



SynCo: Data Synthesis Co-Training for Self-Evolving LLMs via Multi-Agent Reinforcement Learning

Wei Yang¹, Shawn Li¹, Yuehan Qin¹, Yawei Wang², Mingxi Wang¹, Shixuan Li¹
 Tiankai Yang¹, Jiate Li¹, Jesse Thomason³, Xuezhe Ma¹, Yue Zhao¹

¹University of Southern California

²The George Washington University

³Georgia Institute of Technology

Abstract

Self-evolving LLM agents promise to improve autonomously through continual interaction and learning, reducing their dependence on manually curated supervision. Realizing this promise requires not only updating the agent, but also evolving its training experience as its capabilities change. However, most existing pipelines rely on static datasets or separately updated synthesis models, causing previously useful tasks to become trivial while overly difficult tasks remain uninformative. This growing mismatch between agent capability and training experience limits sustained self-improvement. To address this problem, we propose SYNCO, an agentic data synthesis co-training framework for self-evolving LLMs based on multi-agent reinforcement learning. SYNCO jointly optimizes two independently parameterized agents: a Synthesizer that constructs training tasks from the Reasoner’s evolving capability state, and a Reasoner that learns from the resulting experience. Each synthesized task induces multiple Reasoner rollouts whose outcomes provide complementary rewards to both agents. Correctness feedback improves the Reasoner, while task quality, answer reliability, and outcome-grounded teachability guide the Synthesizer. Their updates are fed back into subsequent synthesis rounds, allowing the task-solving policy and its training distribution to evolve together. Extensive experiments across eight mathematical reasoning benchmarks demonstrate that SYNCO substantially outperforms a broad range of existing synthetic-data methods and controlled baselines, achieving the strongest overall performance while deriving most of its gains from previously unsolved problems.

GitHub: <https://github.com/weiyang930/SynCo.git>

Hugging Face: <https://huggingface.co/datasets/weiyang930/SynCo>

1 Introduction

Large language model agents are increasingly expected to improve through interaction and accumulated experience, rather than remain bounded by a fixed collection of human-curated demonstrations [Tang et al., 2025, Gao et al., 2026]. Yet most agent training pipelines still rely on data that is static, externally constructed, or only periodically refreshed. This creates a fundamental mismatch: agent capabilities change throughout training, while the distribution of learning experience does not [Pahuja et al., 2025, Liang et al., 2026]. Tasks that were once informative may become trivial, whereas tasks far beyond the agent’s competence continue to provide little useful learning signal. The resulting misalignment between the learner and its data limits both training efficiency and sustained improvement. A genuinely self-evolving agent therefore requires not merely more synthetic data, but a source of training experience that continuously adapts to its evolving capabilities [Cheng et al., 2025, Zhang et al., 2026a, Yue et al., 2026].

Recent work has begun to address this limitation through adaptive data synthesis, weakness-driven task generation, and teacher–solver or challenger–solver co-evolution [Zhai et al., 2025, Yang et al., 2026a, Yue et al., 2026]. These approaches adapt the training data to the current learner by regenerating, selecting, or reweighting tasks according to model failures, uncertainty, or estimated difficulty. However, an adaptive data distribution does not necessarily imply joint optimization of data synthesis and task learning. Many existing frameworks still alternate between generating tasks from a previous learner checkpoint and subsequently training the learner on the resulting data [Huang et al., 2025, Wang et al., 2025a]. Although this enables iterative adaptation, the synthesis policy and the task policy are not optimized together from the same online interaction. Data generation therefore remains an auxiliary process around agent training rather than a learnable component of the training system itself. True mutual evolution requires jointly optimizing the experience synthesizer and the learner.

