ANALYZING REFINEMENT IN DATA-FREE EVOLUTION OF CODE-GENERATION LLMS

Anonymous authors
Paper under double-blind review

ABSTRACT

Data-free evolution methods reduce reliance on human-curated data: a proposer generates tasks, and a solver learns to solve them. The solver's performance is determined by the quality of its generated tasks, which is often improved through a refinement operation that filters or selects important tasks for solver training. In this paper, we investigate when refinement actually helps the solver, from two specific perspectives. First, from the solver-gradient perspective: the refinement affects the solver only through the alignment between the solver gradient and the downstream task performance. Then, we investigate the proposer-support perspective: the refinement can only select tasks that the proposer generates, so the solver performance is limited by the capability of the proposer. Empirically, on code generation benchmarks, existing refinement operators do not significantly outperform the non-refinement baseline. Furthermore, using a stronger proposer can produce more diverse tasks, which tends to improve the solver's performance even without refinement.

1 INTRODUCTION

Large language models (LLMs) have achieved strong performance across many task domains, including instruction following (Ouyang et al., 2022), mathematical reasoning (Shao et al., 2024), and code generation (Guo et al., 2024; Zhu et al., 2024). Specifically, reinforcement learning from verifiable rewards (RLVR) provides an effective post-training paradigm for LLMs: an LLM generates an output, a verifier checks correctness, and the verification signal is used as the reward without learning a separate reward model (Lambert et al., 2024). However, this paradigm commonly relies on human-curated training data, such as problems, solutions, labels, or verifier specifications. As LLM capabilities improve, recent data-free evolution methods reduce this dependence by generating their training data inside the learning loop (Zhao et al., 2025; Huang et al., 2025; Chen et al., 2025a; Kuba et al., 2025). Such a system manufactures its own training experience, without the requirement of human labour.

Code generation is a popular instance of data-free evolution (Zhao et al., 2025; Jana et al., 2026). In this domain, the loop can be formulated as a proposer–solver loop. Given a topic, a proposer generates a coding task consisting of a problem statement, a reference solution, and unit tests. A solver is trained on the generated tasks by producing solutions from problem statements, while the unit tests remain hidden and provide the verification reward. Existing variants may use multiple proposers, solvers, or model populations, but they share the same dependence on the model-generated training dataset (Chen et al., 2025b; Peng et al., 2026; Jana et al., 2026; Castanyer et al., 2026). Because generated tasks can be invalid, redundant, poorly matched to the current solver, or concentrated on a narrow set of topics, existing systems usually refine the generated task set before solver training. Refinement aims to improve this task set: it selects or filters the generated tasks, keeping those expected to train the solver best. Two representative criteria are learnability (Zhao et al., 2025; Huang et al., 2025), which selects tasks of intermediate difficulty for the current solver, and IFD (Li et al., 2024), which selects tasks by the difficulty of their reference solution.

Prior work studies how LLMs behave and where they fail, including diversity collapse, curriculum collapse, and support limits in on-policy self-improvement (Li et al., 2026a; Mishra, 2026; Wu et al., 2025; Yue et al., 2025; Chae et al., 2025; Li et al., 2026b; Liu et al., 2026), but rarely analyses how

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.

the refinement operation affects the performance of data-free evolution methods. In this paper, we provide the analysis as follows:

- **Solver-gradient.** Refinement affects the solver only through the alignment between the solver gradient and the downstream task performance.
- **Proposer-support.** Refinement can only select tasks generated by the proposer, so the proposer's support fundamentally limits the achievable solver performance.

Empirically, on code generation benchmarks, we compare refinement operators against a non-refinement baseline. We also show that using the relatively stronger proposer, which can produce a more diverse task set, can improve solver performance even without refinement techniques. Our results show that refinement alone is not reliably beneficial: across every model and operator we test, the gap to non-refinement is not statistically distinguishable from noise at our sample size, and its sign varies by model. Proposer-side diversity points in the same direction in both comparisons we run, though only the larger one approaches significance.

2 Related Work

Data-free Evolution Methods. Recent data-free evolution methods train LLMs on supervision generated inside a proposer–solver loop, and all condition the proposer's reward on the solver: Absolute Zero Reasoner rewards the proposer by the solver's solve rate on the proposed task (Zhao et al., 2025); R-Zero (Huang et al., 2025), self-questioning LLMs (Chen et al., 2025a), language self-play (Kuba et al., 2025), and their multi-agent and population variants (Chen et al., 2025b; Peng et al., 2026; Jana et al., 2026; Castanyer et al., 2026) share this closed-loop design, together with multi-round curricula in which the proposer and solver are updated repeatedly against each other. We instead study a decoupled special case of this loop: the proposer reward depends only on well-formedness, topic adherence, and reference-solution self-consistency, never on solver performance, and the proposer is trained to convergence before the solver is touched at all. This isolates the refinement operator from a proposer that is simultaneously adapting to the solver, so any effect we measure can be attributed to refinement rather than to a moving proposer, at the cost of scope: this work characterizes this decoupled, single-round setting, not the closed-loop, multi-round systems above.

Refinement Operations. We use refinement for operations that select or filter the generated training dataset. Related data-dependent strategies include curriculum (Bengio et al., 2009) and self-paced learning (Kumar et al., 2010), and example-valuation methods based on influence functions (Koh & Liang, 2017), Data Shapley (Ghorbani & Zou, 2019), coreset selection (Sener & Savarese, 2018), dataset cartography (Swayamdipta et al., 2020), and example forgetting (Toneva et al., 2019). In data-free evolution, refinement appears as validity checks, unit-test gates, near-duplicate filtering, difficulty targeting, and dropping mastered problems (Zhao et al., 2025; Huang et al., 2025; Chen et al., 2025a; Kuba et al., 2025). These choices are usually evaluated inside a complete system, which makes their solver-level effect hard to isolate. We mainly compare two significant refinement operators: learnability (Zhao et al., 2025; Huang et al., 2025) and IFD (Li et al., 2024).

Existing Related Analysis. Several analyses study failure modes of data-free evolution methods, including diversity illusion (Li et al., 2026a), curriculum collapse (Mishra, 2026), and vocabulary dropout (Dineen et al., 2026); for the proposer support, the invisible-leash analysis argues that on-policy RLVR concentrates probability mass without expanding output support (Wu et al., 2025), with related results on bounded self-play and reasoning limits (Yue et al., 2025; Chae et al., 2025; Li et al., 2026b; Liu et al., 2026). These characterize potential failure modes in the proposer–solver loop, rather than whether the refinement is beneficial. Additionally, Liu et al. (2026) argue that self-play evolves only when the pipeline ensures learnable information gain; we differ by isolating the refinement operator from the rest of the pipeline and asking, empirically and at the level of the solver gradient, whether that operator alone helps. We analyze refinement directly through two complementary lenses: the solver gradient, which characterizes how the refined dataset influences the solver, and proposer support, which characterizes the limitations of the proposer. The latter adapts the theoretical results (Wu et al., 2025) to disentangle refinement from proposer-side changes; the underlying support-preservation mechanism is itself an instance of the entropy-sharpening effects

of on-policy RL (Yue et al., 2025; Chae et al., 2025), so Proposition 2 is an application of this mechanism to the proposer–solver setting.

3 PRELIMINARIES

Algorithm 1 Data-free evolution with a refinement operator (single round).

