ning Lu, Shengcai Liu, Kun Wang, Yanting Yang, Bailong Lin, Chen Jason Zhang, Li Qing, and Ke Tang. Train at moving edge: Online-verified prompt selection for efficient rl training of large reasoning model. *arXiv preprint arXiv:2603.25184*, 2026.

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.

Qiying Yu, Zheng Zhang, Ruofei Zhu, Yufeng Yuan, Xiaochen Zuo, Yu Yue, Weinan Dai, Tiantian Fan, Gaohong Liu, Lingjun Liu, et al. Dapo: An open-source llm reinforcement learning system at scale. *Advances in Neural Information Processing Systems*, 38:113222–113244, 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, 2026.

A Fitness Estimation and Surrogate Bound

A.1 Sampling and Entropy

Fix a problem instance $(x, y^\star, v)$ and Solver checkpoint. Write $p = \pi^{\mathrm{S}}_{\theta_t}$ for the full-softmax policy and $q = q^{\mathrm{S}}_{\theta_t}$ for its rollout distribution. Rollouts use nucleus sampling (Holtzman et al., 2020) at temperature one. For a response Y with $1 \le T(Y) \le T_{\max}$, define

$$h_p(Y) := h_t(Y \mid x) = \frac{1}{T(Y)} \sum_{\ell=1}^{T(Y)} H(p_\ell), \qquad (9)$$
$$s = \mathbb{E}_q[r_v(Y, y^\star)], \qquad L = s(1-s), \qquad U = \mathbb{E}_q[h_p(Y)].$$

Here p_ℓ is the next-token distribution at the realized prefix. Entropy uses natural logarithms and the complete vocabulary $\mathcal{V}$, before nucleus truncation. The entropy statistic averages generated response tokens under the response mask, including a returned end-of-sequence token and excluding padding. Thus $0 \le h_p(Y) \le B := \log |\mathcal{V}|$.

A.2 Proof of Proposition 1

Using p as defined above, the surrogate whose gradient is bounded in Equation 7 is

$$\mathcal{S}_x(\boldsymbol{\eta}) = \mathbb{E}_q[(r-s) \log p_{\boldsymbol{\eta}}(Y \mid x)], \qquad \nabla_{\boldsymbol{\eta}} \mathcal{S}_x = \mathbb{E}_q[(r-s)g]. \qquad (10)$$

The derivative holds q and s fixed and is evaluated at the current logits, with g as defined in Section 3.2.

At any prefix, let S be the nucleus and $\alpha = \sum_{a \in S} p(a)$ its probability mass, so $q(a) = p(a)\mathbf{1}\{a \in S\}/\alpha$. Since S retains the largest probabilities, the p-weighted mean of $p(a)$ within S is at least the corresponding mean outside S. Consequently $\langle q, p \rangle \ge \|p\|_2^2$. When $\alpha = 1$, $q = p$ and equality holds. The token log-probability gradient is $\nabla_{\eta_\ell} \log p_\ell(a) = e_a - p_\ell$, where e_a is a one-hot vector. Therefore

$$\mathbb{E}_{a \sim q_\ell} \|e_a - p_\ell\|_2^2 = 1 - 2\langle q_\ell, p_\ell \rangle + \|p_\ell\|_2^2 \le 1 - \|p_\ell\|_2^2 \le H(p_\ell), \qquad (11)$$

where the entropy bound in Equation 11 follows from $-\log u \ge 1 - u$. Distinct prefixes occupy orthogonal logit blocks, and $\{\ell \le T(Y)\}$ is determined by the preceding tokens, so conditioning on each visited prefix gives

$$\mathbb{E}_q \|g\|_2^2 \le \mathbb{E}_q \sum_{\ell=1}^{T(Y)} H(p_\ell) = \mathbb{E}_q[T(Y) h_p(Y)] \le T_{\max} U, \qquad (12)$$

where the equality undoes the length normalization in h_p and the final step uses $T(Y) \le T_{\max}$. Vector-valued Cauchy–Schwarz, Equation 12, and $\mathbb{E}_q[(r-s)^2] = L$ yield $\|\mathbb{E}_q[(r-s)g]\|_2^2 \le \mathbb{E}_q[(r-s)^2]\, \mathbb{E}_q\|g\|_2^2 \le L T_{\max} U$, proving Equation 7.