Such joint training creates a coupled and non-stationary learning problem. The Synthesizer shapes the Reasoner’s optimization landscape through the tasks it generates, while each Reasoner update redefines which tasks remain useful. A task that is informative at one stage may later become trivial, whereas a nominally difficult task may still provide little supervision if the Reasoner consistently succeeds or fails. The value of synthetic data therefore cannot be judged solely by static properties such as validity, novelty, or nominal difficulty, but must be grounded in the current Reasoner’s actual interaction with the data [Yang et al., 2026a, Kwan et al., 2026]. This leads to the central question of our work: *Can agentic data synthesis and task learning be formulated as a unified multi-agent reinforcement learning process in which the data-synthesis policy and the task-solving policy are jointly optimized and continuously evolve together?* Answering this question requires more than connecting a generator to a learner. It requires a closed loop that integrates learner-state observation, adaptive task generation, shared interaction feedback, and online updates of both policies.

To address this challenge, we propose SYNCO, an agentic data synthesis co-training framework for self-evolving LLMs based on multi-agent reinforcement learning. SYNCO jointly optimizes two independently parameterized agents: a *Synthesizer* that constructs training tasks and a *Reasoner* that learns from them. At each iteration, the Synthesizer observes an evolving context built from the recent synthetic data pool, the Reasoner’s empirical capability profile, and feedback from rejected tasks. It then generates a structured and verifiable task tailored to the Reasoner’s current learning needs. The Reasoner performs multiple solution rollouts, whose outcomes provide complementary learning signals to both agents. Correctness rewards improve the Reasoner, while task quality, answer reliability, and outcome-grounded teachability guide the Synthesizer. Both policies are updated online through group-relative policy optimization within the same multi-agent training process. Their updates define the next synthesis round: the improved Reasoner induces a new capability state, and the improved Synthesizer converts that state into a new training distribution. Through this closed loop, SYNCO enables the task-solving policy and its training data to evolve together.

Our main contributions are summarized as follows:

- We formulate agentic data synthesis and task learning as a coupled multi-agent reinforcement learning problem, and propose SYNCO to jointly optimize a Synthesizer and a Reasoner within a unified online training process.
- We develop a closed-loop self-evolution framework in which the Synthesizer continually adapts the training distribution to the evolving Reasoner, while outcome-grounded interaction feedback from the Reasoner directly supervises the synthesis policy.
- Extensive experiments across diverse mathematical reasoning benchmarks show that SYNCO consistently outperforms a frozen-Synthesizer baseline, with most gains arising from previously unsolved problems.

2 Related Work

Synthetic data has become central to LLM post-training, ranging from instruction bootstrapping methods such as Self-Instruct [Wang et al., 2023] and WizardLM [Xu et al., 2024] to reasoning-oriented pipelines such as STaR [Zelikman et al., 2022], MetaMath [Yu et al., 2023], and DeepSeekMath [Shao et al., 2024]. Recent work further improves realism, diversity, verifiability, and learner relevance: MATRIX [Tang et al., 2025] generates scenarios through multi-agent simulation, FAC Synthesis [Li et al., 2026a] targets missing representation-space features, Explorer [Pahuja et al., 2025] synthesizes