Require: base model θ^0; topic distribution $p_{\mathcal{Z}}$; refinement operator F;
1: $\theta_P, \theta_S \leftarrow \theta^0$ ▷ Initialize proposer and solver
2: $\theta_P \leftarrow \arg\max_{\theta_P} J_P(\theta_P)$ ▷ Training proposer via Equation (1)
3: $D \leftarrow \{\tau : R_P(\tau; z) = 1 | z \sim p_{\mathcal{Z}}, \tau = (x, y_{\text{ref}}, U) \sim P_{\theta_P}(\cdot \mid z)\}$ ▷ Collect valid task sets
4: $D' \leftarrow F(D)$ ▷ Apply refinement (e.g., Definition 3 and Definition 4)
5: $\theta_S \leftarrow \arg\max_{\theta_S} J_S(\theta_S; D')$ ▷ Training solver via Equation (2)
6: **return** θ_S

Data-free evolution can be formalized as a proposer–solver loop, including a proposer that generates tasks, a refinement operator that selects among them, and a solver trained on the result. Overall, the pseudo code of the algorithm is summarized in Algorithm 1.

Proposer objective. Let $p_{\mathcal{Z}}$ be a distribution over a topic space $\mathcal{Z}$. For a topic $z \sim p_{\mathcal{Z}}$, a proposer P_{θ_P} with parameters θ_P samples a coding task

$$\tau = (x, y_{\text{ref}}, U) \sim P_{\theta_P}(\cdot \mid z), \qquad U = \{u_j\}_{j=1}^{m},$$

where x is the problem statement, y_{ref} is a reference solution, and U is a set of m unit tests. Each u_j is a predicate that returns 1 if a candidate solution passes and 0 otherwise. The proposer objective and reward are

$$J_P(\theta_P) = \mathbb{E}_{\substack{z \sim p_{\mathcal{Z}} \\ \tau \sim P_{\theta_P}(\cdot|z)}} \left[R_P(\tau; z) \right], \quad R_P(\tau; z) = \mathbf{1}[\text{valid}(x, U, z)] \prod_{u_j \in U} \mathbf{1}[u_j(y_{\text{ref}}) = 1] \quad (1)$$

where $\mathbf{1}[\cdot]$ is the indicator function, and $\text{valid}(x, U, z) = 1$ means that x is a coding problem on topic z and U is a nonempty executable test set for x. Specifically, for the valid function, we adopt a strong LLM, DeepSeek-v4-pro (Xu et al., 2026), as the validator that verifies the topic condition. Thus $R_P(\tau; z) = 1$ only when the task is well-formed, and the reference solution passes all proposer-generated tests. The objective rewards validity alone and does not promote diversity: a proposer that degenerates to a single valid task per topic still maximizes $J_P(\theta_P)$. Diversity is instead inherited from the stochasticity of the base model's sampling distribution within each topic and from the topic distribution $p_{\mathcal{Z}}$ across topics. Let $D = \{\tau : R_P(\tau; z) = 1\}$ be the set of valid tasks generated by the proposer. Given the topic z, let $D_z \subseteq D$ denote the generated tasks conditioned on z. For each topic z, we denote the proposer task support by $\text{supp}_{\theta_P}(z)$. Since a softmax proposer assigns strictly positive probability to every finite token sequence, the exact support is the entire task space and carries no information. We therefore follow the empirical-support notion of Wu et al. (2025) and define $\text{supp}_{\theta_P}(z) = \{\tau : P_{\theta_P}(\tau \mid z) > \epsilon\}$ as the effective support at a threshold $\epsilon > 0$ set by the finite sampling budget: a task below the threshold is effectively never realized when collecting D.

Refinement operator. Given the generated task set D, a refinement operator F maps D to a refined task set $D' = F(D) \subseteq D$. The operator F is a fixed rule that selects or filters the proposer-generated tasks before solver training; it has no learned parameters. At the topic level, we denote the corresponding refined task set by $D'_z = F(D_z)$. The operator F may include unit-test gates, near-duplicate filtering, difficulty targeting, or dropping mastered problems. Topic sampling and task-set accumulation instead change the generated task set D available to the refinement operator.

Solver objective. The solver is trained on D'. For each $\tau = (x, y_{\text{ref}}, U) \in D'$, a solver S_{θ_S} with parameters θ_S observes only the problem statement x. The unit tests U and reference solution y_{ref} are hidden from the solver and are used only by the verifier. The solver samples $y \sim S_{\theta_S}(\cdot \mid x)$ and maximizes

$$J_S(\theta_S; D') = \mathbb{E}_{\substack{\tau \sim D' \\ y \sim S_{\theta_S}(\cdot|x)}} \left[R_S(x, y, U) \right], \quad R_S(x, y, U) = \prod_{u_j \in U} \mathbf{1}[u_j(y) = 1]. \quad (2)$$

Evaluation performance. Let $M(\theta_S) \in \mathbb{R}$ denote the downstream evaluation performance functional of the solver (e.g., held-out benchmark evaluation). The functional M is not part of R_P, R_S, J_P, or J_S, and is unknown during training. In practice, M is the solver's expected pass rate (e.g., pass@1) on the held-out benchmark.

Gradient signal and performance alignment. To analyze how D' affects M, we define the solver-gradient signal $G(D'; \theta_S)$ and its performance alignment $A(D'; \theta_S, M)$ with M. The score-function estimator corresponds to REINFORCE (Williams, 1992) and the policy-gradient formulation of Sutton et al. (1999). R_S can be replaced by the group-relative advantage estimator introduced by GRPO (Shao et al., 2024).

Definition 1 (Task-induced policy-gradient)**.** Given the refined task set D', the solver-gradient is defined as

$$G(D'; \theta_S) = \frac{1}{|D'|} \sum_{\tau \in D'} g(\tau; \theta_S), \quad g(\tau; \theta_S) = \mathbb{E}_{y \sim S_{\theta_S}(\cdot|x)}[R_S(x, y, U) \nabla_{\theta_S} \log S_{\theta_S}(y \mid x)],$$

where $g(\tau; \theta_S)$ is the expected score-function update direction for the task τ.

Definition 2 (Performance alignment)**.** Given the downstream performance functional M, we define the performance alignment of the refined task set D' as

$$A(D'; \theta_S, M) = \frac{1}{|D'|} \sum_{\tau \in D'} \alpha(\tau; \theta_S, M), \quad \alpha(\tau; \theta_S, M) = \langle \nabla_{\theta_S} M(\theta_S),\, g(\tau; \theta_S) \rangle,$$

where $\alpha(\tau; \theta_S, M)$ is the performance alignment for the task τ.

Refinement criteria in prior work differ in the signal they read from a task. One family scores a task as a function of the solver's reward: the solver is rolled out on the task, and the resulting pass rate measures how difficult the task is for the current solver (Zhao et al., 2025; Huang et al., 2025). Another family instead scores the reference solution: the task is judged by how difficult the solver finds y_{ref} under its own likelihood, requiring no rollout or test execution (Li et al., 2024). We adopt one representative operator from each family, learnability and IFD respectively, so that our comparison covers both reward-based and reference-based signals. For a given task $\tau = (x, y_{\text{ref}}, U)$, we define the two refinement operators as follows. A higher $\text{learn}(\tau)$ means a task of intermediate difficulty for the current solver. A higher $\text{IFD}(\tau)$ means x provides less help in producing y_{ref}, indicating a harder task.

Definition 3 (Learnability)**.** The learnability of a task τ is defined as $\text{learn}(\tau) = p_\tau(1 - p_\tau)$, where p_τ is pass rate of the solver S_{θ_S} on τ.