A.3 Finite-Sample Estimation

The following results assume $G \ge 2$ i.i.d. rollouts from q for a fixed instance, without outcome- or length-dependent filtering. Each pair (r_i, h_i) uses the reward and mean token entropy of the same rollout. The corrected learnability estimate of Equation 3 and the entropy estimate are

$$\hat{L} = \frac{1}{G(G-1)} \sum_{i \ne j} r_i (1 - r_j), \qquad \hat{U} = \frac{1}{G} \sum_{i=1}^{G} h_i, \qquad (13)$$

where the first expression equals $\frac{G}{G-1} \hat{s}(1-\hat{s})$ for binary rewards. Both are unbiased for L and U, respectively. Their product generally is not:

$$\mathbb{E}[\hat{L}\hat{U}] - LU = \mathrm{Cov}(\hat{L}, \hat{U}) = \frac{1-2s}{G} \mathrm{Cov}_q(r, h_p). \qquad (14)$$

To obtain Equation 14, expand the ordered-pair expression in Equation 13 and separate the entropy index according to whether it equals either reward index:

$$\mathbb{E}[\hat{L}\hat{U}] = \frac{G-2}{G}LU + \frac{(1-s)\mathbb{E}_q[rh_p] + s\mathbb{E}_q[(1-r)h_p]}{G}.$$

For $0 < s < 1$, $\mathrm{Cov}_q(r, h_p) = L\big(\mathbb{E}_q[h_p \mid r = 1] - \mathbb{E}_q[h_p \mid r = 0]\big)$. Since $h_p \in [0, B]$, the absolute bias is at most $|1-2s|LB/G \le B/(4G)$. At $s \in \{0, 1\}$, the bias is zero. The finite-sample correction removes the bias of learnability alone, not all bias in the fitness product.

B Implementation Details

We generate Solver rollouts with vLLM (Kwon et al., 2023) and reuse selected cached rollout groups for Solver updates in verl (Sheng et al., 2024). Algorithm 1 summarizes the bootstrap and subsequent Solver iterations. $\mathcal{C}_t$ stores rollout groups, current fitness, and current eligibility, and $\mathcal{B}_t$ is the selected batch of 32 cached groups. EVALUATE returns the candidate's fitness (or $\perp$ if validation or scoring fails) together with $\mathcal{C}_t$, extended by any successfully scored rollout group. Γ rejects $\perp$. FILLBUDGET collects additional rollout groups without changing program fitness. Allocation, stopping, and selection rules are specified below, and Table 3 lists the settings.

Algorithm 1 $\mathrm{R_Q}$-Evolve: archive evolution with cached Solver rollouts.

Require: Hand-authored generator programs $\{P_i\}_{i=1}^{7}$, Solver θ_0, updates $T = 256$
Ensure: Solver θ_T and archive $\mathcal{A}_{T,0}$

$(\mathcal{A}_{1,0}, \mathcal{C}_0) \leftarrow$ BOOTSTRAP$(\{P_i\}_{i=1}^{7}, \theta_0)$ ▷ validate, score, admit
$\mathcal{B}_0 \leftarrow$ SELECTPAD$(\mathrm{Programs}(\mathcal{A}_{1,0}), \mathcal{C}_0)$
$\theta_1 \leftarrow$ SOLVERUPDATE$(\theta_0, \mathcal{B}_0)$ ▷ bootstrap update
for $t = 1, \ldots, T-1$ **do**
  $(\mathcal{A}_{t,0}, \mathcal{C}_t) \leftarrow$ REASSESS$(\mathcal{A}_{t,0}, \theta_t)$ ▷ newly sampled instances
  $k \leftarrow 0$
  repeat
    $(c_k, \ldots, c_{k+31}) \sim \mathrm{Unif}(\mathcal{A}_{t,k})^{\otimes 32}$ ▷ fixed parent batch
    for $j = k, \ldots, k+31$ **do**
      $c'_j \sim q^{\mathrm{E}}_{\theta_t}(\cdot \mid h^{\mathrm{E}}_{t,j}(c_j))$ ▷ expansion: task family → code
      $(f_j, \mathcal{C}_t) \leftarrow$ EVALUATE$(c'_j, \theta_t, \mathcal{C}_t)$ ▷ score and cache
      $\mathcal{A}_{t,j+1} \leftarrow \Gamma(\mathcal{A}_{t,j}, c'_j, f_j)$ ▷ retention
    end for
    $k \leftarrow k + 32$
  until the mutation stopping criterion holds ▷ 32–128 slots
  $\mathcal{P}_t \leftarrow \mathrm{Programs}(\mathcal{A}_{t,k})$ ▷ include retained children
  $\mathcal{C}_t \leftarrow$ FILLBUDGET$(\mathcal{P}_t, \theta_t, \mathcal{C}_t)$
  $\mathcal{B}_t \leftarrow$ SELECTPAD$(\mathcal{P}_t, \mathcal{C}_t)$ ▷ selection: current fitness
  $\theta_{t+1} \leftarrow$ SOLVERUPDATE$(\theta_t, \mathcal{B}_t)$ ▷ reuse cached rollouts
  $\mathcal{A}_{t+1,0} \leftarrow \mathcal{A}_{t,k}$
end for