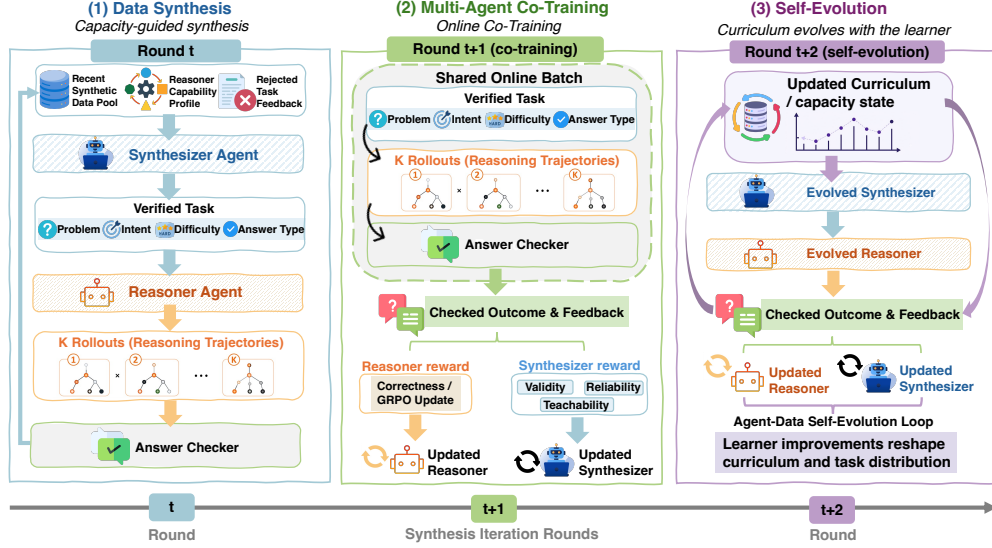


Figure 1: Overview of SYNCO, a closed-loop framework for self-evolving LLMs. (1) A Synthesizer first generates capability-guided and verifiable tasks from the current learning context. (2) The resulting interaction batch is then used to jointly update the Synthesizer and the Reasoner through online multi-agent co-training. (3) The updated agents in turn reshape the future curriculum and task distribution, forming an agent–data self-evolution loop.

trajectories through environment exploration, and SwS [Liang et al., 2026] constructs problems around persistent learner weaknesses. Related pipelines extend synthetic supervision to mathematical reasoning, tool use, web interaction, and executable environments [Gou et al., 2024, Ramrakhya et al., 2025, Wang et al., 2026a, Chen et al., 2026a]. Most of these approaches nevertheless construct an offline corpus or adapt generation separately from downstream policy optimization.

Self-evolving frameworks more directly couple experience generation with learner improvement. R-Zero [Huang et al., 2025], Socratic-Zero [Wang et al., 2025a], and Agent0 [Xia et al., 2025] establish co-evolving proposer, teacher, or curriculum roles for autonomous reasoning improvement. Dr. Zero [Yue et al., 2026] and SCOPE [Kwan et al., 2026] extend this paradigm to multi-turn search and open-ended document-grounded tasks, while CoEvolve [Yang et al., 2026a] adapts tool-use task distributions using rollout-derived forgetting and uncertainty signals. Agent-World [Dong et al., 2026], SEAD [Dai et al., 2026], AgentEvolver [Zhai et al., 2025], EvolveR [Wu et al., 2025], and EvoFSM [Zhang et al., 2026a] further explore the evolution of environments, curricula, memories, experiences, and agent workflows. A comprehensive discussion of synthetic-data generation and self-evolving agents is provided in Appendix A.

3 Method

In this section, we present SYNCO, a multi-agent reinforcement learning framework that jointly optimizes agentic data synthesis and task reasoning for continual self-evolution. The overall framework is illustrated in Figure 1.

3.1 Problem Formulation: Agentic Data Synthesis Co-Training

We consider two independently parameterized agents: a *Synthesizer* $\pi_S(\cdot; \theta_S)$ that generates training tasks and a *Reasoner* $\pi_R(\cdot; \theta_R)$ that learns to solve them. The Synthesizer determines the Reasoner’s training experience, while the Reasoner’s realized behavior provides feedback for improving synthesis.

At iteration t , a dynamic context c_t summarizes the Reasoner’s recent capabilities, the coverage of previously generated tasks, and feedback from rejected samples. Conditioned on c_t , the Synthesizer

generates a structured and verifiable task:

$$x_t \sim \pi_S(\cdot \mid c_t; \theta_S^t), \quad \mathcal{D}_t = \pi_S(\cdot \mid c_t; \theta_S^t), \quad (1)$$

where $\mathcal{D}_t$ is the online training distribution induced by the current learner state and synthesis policy.

For each admitted task x_t with verified answer a_t^* , the Reasoner samples K independent rollouts $y_t^{(k)} \sim \pi_R(\cdot \mid x_t; \theta_R^t)$, whose outcomes are determined by $z_t^{(k)} = \text{Check}(y_t^{(k)}, a_t^*)$. The same interaction yields rollout-level correctness rewards $r_{R,t}^{(k)}$ for the Reasoner and a task-level reward $r_{S,t}$ that evaluates both synthesis quality and realized utility for the current Reasoner.