Definition 4 (Instruction-Following Difficulty (IFD))**.** With reference-solution tokens $y_{\text{ref}} = (w_1, \ldots, w_N)$, the IFD of a task τ is defined as $\text{IFD}(\tau) = \frac{\sum_{i=1}^{N} \log S_{\theta_S}(w_i | x, w_{<i})}{\sum_{i=1}^{N} \log S_{\theta_S}(w_i | w_{<i})}$.

4 Refinement Analysis

We analyze refinement from two different perspectives. The solver-gradient characterizes the update a refined task set induces (Section 4.1), while the proposer-support bounds which tasks refinement can select (Section 4.2). Both perspectives are build on the first-order Taylor expansion of the performance functional (Remark 1) and the proposer support-preservation result of Wu et al. (2025).

4.1 Solver-gradient Analysis

We first state a local first-order expansion of M in the following Lemma.

Remark 1 (Performance Difference Expansion). For the update $\theta_S^+ = \theta_S + \eta\, G(D'; \theta_S)$ with step size $\eta > 0$, the performance difference satisfies

$$\Delta M(D') = M(\theta_S^+) - M(\theta_S) = \eta\, \langle \nabla_{\theta_S} M(\theta_S),\, G(D'; \theta_S) \rangle + O(\eta^2).$$

Applying Definition 2 into Remark 1, we have Proposition 1 (proof in Appendix A.1).

Proposition 1 (Performance Difference Alignment)**.** *The first-order performance difference is*

$$\Delta M(D') = \eta\, A(D'; \theta_S, M) + O(\eta^2),$$

so it depends on the refined task set D' only through its performance alignment $A(D'; \theta_S, M)$.

Proposition 1 reveals that the first-order effect of refinement runs entirely through the refined task set's alignment $A(D'; \theta_S, M)$. We compare refined and unrefined sets in Corollary 1.

Corollary 1 (Refinement Comparison)**.** *The performance difference between the refined task set $D' = F(D)$ and the unrefined task set D satisfies*

$$\Delta M(D') - \Delta M(D) = \eta\big(A(D'; \theta_S, M) - A(D; \theta_S, M)\big) + O(\eta^2),$$

refinement improves performance over the unrefined set if and only if $A(D'; \theta_S, M) > A(D; \theta_S, M)$.

By Corollary 1 (proof in Appendix A.2), refinement helps when it improves the performance alignment. However, the downstream performance functional M is usually unknown which means we cannot access the downstream tasks or real production scenarios. For a refinement operator, we cannot directly evaluate A. In this case, we show that learnability (Definition 3) is an acceptable choice, as it is expectation-optimal under a mild assumption. We provide the formal statement and proof in Appendix A.3.

Remark 2 (Learnability Analysis). Assume the alignment between a task gradient's direction and the performance gradient is independent of the learnability and positive in expectation. Then learnability is expectation-optimal to maximize the expected performance alignment.

This optimality is conditional: as $\mathbb{E}[A(D')] = \bar{a} \cdot \frac{1}{|D'|} \sum_{\tau \in D'} \mathrm{learn}(\tau)$, learnability optimizes selection within the proposer's pool but cannot raise its mean alignment $\bar{a}$. Its lack of a distinguishable empirical gain (Section 5.2) is therefore consistent with a small or near-homogeneous $\bar{a}$, a self-generated pool carrying little signal aligned with downstream generalization.

4.2 Proposer-support Analysis

We next investigate whether refinement can expand the proposer-support available to the solver. Recall that $\mathrm{supp}_{\theta_P}(z)$ denotes the effective support: for a softmax proposer, the exact support is always the full task space, so the statements below are non-vacuous only at the effective-support level. We adopt the support preservation (Wu et al., 2025) into our context as follows:

Lemma 1 (Proposer support preservation)**.** *For any proposer parameter θ_P obtained by on-policy RLVR updates from a base proposer θ_P^0, we have* $\mathrm{supp}_{\theta_P}(z) \subseteq \mathrm{supp}_{\theta_P^0}(z), \quad \forall z \in \mathcal{Z}.$

The mechanism is that on-policy updates reinforce only sampled tasks: a task below the sampling threshold of the base proposer is never drawn, hence never amplified (Wu et al., 2025; Yue et al., 2025). A fixed proposer ($\theta_P = \theta_P^0$) attains equality, whereas replacing the base model changes the proposer-support. Then, we have the support limitation of refinement as follows:

Proposition 2 (Support limitation of refinement)**.** *Under Lemma 1, for any $z \in \mathcal{Z}$, we have*

$$D'_z \subseteq D_z \subseteq \mathrm{supp}_{\theta_P}(z) \subseteq \mathrm{supp}_{\theta_P^0}(z).$$

As implied by Proposition 2, refinement can not expand the proposer-support, and tasks outside the base proposer support $\mathrm{supp}_{\theta_P^0}(z)$ cannot appear in D'_z.

Together, the analyses give a simple condition for useful refinement: the refined task set must induce a positive solver improvement, and the proposer must generate useful tasks for refinement to select.

5 Experiments

5.1 Experiment Setting

Training framework and models. We implement all experiments with Unsloth (Han et al., 2023) and TRL (von Werra et al., 2020), and train the models using LoRA (Hu et al., 2022). We primarily experiment on the Qwen3 family (Yang et al., 2025). Additional results from other model families, Gemma-3 (Team, 2025) and Llama-3.1 (Team, 2024), are provided in Appendix D. By default, the proposer and solver both use the same base model.

models (proposer/solver)	LiveCodeBench-v6	LiveCodeBench-v5	BigCodeBench	HumanEval+	MBPP+	avg
Qwen3-1.7B	22.6±0.3	18.2±0.5	30.6±0.4	71.6±0.5	67.2±0.0	42.0±0.1
human-curated data	**24.0**±0.6	**21.4**±0.6	29.9±0.4	**71.8**±1.3	67.1±0.8	42.8±0.1
non-refinement	**24.0**±0.8	21.0±0.4	**31.2**±0.4	71.1±1.4	**68.2**±0.8	**43.1**±0.7
learnability	23.6±1.2	20.9±1.4	31.0±0.9	71.5±1.1	67.5±0.7	42.9±0.4
IFD	23.2±1.9	20.3±1.7	**31.2**±1.4	70.9±1.5	67.6±1.7	42.6±1.5
Qwen3-4B	30.3±0.5	26.8±0.3	42.2±0.3	84.0±0.2	75.8±0.4	51.8±0.1
human-curated data	32.4±0.6	31.2±1.2	45.4±0.7	85.4±0.3	77.0±0.6	54.3±0.5
non-refinement	**33.5**±1.6	33.5±2.1	43.7±0.3	85.8±0.5	76.4±0.5	54.6±0.8
learnability	33.4±1.0	**35.0**±1.1	**45.8**±0.6	**86.6**±0.3	77.1±0.2	**55.6**±0.5
IFD	33.0±0.2	34.4±0.2	45.4±1.0	85.7±0.2	**77.3**±0.3	55.1±0.2
Qwen3-8B	30.1±0.4	26.5±0.6	44.5±0.3	83.7±1.1	77.3±0.4	52.4±0.2
human-curated data	33.2±1.4	33.7±3.0	47.2±0.7	86.5±1.4	78.7±0.6	55.9±1.3
non-refinement	**35.0**±1.0	**37.1**±2.3	**48.0**±0.9	87.0±0.3	**78.9**±0.7	**57.2**±0.9
learnability	34.5±1.1	36.2±1.4	46.7±0.4	86.3±0.5	78.4±0.2	56.4±0.5
IFD	34.8±0.8	35.7±1.7	47.3±0.8	**87.2**±0.6	78.8±0.5	56.8±0.7
Qwen3-14B	33.0±0.4	32.8±1.6	46.1±0.2	87.4±0.6	79.0±0.4	55.7±0.3
human-curated data	36.2±0.8	39.0±1.2	**49.5**±0.7	88.7±0.1	79.8±0.3	58.7±0.5
non-refinement	35.6±0.7	37.7±1.3	48.9±0.6	**89.0**±0.4	80.3±0.4	58.3±0.5
learnability	36.2±0.2	38.4±0.9	49.0±0.4	**89.0**±0.4	**80.4**±0.1	58.6±0.3
IFD	**37.0**±1.2	**40.3**±2.4	49.4±0.6	88.8±0.9	80.1±0.4	**59.1**±0.7
DeepSeek-R1-Distill-Qwen-7B	28.6±0.6	27.0±0.5	**31.2**±0.0	78.4±0.2	**70.1**±0.9	**47.1**±0.1
human-curated data	30.9±0.3	28.7±0.6	27.4±0.4	**78.6**±0.9	68.5±0.2	46.8±0.1
non-refinement	27.3±6.1	25.7±6.7	27.3±2.6	74.3±6.5	67.1±3.8	44.4±5.1
learnability	**31.0**±0.5	**28.8**±1.9	28.9±1.0	77.9±1.6	68.6±1.4	47.0±1.1
IFD	30.0±1.6	27.7±2.2	27.0±1.2	75.6±2.1	67.9±1.0	45.6±1.5