Program representation and initialization. Bootstrap validates and scores seven hand-authored generator programs, one per mathematical domain, and applies the same archive admission rule. Each program returns a problem instance with a data-only verifier specification. Cached rollout groups supply the first Solver update and are repeated to fill the 32-prompt batch without sampling additional instances. Mutation begins at the next iteration. Each program reserves seeds 0–4 for validation and uses unused seeds for subsequent generation.

Archive cells. The archive has $7 \times 5 = 35$ cells, and a program's cell $b(P)$ pairs its mathematical domain with its required output form. The seven domains are algebra, geometry, number theory, discrete mathematics, applied mathematics, calculus, and precalculus. The five output forms are *decision*, requiring a Boolean answer; *search*, requiring a complete finite solution set; and *counting*, *optimization*, and *function*, which use expression equivalence. Output form is inferred from the

Table 3: Generator program evolution and Solver settings. KL denotes Kullback–Leibler divergence.

Component	Item	Value
Mutation	batch size / minimum / maximum	32 / 32 / 128
	temperature / top-p, both stages	0.45 / 0.85
	output-token cap, each stage	2048
	validation seeds / runs per seed	5 / 2
Solver	prompts / responses per prompt	32 / 10
	training selection priority	current fitness
	learning rate / weight decay	10^{-6} / 10^{-2}
	KL penalty coefficient	10^{-3}
	optimization epochs / update steps	1 / 256
	temperature / top-p	1.0 / 0.95
	prompt / response token cap	6000 / 5000

statement, checked against the verifier, and required to agree across validation seeds. After validation, the current Solver evaluates a generated program's fixed task family against each of the seven domain definitions through a one-token YES/NO probe, and the largest probability of YES selects the domain. Bootstrap programs instead use their hand-authored declarations. Cell assignments are cached and remain fixed during reassessment.

Two-stage generation and validation. The first stage withholds the destination cell along with the source code, and specifies a structural change, a parameterized child statement, and a finite answer specification. The second stage receives the parent's source with its cell label removed and returns an output mode together with a `build_instance(rng)` core. The output mode is `expression`, `boolean`, or `set`. The assembler constructs `generate(seed)` around an immutable statement template.

Static checks enforce the source and cross-check contracts. Before scoring, generated programs are run twice on each of five validation seeds in restricted, time-limited worker processes. For each seed, the reference answer must match the cross-check, and both runs must return identical fields. Across validation seeds, the family template, verifier mode, and output form must agree, with at least two distinct visible statements. Constant reference answers are allowed. Archive admission additionally checks behavioral, normalized-template, near-template, and structural duplication, and rejects detected copies of prompt examples.

Scoring and cached rollouts. Fitness estimation excludes overlong responses and uses the remaining response count in both the averages and the finite-sample correction. With one remaining response, learnability is set to zero. A group with no remaining responses is treated as a scoring failure. The bound in Proposition 1 and the estimation results in Appendix A.3 assume unfiltered rollouts. After successful scoring, the original rollout group is cached for Solver training.