The agents optimize distinct but coupled objectives:

$$J_R(\theta_R) = \mathbb{E} \left[\frac{1}{K} \sum_{k=1}^K r_{R,t}^{(k)} \right], \quad J_S(\theta_S) = \mathbb{E}[r_{S,t}]. \quad (2)$$

The objectives are coupled because π_S determines the distribution underlying J_R , whereas π_R produces the interaction feedback underlying J_S . Both policies are updated from the same online batch using agent-specific group-relative policy optimization, forming a unified co-training process rather than sequential synthesis and learning.

3.2 Capability-Conditioned Agentic Data Synthesis

A self-evolving Reasoner requires training experience that remains aligned with its changing capabilities. Rather than sampling from a fixed generator, the Synthesizer observes recent agent–data interactions, identifies which capabilities should be exercised next, and realizes this decision as a structured and verifiable task. Data generation therefore becomes an adaptive policy conditioned on the evolving learner.

Capability-conditioned synthesis context. Let $\mathcal{M}_t$ denote the interaction memory before iteration t , containing recent tasks, validation outcomes, Reasoner rollouts and success rates, synthesis rewards, and rejection records. We summarize it as

$$c_t = G(\mathcal{M}_t) = (p_t, d_t, e_t), \quad (3)$$

where p_t characterizes the current synthetic pool and its coverage, d_t summarizes the Reasoner’s empirical performance and remaining weaknesses, and e_t records synthesis failures, redundancy, and under-explored task regions. The context is reconstructed throughout training, allowing synthesis to adapt as both the Reasoner and the accumulated task distribution evolve.

Structured pedagogical task generation. Conditioned on c_t , the Synthesizer generates a structured task card

$$x_t = (q_t, \mathbf{m}_t, s_t^*, a_t^*) \sim \pi_S(\cdot \mid c_t; \theta_S^t), \quad (4)$$

where q_t is the problem presented to the Reasoner, $\mathbf{m}_t$ specifies its pedagogical intent and estimated difficulty, and (s_t^*, a_t^*) provides a reference solution and final answer for verification. The metadata encourages an explicit synthesis decision by specifying the targeted reasoning behavior, anticipated weakness, and novelty or verification requirements. Accordingly, the Synthesizer is instructed to produce tasks that are original, self-contained, non-trivial, and aligned with weaknesses or under-explored capabilities identified in c_t .

The reference solution, answer, and metadata are used only for validation and reward computation. The Reasoner receives q_t alone, preventing answer leakage and ensuring that its rollout outcomes reflect its own policy.

Verified admission. Because open-ended generation may produce malformed, ambiguous, inconsistent, or repetitive tasks, every task is evaluated by a verifier $V(x_t) = (v_t, \rho_t)$. Here, ρ_t records schema validity, problem completeness, answer consistency, output format, and duplication, while v_t determines whether the task enters Reasoner training:

$$v_t = \mathbb{I}[x_t \in \mathcal{X}_{\text{valid}}], \quad \tilde{\mathcal{D}}_t = \{x_t \sim \pi_S(\cdot \mid c_t; \theta_S^t) \mid v_t = 1\}. \quad (5)$$

When enabled, an additional consensus check independently solves the generated problem and rejects tasks whose inferred answers do not support the Synthesizer-provided reference. Invalid tasks are excluded from Reasoner training and return negative feedback to the Synthesizer, whereas admitted tasks form the adaptive distribution used for subsequent multi-agent optimization.

3.3 Interaction-Grounded Multi-Agent Co-Training

The verified tasks provide a shared interaction space for both agents. For each admitted task, the Reasoner’s rollouts simultaneously provide direct supervision for task solving and outcome-grounded feedback for data synthesis. This shared interaction couples the two policies within the same online training process.