Table 1: Solver performance (pass@1) on the five code benchmarks; mean±std over three seeds, best performance in **blue**. Section 5.3 reports interval estimates for the operator-vs-non-refinement differences.

Training pipeline. Because our goal is to isolate how refinement and the proposer affect solver performance, we run the proposer–solver loop for only a single round, as summarized in Algorithm 1. Specifically, we first train the proposer (Equation (1)), using algorithm tags as the topics (Codeforces, 2026; Li et al., 2023). The trained proposer then produces a dataset of valid tasks (e.g., 500 tasks). We consider two such operators, learnability (Zhao et al., 2025; Huang et al., 2025) and IFD (Li et al., 2024): for a task τ, learnability scores the task by $p_\tau(1-p_\tau)$ (Definition 3), whereas IFD scores it by the instruction-following difficulty (IFD) of its reference solution (Li et al., 2024). Finally, we train the solver (Equation (2)) on either the unrefined or the refined task set, and evaluate the resulting solver.

Benchmark and Evaluation. We report the solver's pass@1 on five code-generation benchmarks, including LiveCodeBench-v5/v6 (Jain et al., 2025), BigCodeBench (Zhuo et al., 2025), HumanEval+, and MBPP+ (Liu et al., 2023). We use the unbiased pass@k estimator of Chen et al. (2021) at $k = 1$, computed from 8 samples per problem; all reported numbers use this same estimator. Every experiment is run over three seeds, where each seed is a fully independent replicate: the proposer, task collection, refinement, and solver training all run at that seed. We provide more experiment details in Appendix B.

5.2 Experiment Results

Performance comparison. We first investigate whether training on generated tasks helps, and whether those tasks can stand in for human-curated tasks. We train each solver on proposer-generated data without refinement, and compare against a matched set of human-curated data (Prime Intellect, 2025). As shown in Table 1, training on the generated data is competitive with the human-curated data across the four Qwen3 sizes. Specifically, the generated data exceeds the human-curated data on LiveCodeBench-v5 at Qwen3-4B and Qwen3-8B, showing 33.5 vs 31.2 and 37.1 vs 33.7, respectively. Overall, the data-free method can achieve performance competitive with human-curated data.

Refinement comparison. We investigate how different refinement operators affect performance, comparing learnability (Zhao et al., 2025; Huang et al., 2025) and IFD, which we instantiate with the instruction-following difficulty (IFD) of the reference solution (Li et al., 2024). We retrain each solver

proposer	operator	pass rate	learnability metric	IFD metric
Qwen3-1.7B	non-refinement	0.53±0.10	0.078±0.003	1.00±0.05
	learnability	0.53±0.05	0.156±0.007	0.98±0.02
	IFD	0.53±0.10	0.074±0.006	1.21±0.09
Qwen3-4B	non-refinement	0.64±0.07	0.064±0.007	1.01±0.05
	learnability	0.58±0.06	0.127±0.013	0.99±0.02
	IFD	0.60±0.08	0.072±0.003	1.24±0.08
Qwen3-8B	non-refinement	0.58±0.01	0.072±0.001	1.31±0.09
	learnability	0.55±0.01	0.145±0.003	1.23±0.03
	IFD	0.60±0.03	0.070±0.007	1.64±0.17
Qwen3-14B	non-refinement	0.72±0.02	0.063±0.007	0.84±0.03
	learnability	0.63±0.01	0.127±0.015	0.86±0.04
	IFD	0.64±0.02	0.081±0.008	0.99±0.03

Table 2: Training dataset refinement measurement, including the base-solver pass rate, the learnability metric, and the IFD metric over three seeds.

solver	proposer	LiveCodeBench-v6	LiveCodeBench-v5	BigCodeBench	HumanEval+	MBPP+	avg
Qwen3-1.7B	Qwen3-1.7B	24.0±0.8	21.0±0.4	31.2±0.4	71.1±1.4	**68.2**±0.8	43.1±0.7
	Qwen3-8B	**24.2**±0.9	**21.3**±1.5	**31.7**±0.3	**72.0**±1.5	**68.2**±0.2	**43.5**±0.5
	Llama-3.1-8B-Instruct	23.7±0.7	19.8±1.1	30.3±0.3	71.1±0.7	67.2±0.5	42.4±0.5
Qwen3-8B	Qwen3-1.7B	33.6±1.6	33.8±2.8	46.9±1.3	86.6±0.4	78.5±0.6	55.9±1.2
	Qwen3-8B	**35.0**±1.0	**37.1**±2.3	**48.0**±0.9	**87.0**±0.3	**78.9**±0.7	**57.2**±0.9
	Llama-3.1-8B-Instruct	33.9±1.1	34.3±2.4	47.4±0.6	86.5±0.6	78.4±0.4	56.1±0.8

Table 3: Solver performance (pass@1) comparison under different proposers over three seeds, best performance is in **blue**.

on the top 50% of its generated data ranked by each (Table 2). As shown in Table 1, learnability is numerically better at Qwen3-1.7B and Qwen3-4B (42.9 vs 42.6; 55.6 vs 55.1) and IFD at Qwen3-8B and Qwen3-14B (56.8 vs 56.4; 59.1 vs 58.6); none of these gaps is statistically distinguishable from zero at our sample size (Section 5.3 reports the interval estimates).

Proposer support. We study how proposer support affects the solver, holding the solver fixed while retraining it on data generated by a different proposer (Table 3). A weaker proposer (Qwen3-1.7B) lowers the Qwen3-8B solver from 57.2 to 55.9, while a stronger proposer (Qwen3-8B) raises the Qwen3-1.7B solver from 43.1 to 43.5; only the larger of these two swaps approaches significance (Section 5.3), so we read the pair as directionally suggestive rather than as establishing a consistent effect. We report two diversity metrics in Table 4: mean NN, the similarity of each task to its nearest neighbour, and distinct@0.8, the fraction of tasks that are not near-duplicates. By both metrics, the Qwen3-8B proposer generates more diverse tasks than the Qwen3-1.7B (mean NN 0.53 vs 0.63; distinct@0.8 98.5 vs 93.5), consistent with the direction of the solver swap above. The direction aligns with the proposer-support argument (Proposition 2): refinement only chooses within the proposer's output, so the proposer's properties, not the selection over it, bound the solver's ceiling. We caution against a causal reading of diversity specifically: with only three proposers, diversity is confounded with proposer capacity and generated-task correctness, which we do not isolate. The third proposer, Llama-3.1-8B-Instruct, makes the point directly — its task set is as diverse as Qwen3-8B's (mean NN 0.528 vs 0.530; distinct@0.8 98.1 vs 98.5) yet it yields a worse solver at both scales (42.4 vs 43.5 and 56.1 vs 57.2), so diversity alone does not predict the solver outcome in our data. Proposition 2 licenses only the structural claim that refinement cannot select outside the proposer's output; which proposer-side property sets the ceiling is left to future work with capability-matched proposers.