Admission and mutation budget. Let $\rho(P)$ denote an incumbent's latest successfully measured fitness. For a validated, nonduplicate candidate P' with $\nu = b(P')$, the update Γ in Equation 5 updates each cell ω as

$$\mathcal{A}_{t,k+1}(\omega) = \begin{cases} P', & \omega = \nu,\ \mathcal{A}_{t,k}(\nu) = \varnothing, \\ P', & \omega = \nu,\ \mathcal{A}_{t,k}(\nu) \neq \varnothing,\ \hat{R}_Q(P'; \theta_t) > \rho(\mathcal{A}_{t,k}(\nu)), \\ \mathcal{A}_{t,k}(\omega), & \text{otherwise.} \end{cases} \tag{15}$$

Parents are sampled uniformly with replacement from occupied cells, including those with zero fitness. All parents in a batch are sampled before child admission, so newly admitted children are eligible as parents only in later batches. Each iteration uses 32–128 mutation slots in batches of 32. After each completed batch, mutation stops once at least one child yielding both correct and incorrect responses has been admitted or at least eight such candidates have been scored. Invalid children consume slots but receive no repair attempt.

Reassessment and instance allocation. Each iteration first evaluates one newly sampled instance per incumbent and caches its rollouts. A successful measurement overwrites the stored fitness, including zero values. A scoring failure preserves it but supplies no cached group for training. Zero fitness alone causes no eviction. After a successful measurement, reassessment removes the incumbent and attempts reinsertion. The cell remains empty if the admission checks reject reinsertion.

After mutation, the allocation pool consists of retained programs with cached groups whose scored instance yields both correct and incorrect responses. If this pool is empty, all retained programs with cached groups form the fallback pool. If the pool supplies fewer than 32 rollout groups, additional instances are sampled from its programs round-robin in descending order of current fitness to fill the shortfall. These instances are evaluated with the same Solver checkpoint, and their rollout groups are cached without changing program fitness or the primary instance's success rate.

Training-batch selection and Solver update. The training-program pool consists of programs retained in the archive after mutation, including newly admitted children. Programs with cached rollouts are eligible if their primary scored instance yields both correct and incorrect responses in the current evaluation. Eligible programs are ranked by their current fitness, and their cached rollout groups are appended up to a 32-prompt budget. Extra instances of an eligible program need not individually yield both correct and incorrect responses. If no eligible program supplies a group, the implementation falls back to available cached groups from retained programs without requiring both correct and incorrect responses. A nonempty underfilled batch is padded by cyclically repeating both instances and their cached responses. Rejected or displaced programs are excluded even if their rollout groups remain cached.

Solver training reuses the original cached groups of ten responses and applies the centered advantage in Equation 2. One epoch over a single 32-prompt mini-batch yields one global actor optimizer step with microbatch accumulation. The clipped actor objective includes KL regularization, and the optimizer uses weight decay.

C Evaluation and Comparison Protocols

C.1 Benchmarks and Evaluation

We trained R-Zero at both model scales. At each scale, we compare our reproduced result with the published result and report the complete row with the higher mathematical average in Table 1. This selects the published R-Zero result at 4B (Huang et al., 2026b), marked with ‡. At 8B, our reproduced R-Zero run has the higher mathematical average, and its peak checkpoint supplies the table row. The final-to-final comparison uses its final checkpoint at update 256. The Absolute Zero and G-Zero rows, marked with †, use scores reported in the R-Zero paper (Huang et al., 2026b) and the J-Zero paper (Chu et al., 2026), respectively. Unmarked rows report our evaluations of the base models, R_Q-Evolve, and R-Zero at 8B.

Evaluation protocol. For our evaluations in Table 1, Solver responses use greedy decoding with temperature 0, top-$p = 1$, and one generation per input row. Prompts request step-by-step reasoning and a final answer in `\boxed{}`, with multiple-choice final answers restricted to the option letter. The output limits are 4,096 tokens for mathematical reasoning and 8,192 for general reasoning.

Following R-Zero's evaluation code, AMC'23, AIME'24, and AIME'25 repeat each problem instance 32 times and average binary correctness over these greedy repetitions. Mathematical answers are extracted from the first `\boxed` expression and checked with Math-Verify (Kydlíček). Answers not accepted by this check are rechecked once by `gpt-4o`[1] at temperature 0.1.

Mathematical reasoning.