Reasoner interaction and outcome verification. Given a valid task x_t , the Reasoner receives only its problem statement q_t and samples K independent trajectories:

$$\begin{aligned} y_t^{(k)} &\sim \pi_R(\cdot \mid q_t; \theta_R^t), \\ z_t^{(k)} &= \text{Check}(\hat{a}_t^{(k)}, a_t^*) \in \{0, 1\}, \\ \mu_t &= \frac{1}{K} \sum_{k=1}^K z_t^{(k)}, \quad k = 1, \dots, K, \end{aligned} \tag{6}$$

where $\hat{a}_t^{(k)}$ is the extracted answer and μ_t is the empirical success rate of the current Reasoner on task x_t . The checker supports normalized exact, numerical, symbolic, and set- or tuple-level equivalence. The resulting rollout group distinguishes uniformly solved or failed tasks from those producing mixed outcomes, making μ_t an outcome-grounded measure of task utility for the current Reasoner.

Agent-specific interaction rewards. The Reasoner is optimized with rollout-level correctness rewards, encouraging trajectories that produce verified answers. The Synthesizer instead receives a task-level gated reward:

$$r_{S,t} = G_q(r_t^{\text{quality}}) G_r(r_t^{\text{reliability}}) T(\mu_t), \tag{7}$$

where G_q and G_r softly gate task quality and answer reliability, while $T(\mu_t)$ measures realized teachability from the Reasoner’s rollout outcomes. The teachability signal is maximized for tasks producing mixed success and decreases toward uniformly solved or failed tasks, which provide little group-relative learning signal. Thus, quality and reliability constrain the feasible synthesis space, while the current Reasoner’s behavior determines which valid tasks are most useful for learning. Exact reward definitions and implementation constants are provided in Appendix B.2.

Joint multi-agent update. At each iteration, the system collects a shared online interaction batch

$$\mathcal{B}_t = \left\{ \left(c_i, x_i, \left\{ y_i^{(k)}, z_i^{(k)} \right\}_{k=1}^K \right) \right\}_{i=1}^B. \tag{8}$$

The same interaction batch induces separate group-relative objectives for the two agents: rollout-level correctness optimizes the Reasoner, while gated task utility optimizes the Synthesizer. Although their parameters and rewards remain separate, both policies are updated within the same training iteration:

$$\theta_R^{t+1} = \theta_R^t - \eta_R \nabla_{\theta_R} \mathcal{L}_R^t, \quad \theta_S^{t+1} = \theta_S^t - \eta_S \nabla_{\theta_S} \mathcal{L}_S^t. \tag{9}$$

The Synthesizer determines the task distribution from which the Reasoner learns, while the Reasoner’s realized outcomes determine the learning signal for the Synthesizer. Updating either policy therefore changes the other’s subsequent learning problem, forming a coupled online co-training process.

3.4 Closed-Loop Agent–Data Self-Evolution

The co-training procedure in Section 3.3 updates both the Reasoner and the policy that generates its training experience. To sustain self-evolution, each interaction batch is fed back into a persistent memory that reconstructs the learner state and conditions the next synthesis round.

Feedback-driven context renewal. After processing $\mathcal{B}_t$, the system records the generated tasks, validation results, verified rollouts, empirical success rates, and agent-specific rewards. Rejected tasks and their failure reasons are also retained to discourage repeated malformed, unreliable, or redundant generations. The memory and synthesis context are updated as

$$\begin{aligned} \mathcal{M}_{t+1} &= \text{Update}(\mathcal{M}_t, \mathcal{B}_t), \\ c_{t+1} &= G(\text{Recent}(\mathcal{M}_{t+1})). \end{aligned} \tag{10}$$

Table 1: Reasoning accuracy (%) across eight mathematical reasoning benchmarks. Vanilla denotes the initial Qwen3-8B model. Frozen Synthesizer retains online task generation but disables synthesis-policy updates. Avg. is the unweighted average across all benchmarks. Bold and underlined values indicate the best and second-best results, respectively.