Beyond the Qwen3 models. We additionally repeat the comparison on DeepSeek-R1-Distill-Qwen-7B (DeepSeek-AI, 2025). Among the refinement operators, learnability is the strongest and is competitive with the human-curated data (average performance 47.0 vs 46.8). This model is also the one case in Table 1 where no training arm improves on the untrained base (47.1): the non-refinement arm in particular drops to 44.4 with a much larger seed spread (±5.1 versus ±0.1 for the base), so for

proposer	mean NN ↓	distinct@0.8 (%) ↑
Qwen3-1.7B	0.628±0.008	93.5±1.3
Qwen3-8B	0.530±0.010	98.5±0.1
Llama-3.1-8B-Instruct	0.528±0.076	98.1±2.5

Table 4: Proposer-generated task set diversity, including mean NN and distinct@0.8 over three seeds.

this model the loop is unstable rather than merely unhelpful, and we do not read its operator ordering as evidence that refinement helps. We also provide additional experiment results of other models in Appendix D.

5.3 Additional Discussion and Limitations

Statistical importance. Section 5.2 report only mean±std over three seeds, which cannot by itself separate a true null effect from insufficient power to detect a real one. Treating each arm's reported mean and std as summary statistics (n=3), a Welch two-sample t-test on the `avg` column gives $|t| \leq 1.84$ and $p \geq 0.15$ for every operator-vs-non-refinement comparison in Table 1 (ten comparisons: 2 operators $\times$ 5 models), with 95% confidence intervals on the mean difference that include zero in every case — half-widths run from about 1 point (Qwen3-14B) to about 12 points (DeepSeek-R1-Distill-Qwen-7B, whose non-refinement arm is itself highly unstable across seeds); the two proposer-support comparisons in Table 3 give $p = 0.47$ ($43.1 \rightarrow 43.5$) and $p = 0.21$ ($55.9 \rightarrow 57.2$). None of these differences is significant at $\alpha = 0.05$. This is *consistent with*, but does not *establish*, a true null effect: at $n = 3$ our tests have low power, so a moderate true effect on the order of the observed point differences could easily fail to reach significance, and the sign of the operator-vs-non-refinement gap in fact varies by model (Section 5.2) rather than looking uniformly null. We therefore read the result as *refinement is not distinguishable from non-refinement at this sample size*, not as *refinement has no effect*, and have revised Sections 1, 5.2 and 6 to keep that distinction explicit. A pre-registered equivalence test at larger n is the natural next step toward a positive equivalence claim.

Theory-practice gap. Each refinement operator shifts its own target metric without a significant effect on solver performance, which is what we would expect if refinement leaves the alignment A largely unchanged; the wide confidence intervals above mean this is also what we would expect from an underpowered comparison to a real but moderate effect, and our data cannot yet separate the two readings. Directly measuring $A(D')$ against the realized one-step change ΔM is itself challenging. Testing whether the per-task alignment $a_\tau = \langle \nabla M, \Delta_\tau \rangle$ is uncorrelated with learnability, as Remark 2 assumes, would both validate the assumption and guide the design of alignment-aware operators; we leave this to future work.

Experiment scope. The decoupled, single-round setting motivated in Section 2 is close to the least favorable case for observing a refinement benefit: one small, fixed-budget LoRA (rank 32) update gives little room for refinement's effect to compound the way it would across the multi-round curricula that motivate it in deployed systems, since a fixed proposer never adapts to the tasks the solver is failing on. Our null result is evidence about this narrow regime, not the closed-loop, multi-round systems it is more commonly used in; testing that directly is the natural next step. The proposer-support result separately assumes Lemma 1, which off-policy data or support-expanding decoding can violate. Moreover, it holds at the effective-support level (e.g., exact softmax support is the full task space), so it is a statement about what finite sampling can realize rather than a hard impossibility. We use LoRA and vary the proposer only by model size, so proposer diversity and capacity are confounded (Section 5.2) and broader, capability-matched settings remain to be tested.

Failure cases. As shown in Appendix D, not all models train stably in the proposer–solver loop: Llama-3.1-8B-Instruct degrades under every method with std up to ±36.6 points, an instability we cannot yet attribute to a single cause. One plausible reading is that LoRA's constrained update capacity, not the refinement operator, is the bottleneck for this model — if so, the flat, non-significant Qwen3 results in Section 5.2 could partly reflect an optimizer that is insensitive to task selection generally, rather than refinement specifically being ineffective, and the same confound could inflate or mask a true refinement effect wherever it is present. We flag this as our own inference from the reported variance rather than a claim the experiments were designed to test; distinguishing an

insensitive optimizer from a genuinely null refinement effect needs a direct comparison (e.g. full fine-tuning or higher-rank LoRA on the same models) that we leave to future work. Characterizing when data-free evolution trains stably, and extending it reliably across model families, is an important direction in its own right.

6 CONCLUSION

In this work, we investigate the refinement operator in data-free evolution through its effect on solver gradient and proposer support, and located two limits: its effect runs through the alignment between the solver gradient and downstream performance functional, and it can only select tasks the proposer emits. Empirically, no refinement operator showed a statistically distinguishable advantage over the unrefined task set, and the direction of the (non-significant) gap varied by model. Using a stronger, more diverse proposer pointed toward better solver performance in both comparisons we ran, though only the larger swap approached significance; we read this as suggestive that proposer-side diversity, not selection over a fixed proposer, is the more promising lever, and as a call for the larger, capability-matched comparisons needed to establish it. This lever is currently unexploited by design: the proposer objective rewards validity alone, so whatever diversity the generated task set exhibits is inherited from the base model's sampling distribution and the topic distribution $p_{\mathcal{Z}}$ rather than optimized for. Making diversity an explicit term in the proposer objective is therefore a concrete direction for future work.

AI USE STATEMENT

We used generative AI tools to improve the grammar, language, and readability of the manuscript. No generative AI tools were used for research methodology, experiments, theoretical claims, or analysis. All AI-assisted text was reviewed by the authors, who take full responsibility for the final content.

ETHICS STATEMENT

This work does not involve human subjects or sensitive data, and all experiments were conducted in accordance with the principles of responsible research.

REPRODUCIBILITY STATEMENT

All assumptions and complete proofs of the theoretical claims in Section 4 are given in Appendix A. The proposer–solver loop, the implementation of every refinement operator we compare, and the offline construction of each refined task set are described in Appendix B; the full set of training and evaluation hyperparameters is listed in Table 5, and the topic and prompt templates used by the proposer are given in Table 6. Per-model results, including the failure cases discussed in Section 6, are reported in Appendix D, and training curves in Appendix C. We will release the code and the generated task sets upon acceptance.

REFERENCES

Yoshua Bengio, Jérôme Louradour, Ronan Collobert, and Jason Weston. Curriculum learning. In *Proceedings of the 26th annual international conference on machine learning*, pp. 41–48, 2009.