- **MATH-500** (Hendrycks et al., 2021; Lightman et al., 2024) contains 500 competition mathematics problems selected from MATH, covering seven mathematical subjects.
- **GSM8K** (Cobbe et al., 2021) contains grade-school word problems requiring multistep arithmetic reasoning.

[1] GPT-4o system card: `https://openai.com/index/gpt-4o-system-card/`.

- **AMC'23** uses problems from the 2023 AMC 12A and 12B competitions to evaluate high-school mathematical problem solving.
- **AIME'24** comprises integer-answer problems from the 2024 American Invitational Mathematics Examination.
- **AIME'25** uses the 2025 AIME problem set, providing another edition of advanced integer-answer competition problems.
- **Minerva** (Lewkowycz et al., 2022) contains 272 undergraduate-level STEM problems drawn from MIT OpenCourseWare.
- **OlympiadBench** (He et al., 2024) contributes its English, text-only subset of open-ended mathematical problems.

General reasoning.

- **SuperGPQA** (Du et al., 2026) tests graduate-level knowledge and reasoning across 285 disciplines through multiple-choice questions.
- **MMLU-Pro** (Wang et al., 2024) covers 14 subjects with knowledge and reasoning questions offering up to ten answer options.
- **BBEH** (Kazemi et al., 2025) comprises 23 tasks that extend the reasoning skills tested by BIG-Bench Hard with more difficult problems.

C.2 Timing and Curriculum Dynamics

Timing. We measure total wall-clock time from launching each training run to its completion after 256 Solver updates at 8B, including curriculum construction and Solver training. Both runs use the same server with eight NVIDIA RTX PRO 6000 GPUs, and R-Zero uses its published Challenger configuration. The totals are 25.25 hours for R_Q-Evolve and 85.67 hours for R-Zero.

Archive dynamics. Figure 5 tracks mathematical performance, archive occupancy, and candidate admissions at both scales. After the bootstrap update, snapshot i accompanies Solver update $i + 1$.

At 4B, the mathematical average rises to 51.34 at update 32, falls to 49.96 at 64, and reaches its highest value of 52.29 at the final checkpoint. Within the archive, refinement accounts for 100% and 98.9% of admissions in the last 96 updates at 4B and 8B, respectively. Program replacement continues as archive coverage stabilizes.

Distribution construction. The R_Q-Evolve windows in Figure 4 cover Solver steps 0–32, 97–128, and 225–255. R-Zero uses generation pools from rounds 1, 4, and 8. For R_Q-Evolve, problem statements are reconstructed from saved generator sources and seeds, including instances from mutation candidates recorded in the rollout logs. After stripping surrounding whitespace, we discard empty statements and retain the first occurrence of each statement within each R_Q-Evolve window and R-Zero round.

For both methods, `gpt-5.6-luna`[2] assigns domains using the same post-hoc classification prompt adapted from Omni-MATH (Gao et al., 2025). We apply the same frozen, statement-only rules to assign output forms. The joint grid includes only instances with exactly one domain labeled `high` confidence by the classifier and an assigned output form.

After excluding ambiguous assignments, the early, middle, and final windows contain 1,278, 1,424, and 1,691 classified instances for R_Q-Evolve, and 3,469, 5,565, and 5,722 for R-Zero, respectively. Figure 6 reports the cell shares within each of these samples. Each panel's shares sum to one, and effective-cell counts are computed from those shares as defined in the caption of Figure 4. This external analysis labeler is distinct from the domain assignment used during archive admission. Table 4 reports the filtering counts, with R_Q-Evolve input counts reconstructed from the saved programs and seeds using the original export and deduplication procedure. Two early-window instances fail reconstruction because their reference answers disagree with their cross-checks and are excluded before the input count. All logged instances in the middle and final windows are reconstructed.

[2]GPT-5.6 system card: https://deploymentsafety.openai.com/gpt-5-6.