Method	Primary Benchmarks					Advanced Benchmarks			Avg.
	GSM8K	SVAMP	ASDiv	GSM-Hard	MATH	AIME24	AIME25	Minerva	
Vanilla	92.42	84.00	95.98	51.78	81.20	73.33	<u>66.67</u>	33.46	72.36
EnvScaler	93.10	82.67	96.47	51.18	82.40	73.33	60.00	35.66	71.85
AgenticQwen	<u>95.00</u>	89.33	97.29	55.04	88.00	60.00	60.00	30.88	71.94
AgentSkiller	83.17	71.00	92.45	39.88	73.60	66.67	56.67	31.62	64.38
Klear-AgentForge	91.21	89.67	97.70	55.42	75.20	26.67	13.33	27.21	59.55
MetaMathQA	94.84	87.00	96.72	55.42	84.00	70.00	<u>66.67</u>	34.56	73.65
ScaleQuest-Math	94.54	89.67	97.62	57.39	84.00	<u>76.67</u>	<u>66.67</u>	33.09	74.96
OpenMathInstruct-2	94.54	89.00	97.45	57.62	86.40	70.00	56.67	33.46	73.14
Reject-Sample SFT	92.49	83.67	96.14	51.48	82.40	73.33	<u>66.67</u>	31.62	72.23
Static-Pool GRPO	92.95	85.00	96.06	52.46	82.20	70.00	63.33	31.99	71.75
Off-Augment GRPO	92.42	81.33	96.06	51.71	80.40	83.33	<u>66.67</u>	33.09	73.13
R-Zero	92.80	84.67	96.31	51.55	83.20	80.00	66.67	33.46	73.58
SYNCO (Frozen Syn)	94.77	<u>91.33</u>	<u>97.78</u>	<u>61.26</u>	85.60	<u>76.67</u>	<u>66.67</u>	32.72	<u>75.85</u>
SYNCO	95.75	93.00	97.95	63.31	<u>87.80</u>	83.33	73.33	<u>35.29</u>	78.72

The recent interaction window emphasizes the Reasoner’s current behavior while retaining sufficient history to summarize task coverage, recurring weaknesses, and synthesis failures. Thus, the synthesis condition evolves from realized learning behavior: previously challenging tasks may become trivial, new weaknesses may emerge, and over-produced reasoning patterns may be suppressed.

Coupled evolution of agents and data. As the Reasoner changes, the utility of a task becomes non-stationary: $\mu(x; \pi_R^{t+1}) \neq \mu(x; \pi_R^t)$. A task that once produced mixed outcomes may later become uniformly solvable, while another may enter the learnable region as prerequisite capabilities improve. Meanwhile, the updated Synthesizer changes how the renewed learner state is translated into future tasks. The next verified distribution is therefore

$$\tilde{\mathcal{D}}_{t+1} = \{x \sim \pi_S(\cdot | c_{t+1}; \theta_S^{t+1}) \mid V(x).v = 1\}. \quad (11)$$

Compared with $\tilde{\mathcal{D}}_t$, this distribution changes through both a renewed context and an updated synthesis policy. SYNCO therefore does not merely query a fixed generator with a revised prompt; it jointly adapts the state conditioning generation and the policy that performs it.

4 Experiments

Benchmarks. We evaluate mathematical reasoning on eight benchmarks: GSM8K, SVAMP, ASDiv, GSM-Hard, MATH-500, AIME24, AIME25, and Minerva, reporting answer accuracy under a unified verification protocol. **Baselines.** We compare SYNCO against existing synthetic-data methods, including AgenticQwen, EnvScaler, AgentSkiller, Klear-AgentForge, MetaMathQA, ScaleQuest-Math, OpenMathInstruct-2, and R-Zero, together with three controlled alternatives: Static-Pool GRPO, Offline-Augment GRPO, and Reject-Sample SFT. We additionally include SYNCO (Frozen Synth), which retains learner-conditioned online synthesis but disables Synthesizer updates. **Implementation Details.** We use Qwen3-8B as the default backbone and additionally evaluate Qwen3-4B for cross-scale validation. Training is implemented with the UnityMAS-O multi-agent RL framework using GRPO. Complete benchmark descriptions, training configurations, synthesis prompts, hyperparameters, and evaluation details are provided in Appendix C.