Roger Creus Castanyer, Geoffrey Bradway, Lorenz Wolf, Maxwill Lin, Augustine N Mavor-Parker, and Matthew James Sargent. Populora: Co-evolving llm populations for reasoning self-play, 2026.

Justin Yang Chae, Md Tanvirul Alam, and Nidhi Rastogi. Towards understanding self-play for llm reasoning, 2025.

Lili Chen, Mihir Prabhudesai, Katerina Fragkiadaki, Hao Liu, and Deepak Pathak. Self-questioning language models, 2025a.

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, 2021. URL https://arxiv.org/abs/2107.03374.

Yixing Chen, Yiding Wang, Siqi Zhu, Haofei Yu, Tao Feng, Muhan Zhang, Mostofa Patwary, and Jiaxuan You. Multi-agent evolve: Llm self-improve through co-evolution, 2025b.

Codeforces. Codeforces problemset: Problem tags. https://codeforces.com/problemset, 2026. Algorithm tag taxonomy; accessed 2026-06-04.

DeepSeek-AI. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning, 2025. URL https://arxiv.org/abs/2501.12948.

Jacob Dineen, Aswin RRV, Zhikun Xu, and Ben Zhou. Vocabulary dropout for curriculum diversity in llm co-evolution, 2026.

Amirata Ghorbani and James Zou. Data shapley: Equitable valuation of data for machine learning. In *International conference on machine learning*, pp. 2242–2251. PMLR, 2019.

Daya Guo, Qihao Zhu, Dejian Yang, Zhenda Xie, Kai Dong, Wentao Zhang, Guanting Chen, Xiao Bi, Yifan Wu, YK Li, et al. Deepseek-coder: when the large language model meets programming–the rise of code intelligence. *arXiv preprint arXiv:2401.14196*, 2024.

Daniel Han, Michael Han, and Unsloth team. Unsloth, 2023. URL https://github.com/unslothai/unsloth.

Edward J Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, Weizhu Chen, et al. Lora: Low-rank adaptation of large language models. volume 1, pp. 3, 2022.

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, 2025.

Naman Jain, Alex Gu, Wen-Ding Li, Fanjia Yan, Tianjun Zhang, Sida Wang, Armando Solar-Lezama, Koushik Sen, Ion Stoica, et al. Livecodebench: Holistic and contamination free evaluation of large language models for code. In *International Conference on Learning Representations*, volume 2025, pp. 58791–58831, 2025.

Swadesh Jana, Cansu Sancaktar, Tomáš Daniš, Georg Martius, Antonio Orvieto, and Pavel Kolev. Gasp: Guided asymmetric self-play for coding llms, 2026.

Pang Wei Koh and Percy Liang. Understanding black-box predictions via influence functions. In *International conference on machine learning*, pp. 1885–1894. PMLR, 2017.

Jakub Grudzien Kuba, Mengting Gu, Qi Ma, Yuandong Tian, Vijai Mohan, and Jason Chen. Language self-play for data-free training, 2025.

M Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. volume 23, 2010.

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, 2024.

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, 2026a.

Ming Li, Yong Zhang, Zhitao Li, Jiuhai Chen, Lichang Chen, Ning Cheng, Jianzong Wang, Tianyi Zhou, and Jing Xiao. From quantity to quality: Boosting llm performance with self-guided data selection for instruction tuning. pp. 7602–7635, 2024.

Rongao Li, Jie Fu, Bo-Wen Zhang, Tao Huang, Zhihong Sun, Chen Lyu, Guang Liu, Zhi Jin, and Ge Li. Taco: Topics in algorithmic code generation dataset. *arXiv preprint arXiv:2312.14852*, 2023.

Shangzhe Li, Xuchao Zhang, Chetan Bansal, and Weitong Zhang. Your self-play algorithm is secretly an adversarial imitator: Understanding llm self-play through the lens of imitation learning, 2026b.

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. volume 36, pp. 21558–21572, 2023.

Wei Liu, Siya Qi, Yali Du, and Yulan He. Self-play only evolves when self-synthetic pipeline ensures learnable information gain, 2026.

Zichen Liu, Changyu Chen, Wenjun Li, Penghui Qi, Tianyu Pang, Chao Du, Wee Sun Lee, and Min Lin. Understanding r1-zero-like training: A critical perspective. 2025.

Vaibhav Mishra. Preventing curriculum collapse in self-evolving reasoning systems, 2026.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. volume 35, pp. 27730–27744, 2022.

Yulin Peng, Xinxin Zhu, Chenxing Wei, Nianbo Zeng, Leilei Wang, Ying Tiffany He, and F Richard Yu. Sage: Multi-agent self-evolution for llm reasoning, 2026.

Prime Intellect. DeepCoder gold-standard solutions. https://huggingface.co/datasets/PrimeIntellect/deepcoder-gold-standard-solutions, 2025.

Ozan Sener and Silvio Savarese. Active learning for convolutional neural networks: A core-set approach. In *International Conference on Learning Representations*, 2018.

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, 2024.

Richard S Sutton, David McAllester, Satinder Singh, and Yishay Mansour. Policy gradient methods for reinforcement learning with function approximation. volume 12, 1999.

Swabha Swayamdipta, Roy Schwartz, Nicholas Lourie, Yizhong Wang, Hannaneh Hajishirzi, Noah A Smith, and Yejin Choi. Dataset cartography: Mapping and diagnosing datasets with training dynamics. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pp. 9275–9293, 2020.

Gemma Team. Gemma 3 technical report, 2025. URL https://arxiv.org/abs/2503.19786.

Llama3 Team. The llama 3 herd of models, 2024. URL https://arxiv.org/abs/2407.21783.

Mariya Toneva, Alessandro Sordoni, Remi Tachet des Combes, Adam Trischler, Yoshua Bengio, and Geoffrey J Gordon. An empirical study of example forgetting during deep neural network learning. In *International Conference on Learning Representations*, 2019.

Leandro von Werra, Younes Belkada, Lewis Tunstall, Edward Beeching, Tristan Thrush, Nathan Lambert, Shengyi Huang, Kashif Rasul, and Quentin Gallouédec. TRL: Transformers Reinforcement Learning, 2020. URL https://github.com/huggingface/trl.

Ronald J Williams. Simple statistical gradient-following algorithms for connectionist reinforcement learning. *Machine learning*, 8(3):229–256, 1992.

Fang Wu, Weihao Xuan, Ximing Lu, Mingjie Liu, Yi Dong, Zaid Harchaoui, and Yejin Choi. The invisible leash: Why rlvr may or may not escape its origin, 2025.

Anyi Xu, Bangcai Lin, Bing Xue, Bingxuan Wang, Bingzheng Xu, Bochao Wu, Bowei Zhang, Chaofan Lin, Chen Dong, Chenchen Ling, et al. Deepseek-v4: Towards highly efficient million-token context intelligence. *arXiv preprint arXiv:2606.19348*, 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, 2025.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model? volume 38, pp. 57654–57689, 2025.

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, 2025.

Qihao Zhu, Daya Guo, Zhihong Shao, Dejian Yang, Peiyi Wang, Runxin Xu, Y Wu, Yukun Li, Huazuo Gao, Shirong Ma, et al. Deepseek-coder-v2: Breaking the barrier of closed-source models in code intelligence. *arXiv preprint arXiv:2406.11931*, 2024.