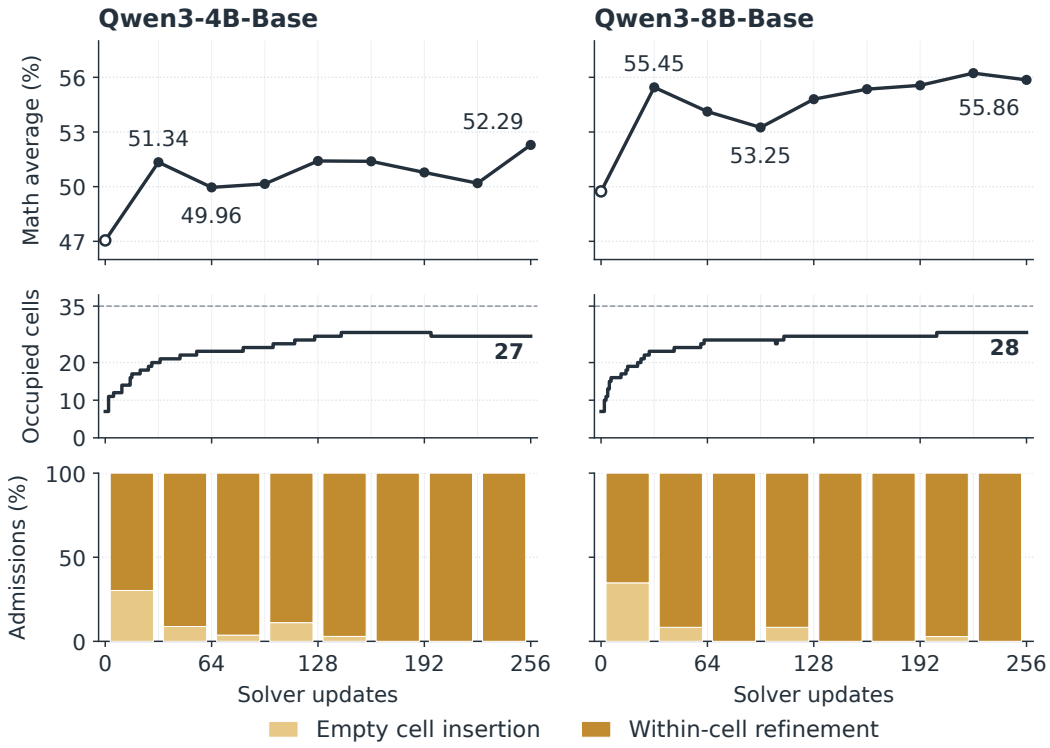


Figure 5: Archive dynamics and mathematical performance by initialization model. Hollow markers denote the base models at step 0, and dashed lines mark the 35-cell capacity. Admission shares normalize empty cell insertions and within-cell refinements by their combined count in each 32-update interval, excluding bootstrap seeds. Refinement replaces an incumbent with a different program.

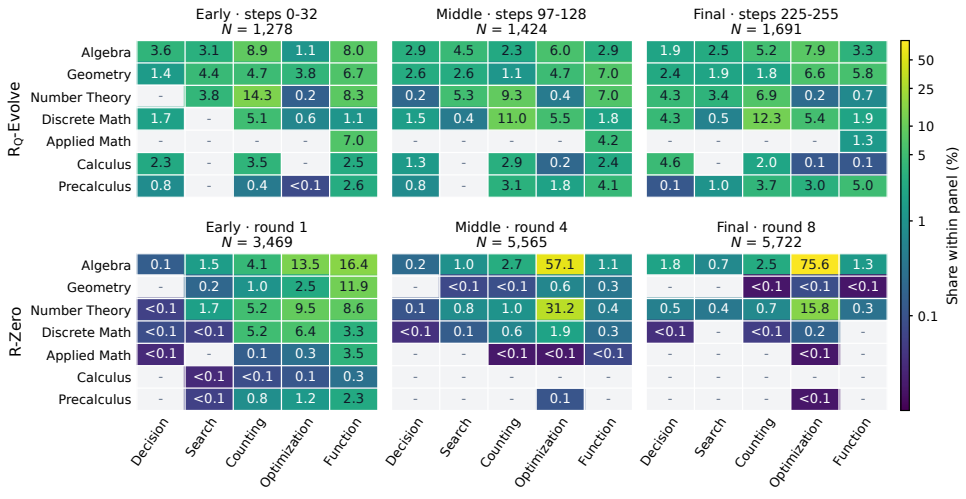


Figure 6: Domain–output-form distributions across training. Cell values are percentages of the N classified instances in each panel. A dash denotes zero mass and < 0.1 a nonzero percentage below the displayed precision. The color scale is logarithmic.