4.1 RQ1: Does Co-Training Data Synthesis Improve Reasoning Performance?

Table 1 presents the overall reasoning performance across eight mathematical benchmarks. Across the broader comparison, existing synthetic-data methods achieve substantially lower average performance

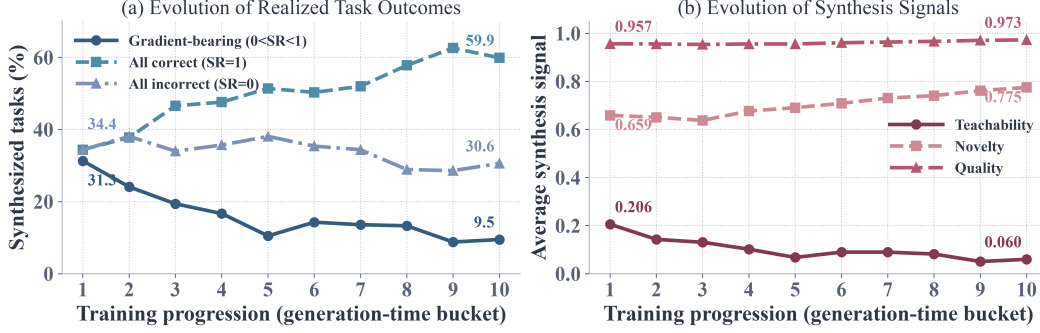


Figure 2: Evolution of the synthesized curriculum during co-training. Tasks are ordered by generation time and divided into ten equally sized buckets. **(a)** Realized Reasoner outcomes, where gradient-bearing tasks have non-identical rollout rewards (equivalently, $0 < SR < 1$ under our reward setting). **(b)** Evolution of synthesis signals, showing that task quality and novelty remain high while realized teachability declines. Generation order approximates training progression because exact iteration indices were not recorded.

than SYNCO; among them, ScaleQuest-Math is the strongest with 74.96%, while the controlled Static-Pool GRPO, Reject-Sample SFT, and Offline-Augment GRPO baselines reach 71.75%, 72.23%, and 73.13%, respectively. The frozen-Synthesizer baseline improves further to 75.85%, whereas SYNCO reaches 78.72%, outperforming the strongest external baseline by 3.76 percentage points, Frozen by 2.87 points, and Vanilla by 6.36 points. SYNCO achieves the best or tied-best result on six of the eight benchmarks and ranks second on the remaining two, indicating substantially more consistent improvements than methods that peak on individual datasets. Relative to Frozen, the gains include 2.20 points on MATH-500, 2.05 points on GSM-Hard, and 1.67 points on SVAMP, while ASDiv is already near saturation. On the advanced benchmarks, SYNCO improves AIME24 and AIME25 by 6.66 points each and Minerva by 2.57 points over Frozen. The overall results show that jointly adapting the synthesis policy yields a stronger and more broadly effective training distribution than existing synthetic-data approaches or a frozen Synthesizer. The same pattern is reproduced with Qwen3-4B, where co-training yields consistent improvements over the frozen variant, as reported in Appendix D.1.