Terry Yue Zhuo, Minh Chien Vu, Jenny Chim, Han Hu, Wenhao Yu, Ratnadira Widyasari, Imam Nur Bani Yusuf, Haolan Zhan, Junda He, Indraneil Paul, et al. Bigcodebench: Benchmarking code generation with diverse function calls and complex instructions. In *International Conference on Learning Representations*, volume 2025, pp. 66602–66656, 2025.

A PROOFS

All quantities are defined as in Sections 3 and 4. We write ∇M for $\nabla_{\theta_S} M(\theta_S)$ and abbreviate

$$G(D') = \frac{1}{|D'|} \sum_{\tau \in D'} g(\tau; \theta_S),$$
$$\alpha(\tau) = \alpha(\tau; \theta_S, M) = \langle \nabla M, g(\tau; \theta_S) \rangle,$$
$$A(D') = \frac{1}{|D'|} \sum_{\tau \in D'} \alpha(\tau) = \langle \nabla M, G(D') \rangle.$$

The $O(\eta^2)$ notation is understood under the local Taylor remainder in Remark 1: for the refined task sets considered below, there exists $C < \infty$ such that

$$|M(\theta_S + \eta G(D')) - M(\theta_S) - \eta \langle \nabla_{\theta_S} M(\theta_S), G(D') \rangle| \leq C\eta^2.$$

This is the only regularity condition used in the first-order comparisons.

A.1 PROOF OF THE PERFORMANCE DIFFERENCE ALIGNMENT PROPOSITION

Proposition 3 (Alignment determines first-order performance, restated)**.** *The first-order performance difference is*

$$\Delta M(D') = \eta \, A(D'; \theta_S, M) + O(\eta^2),$$

so it depends on the refined task set D' only through its performance alignment $A(D'; \theta_S, M)$.

Proof. By Remark 1 and Definition 2,

$$\begin{aligned}
\Delta M(D') &= \eta \langle \nabla M, G(D') \rangle + O(\eta^2) \\
&= \eta \left\langle \nabla M, \frac{1}{|D'|} \sum_{\tau \in D'} g(\tau; \theta_S) \right\rangle + O(\eta^2) \\
&= \eta \frac{1}{|D'|} \sum_{\tau \in D'} \langle \nabla M, g(\tau; \theta_S) \rangle + O(\eta^2) \\
&= \eta A(D') + O(\eta^2).
\end{aligned}$$

□

A.2 PROOF OF THE REFINEMENT COMPARISON COROLLARY

Corollary 2 (Refinement Comparison, restated)**.** *The performance difference between the refined task set $D' = F(D)$ and the unrefined task set D satisfies*

$$\Delta M(D') - \Delta M(D) = \eta \big(A(D'; \theta_S, M) - A(D; \theta_S, M) \big) + O(\eta^2),$$

so to first order, refinement improves performance over the unrefined set if and only if $A(D'; \theta_S, M) > A(D; \theta_S, M)$.

Proof. By Proposition 3, $\Delta M(D) = \eta A(D) + O(\eta^2)$ and $\Delta M(D') = \eta A(D') + O(\eta^2)$, so

$$\Delta M(D') - \Delta M(D) = \eta \big(A(D') - A(D) \big) + O(\eta^2),$$

whose first-order term is positive if and only if $A(D') > A(D)$. □

A.3 EXPECTATION-OPTIMALITY OF LEARNABILITY

Remark 3 (Expectation-optimality of learnability, restated). Assume the alignment between a task gradient's direction and the performance gradient is independent of the learnability and positive in expectation. Then learnability is expectation-optimal to maximize the expected performance alignment.

Proof. Due to $g(\tau;\theta_S)$ is invariant to a constant baseline, since the score function has zero mean by the log-derivative identity, $\mathbb{E}_{y\sim S_{\theta_S}(\cdot|x)}[\nabla_{\theta_S}\log S_{\theta_S}(y\mid x)]=0$.

We have

$$p_\tau = \mathbb{E}_{y\sim S_{\theta_S}(\cdot|x)}[R_S(x,y,U)].$$

Then, we have

$$\begin{aligned}
g(\tau;\theta_S) &= \mathbb{E}_{y\sim S_{\theta_S}(\cdot|x)}\big[R_S(x,y,U)\,\nabla_{\theta_S}\log S_{\theta_S}(y\mid x)\big]\\
&= \mathbb{E}_{y\sim S_{\theta_S}(\cdot|x)}\big[\big(R_S(x,y,U)-p_\tau\big)\,\nabla_{\theta_S}\log S_{\theta_S}(y\mid x)\big]\\
&= p_\tau\,(1-p_\tau)\,\bar{u}_+ + (1-p_\tau)\,(0-p_\tau)\,\bar{u}_-\\
&= p_\tau(1-p_\tau)\,\bar{u}_+ - p_\tau(1-p_\tau)\,\bar{u}_-\\
&= p_\tau(1-p_\tau)\,(\bar{u}_+ - \bar{u}_-)\\
&= \mathrm{learn}(\tau)\,\Delta_\tau,
\end{aligned}$$

where $\bar{u}_+ = \mathbb{E}_{y\sim S_{\theta_S}(\cdot|x)}[\nabla_{\theta_S}\log S_{\theta_S}(y\mid x)\mid R_S=1]$ and $\bar{u}_- = \mathbb{E}_{y\sim S_{\theta_S}(\cdot|x)}[\nabla_{\theta_S}\log S_{\theta_S}(y\mid x)\mid R_S=0]$ are the solver's mean updates on its passing and failing rollouts, $\mathrm{learn}(\tau)=p_\tau(1-p_\tau)$ is the learnability, and $\Delta_\tau=\bar{u}_+-\bar{u}_-$ is the gradient direction.

Let $a_\tau=\langle\nabla M,\Delta_\tau\rangle$ be the alignment between this direction and the performance gradient; by hypothesis it is independent of $\mathrm{learn}(\tau)$ with $\mathbb{E}[a_\tau]=\bar{a}>0$. By Proposition 1, each task contributes $\alpha(\tau)=\mathrm{learn}(\tau)\,a_\tau$, so

$$\begin{aligned}
\mathbb{E}\big[A(D')\big] &= \frac{1}{|D'|}\sum_{\tau\in D'}\mathbb{E}[\alpha(\tau)]\\
&= \frac{1}{|D'|}\sum_{\tau\in D'}\mathrm{learn}(\tau)\,\mathbb{E}[a_\tau]\\
&= \bar{a}\,\frac{1}{|D'|}\sum_{\tau\in D'}\mathrm{learn}(\tau),
\end{aligned}$$

which, as $\bar{a}>0$, is maximized over size-N subsets by the N tasks of largest $\mathrm{learn}(\tau)$ using learnability. □

A.4 Proof of the Support Limitation Proposition

Proposition 4 (Support limitation of refinement, restated)**.** *Under Lemma 1, for any $z\in\mathcal{Z}$, we have*

$$D'_z \subseteq D_z \subseteq \mathrm{supp}_{\theta_P}(z) \subseteq \mathrm{supp}_{\theta_P^0}(z).$$

Proof. We have

$$D'_z = F(D_z)\subseteq D_z.$$

Since D_z is a finite sample from $P_{\theta_P}(\cdot\mid z)$ and the effective-support threshold ϵ is set by the sampling budget, a task with $P_{\theta_P}(\tau\mid z)\le\epsilon$ is not realized when collecting D_z, so

$$\tau\in D_z \quad\Longrightarrow\quad \tau\in\mathrm{supp}_{\theta_P}(z)$$

Therefore,

$$D'_z\subseteq D_z\subseteq\mathrm{supp}_{\theta_P}(z)$$

Lemma 1 gives

$$\mathrm{supp}_{\theta_P}(z)\subseteq\mathrm{supp}_{\theta_P^0}(z).$$

Therefore, we have

$$D'_z\subseteq D_z\subseteq\mathrm{supp}_{\theta_P}(z)\subseteq\mathrm{supp}_{\theta_P^0}(z).$$

□

The proposer-support analysis is conditional on Lemma 1, while the solver-gradient analysis uses only the local expansion in Remark 1.