Table 4: Filtering for the generated-instance distributions. Input counts follow deduplication and precede classification exclusions. R_Q-Evolve counts are reconstructed from saved programs and seeds. Excluded equals Input minus Classified, with percentages relative to Input.

Method	Window	Input	Excluded	Excluded (%)	Classified
R_Q-Evolve	Early	1,379	101	7.32	1,278
	Middle	1,556	132	8.48	1,424
	Final	1,848	157	8.50	1,691
R-Zero	Round 1	4,079	610	14.95	3,469
	Round 4	5,799	234	4.04	5,565
	Round 8	6,242	520	8.33	5,722

C.3 Component Variants

All configurations leave the archive grid and two-stage mutation unchanged. The No Refresh variant repeatedly runs each generator program with seed 0 and samples new Solver responses to that fixed instance. The No U variant sets the token-entropy factor to one, so that archive comparison, training priority, and extra-instance allocation use learnability. The Random variant leaves validation, domain–output-form cells, and uniform parent sampling unchanged. It inserts into empty cells, replaces an occupied cell with probability 0.5, and randomizes training priority and extra-instance allocation, with fitness measured for logging but not controlling these decisions.

All configurations reach update 256, and checkpoints are evaluated every 32 updates. Unlike Table 1, which reports R_Q-Evolve at update 256, Table 2 and the ablation panel of Figure 1 report each variant at its checkpoint with the highest mathematical average within 256 updates. For the full 4B R_Q-Evolve run, the two criteria select the same checkpoint.

D Case Study

Table 5 follows three generator programs admitted at iterations 101, 133, and 170 of the 4B run. Each later program is a direct mutation of the preceding one. All three sample n uniformly from $\{1, \ldots, 1000\}$ and enumerate its positive divisors. Recorded seed 32 produces $n = 80$ in every program. The shared sampler is:

```
rng = random.Random(seed)
n = rng.randint(1, 1000)
D = [d for d in range(1, n+1) if n % d == 0]
```

The table condenses the original filtering, summation, and pair-enumeration logic, omitting rendering and cross-checks. `isqrt` denotes the integer square-root function `math.isqrt`.

From individual divisors to relations between divisors. For $80 = 2^4 \cdot 5$, the positive divisors are $\{1, 2, 4, 5, 8, 10, 16, 20, 40, 80\}$. The parent selects the perfect squares $\{1, 4, 16\}$ and returns their sum, 21. The first mutation squares every divisor before summing, giving

$$\sum_{d|80} d^2 = (1 + 4 + 16 + 64 + 256)(1 + 25) = 8866.$$

The second mutation counts unordered pairs of distinct divisors whose product is a perfect square. For each prime, the exponents in the two divisors must have the same parity. Grouping the divisors by this condition gives $\{1, 4, 16\}$, $\{2, 8\}$, $\{5, 20, 80\}$, and $\{10, 40\}$, hence

$$\binom{3}{2} + \binom{2}{2} + \binom{3}{2} + \binom{2}{2} = 8.$$

Thus the lineage progresses from a condition on individual divisors, through a transformation of their values, to a condition relating pairs of divisors.

Table 5: One 4B lineage with problem parameter $n = 80$. `D` lists the positive divisors of n. Red text highlights the corresponding task-family description and code.

Generated instance	Generator program (condensed)
Parent: filter divisors	
Sum the divisors of 80 **that are perfect squares**. **Answer:** 21.	`answer = sum(d for d in D if isqrt(d)**2 == d)`
Mutation 1: transform values	
Sum **the squares** of all divisors of 80. **Answer:** 8,866.	`answer = sum(d*d for d in D)`
Mutation 2: relate divisor pairs	
Count **unordered pairs of distinct divisors** of 80 whose **product is a perfect square**. **Answer:** 8.	`answer = sum( isqrt(a*b)**2 == a*b for i, a in enumerate(D) for b in D[i+1:])`