B Implementation and Selection Operators

Hyperparameters. The training and evaluation settings are listed in Table 5.

hyperparameter	value
LoRA rank	32
RLVR objective	Dr. GRPO (Liu et al., 2025)
proposer–solver rounds	1
proposer / solver update steps	500 / 500
refinement fraction	top 50%
learning rate	$1e{-}5$
batch size	8
group size	8
max sequence length	8192
proposer topics	40 Codeforces algorithm tags (Codeforces, 2026)
eval samples / temperature / top-p	8 / 0.3 / 0.95

Table 5: Training and evaluation hyperparameters.

Reproducibility. We will release the code upon acceptance. The topics and prompts example is provided in Table 6.

Component	**Content**
Topics	array manipulation; string processing; basic math operations; list sorting and filtering; counting and frequency analysis; simple simulation; bit manipulation basics; interval and date arithmetic; matrix indexing; character frequency and anagrams; two pointers technique; binary search; hash map usage; stack-based parsing; greedy algorithms; prefix sums; sliding window; recursion and backtracking; matrix operations; interval scheduling; heap / priority queue; deque techniques; regex and pattern matching; state machine simulation; game / move simulation; geometry primitives; combinatorial enumeration; string pattern search; graph traversal (BFS/DFS); dynamic programming; shortest path algorithms; tree algorithms; number theory and modular arithmetic; segment trees or binary indexed trees; combinatorics; union-find / disjoint sets; topological sorting; minimum spanning tree; monotonic stack / queue; digit DP / bitmask DP.
Proposer system prompt	”You are an expert competitive programming problem setter. Your task is to create a novel Python programming problem, along with unit tests and a reference solution. The problem should test algorithmic thinking, resemble contest-style programming tasks, and be solvable using standard input/output.”
Proposer user template	”Create a Python programming problem about {topic}. Avoid toy scripting tasks, classroom phrasing, and generic templates like ’Write a Python program’. Prefer contest-style statements with a concrete scenario, constraints, edge cases, and a nontrivial algorithmic choice. Response must contain exactly three sections: `## Problem Description` (statement with stdin/stdout format), `## Unit Tests` (exactly 4 test cases as a JSON list of {input, output} objects), and `## Reference Solution` (complete Python code block).”
Solver system prompt	”You are an expert Python programmer. You will be given a question (problem specification) and will generate a correct Python program that matches the specification and passes all tests.”
Solver user template	”Question: {problem}\n Please reason step by step, and write your code in markdown format, e.g., ```python\n# YOUR CODE HERE\n```.”
Topic validation prompt	”You are checking whether a generated competitive-programming task genuinely belongs to its ASSIGNED TOPIC. ASSIGNED TOPIC: {topic}. PROBLEM STATEMENT: {problem}. REFERENCE SOLUTION: ```python\n{solution}\n```. Judge from the reference solution’s CORE ALGORITHM, not from the wording of the problem. Answer NO if the topic is only named/implied in the prose but the solution’s algorithm does not actually use it, or if the task is generic and unrelated to the topic. Answer YES only if genuinely solving this task exercises the topic ”{topic}”. Reply with exactly one word: YES or NO.”

Table 6: Topics and prompt templates.

C TRAINING CURVES

We provide additional learning curves of the proposer (Figure 1) and the solver (Figure 2).

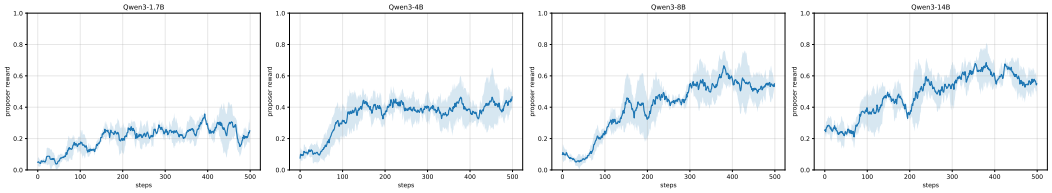


Figure 1: Proposer training reward.

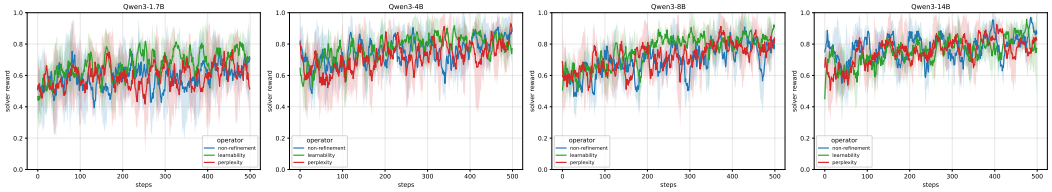


Figure 2: Solver training reward.

D ADDITIONAL RESULTS

We provide additional models in the following table. Gemma-3-1B-Instruct stays near its base performance across all methods, likely because its limited capacity under LoRA leaves little room for additional training gains. Llama-3.1-8B-Instruct degrades under every method with high variance, which we read as evidence that LoRA's constrained updates struggle to stabilize training for this model (see the caveat in Section 5.3: we cannot separate this from an optimizer that is insensitive to task selection generally). We also ran DeepSeek-R1-Distill-Qwen-1.5B, but report it here only as an incomplete run rather than in the table: of its five arms, only the untrained base completed all three seeds, while the non-refinement and learnability arms completed two seeds each and the IFD arm only one, so per-arm variance is not estimable on a common footing and any operator ordering read from it would be an artifact of which seeds happened to finish.

models (proposer/solver)	LiveCodeBench-v6	LiveCodeBench-v5	BigCodeBench	HumanEval+	MBPP+	Avg
Gemma-3-1B-Instruct	3.9±0.3	3.0±0.1	10.0±0.3	**36.5**±0.3	**54.4**±0.2	**21.6**±0.1
human-curated data	2.1±1.5	3.6±1.2	10.1±0.8	34.3±0.6	52.5±2.3	20.5±0.8
non-refinement	4.5±0.4	3.0±0.4	**10.7**±0.6	35.9±1.2	52.5±2.2	21.3±0.2
learnability	**4.8**±0.7	**3.8**±0.5	10.3±0.6	33.5±1.0	51.2±2.1	20.7±0.6
IFD	3.9±0.6	2.3±0.5	9.8±0.7	35.4±1.2	53.9±0.6	21.0±0.7
Llama-3.1-8B-Instruct	15.8±0.2	10.4±0.8	40.7±0.5	**64.2**±0.8	**53.0**±0.5	**36.8**±0.1
human-curated data	**20.1**±1.2	**14.2**±1.4	**41.7**±0.2	54.3±15.8	37.8±33.3	33.6±10.0
non-refinement	17.1±1.4	11.4±0.9	41.1±0.7	57.7±4.7	26.7±23.1	30.8±5.5
learnability	16.9±1.8	10.9±1.3	33.8±11.8	42.2±36.6	40.8±35.4	28.9±17.4
IFD	17.1±0.6	11.0±2.0	33.4±12.6	34.3±23.0	5.8±9.5	20.3±7.9

Table 7: Solver performance (pass@1) of additional models on the five code benchmarks; mean±std over three seeds, best performance in **blue**.