Fitness along the lineage. The parent occupied the number theory/function cell from iteration 101 to 132, and its last six snapshots each recorded $\hat{R}_Q = 0$ with $\hat{s}_z = 1.00$. The first mutation, which replaced it in that cell at iteration 133, likewise ended with eight consecutive snapshots at $\hat{R}_Q = 0$ and $\hat{s}_z = 1.00$. The two ancestors averaged success rates of 0.93 and 0.92 over their tenures. The second mutation entered the number theory/counting cell at iteration 170 with $\hat{s}_z = 7/9 \approx 0.78$ (seven of nine responses remaining after filtering were correct) and $\hat{U}_z = 0.231$, yielding $\hat{R}_Q = 0.0450$, the highest admission fitness in the lineage. It recorded a mean success rate of 0.54 and a mean token entropy of 0.153 across its 50 snapshots, compared with a mean token entropy of 0.061 for the first mutation. Of its 33 zero-fitness snapshots, 22 had $\hat{s}_z = 1.00$ and 11 had $\hat{s}_z = 0.00$. These snapshots score different sampled instances against changing Solver checkpoints.

E Omni-MATH-2 Evaluation

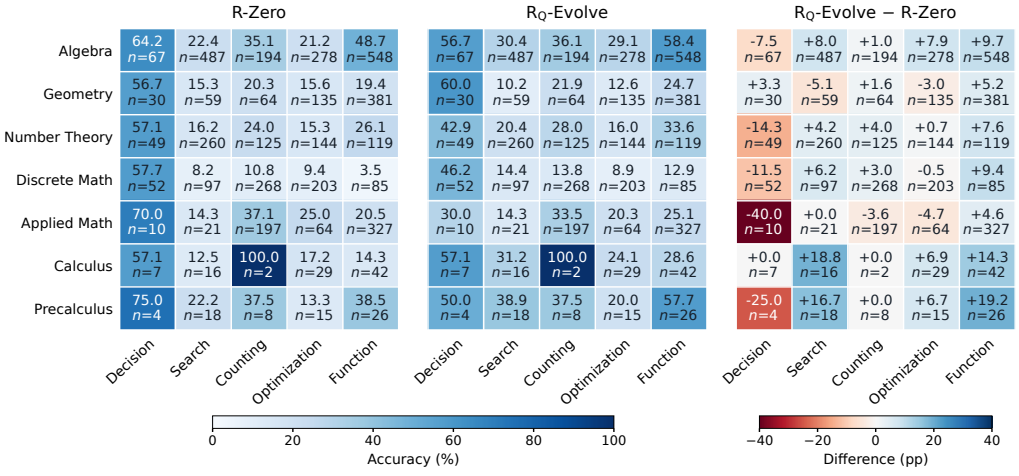


Figure 7: Cellwise Omni-MATH-2 accuracy on the mapped subset for the 8B checkpoints at step 256. The difference is R_Q-Evolve minus R-Zero in percentage points. Each cell reports its sample size n. Instances with multiple domain labels contribute to multiple cells.

Scope, data, and exclusions. We evaluate Qwen3-8B-Base and the 8B checkpoints of R_Q-Evolve and R-Zero at updates 32, 128, and 256 on a snapshot of `martheballon/Omni-MATH-2` (Ballon

et al., 2026) with greedy decoding and one response per instance. The snapshot revision is recorded in the accompanying evaluation metadata. Of 4,428 source rows, 4,174 carry no author-supplied exclusion tag, and we exclude four known defects from scoring: blank reference answers at source IDs 4168 and 4416, and missing required diagrams at IDs 2846 and 2913. Both final checkpoints have complete judgments for the remaining 4,170 rows, compared with the 4,181 filtered instances reported by Ballon et al. (2026). All models use identical prompts and tokenizer templates with a response limit of 4,096 tokens. Answers are checked by `gpt-5-mini-2025-08-07`[3] with medium reasoning effort and a shared equivalence-checking prompt adapted from the Omni-MATH-2 protocol.

Cell assignment. For the cellwise comparison of the final checkpoints in Figure 7, published domain labels are not mutually exclusive, and output forms use the frozen statement-only classifier rather than official Omni-MATH-2 labels. The 3,534 mapped rows contribute 4,431 cell memberships. The remaining 636 rows are included in overall accuracy but excluded from the cellwise analysis. All 35 cells are shown, with sizes from 2 to 548, and eight have fewer than 20 rows.

[3]GPT-5 system card: https://deploymentsafety.openai.com/gpt-5.