4.2 RQ2: How Does the Synthesized Curriculum Evolve During Co-Training?

Figure 2 reveals a progressive shift in the utility of the synthesized curriculum. From the first to the final bucket, the proportion of gradient-bearing tasks decreases from 31.3% to 9.5%, while the all-correct proportion increases from 34.4% to 59.9%, indicating that an increasing fraction of generated tasks becomes uniformly solvable as the Reasoner improves. Meanwhile, task quality remains consistently high and novelty increases from 0.659 to 0.775, whereas teachability declines sharply from 0.206 to 0.060. The full-pool statistics in Appendix D. Table 5 further show that only 16.2% of the 2,940 verified tasks provide non-degenerate group-relative feedback, and that teachability strongly tracks this gradient-bearing property ($r = 0.934$), unlike novelty ($r = -0.041$). These results distinguish task validity and diversity from realized optimization utility: well-formed and novel tasks may still provide no effective policy gradient for the current Reasoner. Thus, although SYNCO achieves consistent downstream gains, maintaining an informative curriculum becomes increasingly difficult as the learner evolves, motivating synthesis objectives grounded directly in realized rollout outcomes. Additional analyses are in Appendix D.

4.3 RQ3: Where Do the Gains from Co-Training Arise?

To distinguish capability acquisition from preservation of existing knowledge, we partition each benchmark according to the initial Qwen3-8B predictions. As shown in Table 2, SYNCO and SYNCO (Frozen Synth.) remain nearly identical on the Retain subset, whereas a much larger gap emerges on examples that the initial model fails to solve. In the pooled results, the Learn subset improves from 27.11% to 32.23%, while the Retain subset changes only from 98.52% to 98.95%.

Table 2: Instance-level attribution of the terminal performance difference between SYNCO and the frozen-Synthesizer baseline. Examples are partitioned by the initial Qwen3-8B predictions: *Retain* contains initially correct examples, whereas *Learn* contains initially incorrect examples. Frozen and SYNCO report the number and percentage of solved examples within each subset. ΔQ denotes the additional examples solved by SYNCO, and *Net* sums the contributions from both subsets.

Benchmark	Retain: Initially Correct				Learn: Initially Incorrect				Net ΔQ
	<i>N</i>	Frozen	SYNCO	ΔQ	<i>N</i>	Frozen	SYNCO	ΔQ	
GSM-Hard	683	661 (96.8%)	667 (97.7%)	+6	636	147 (23.1%)	168 (26.4%)	+21	+27
MATH-500	406	401 (98.8%)	403 (99.3%)	+2	94	27 (28.7%)	36 (38.3%)	+9	+11
GSM8K	1,219	1,210 (99.3%)	1,212 (99.4%)	+2	100	40 (40.0%)	51 (51.0%)	+11	+13
SVAMP	252	250 (99.2%)	251 (99.6%)	+1	48	24 (50.0%)	28 (58.3%)	+4	+5
Pooled	2,560	2,522 (98.52%)	2,533 (98.95%)	+11	878	238 (27.11%)	283 (32.23%)	+45	+56

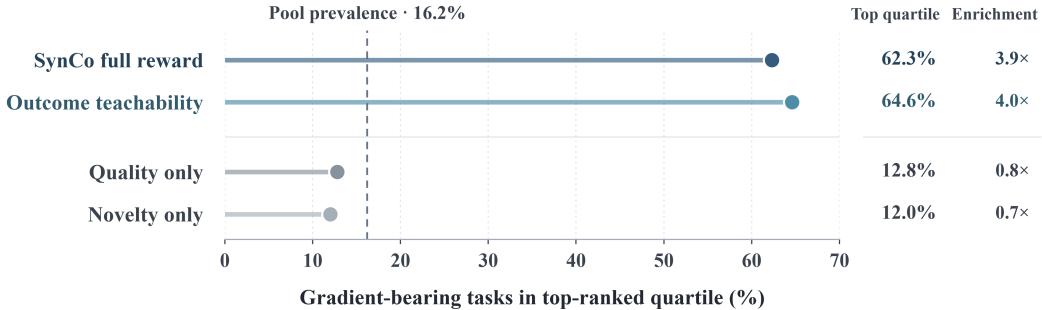


Figure 3: Counterfactual enrichment analysis of synthesis signals. Tasks are reranked offline using different signals, and the proportion of gradient-bearing tasks in the top-ranked quartile is reported. Outcome-grounded teachability strongly enriches for gradient-bearing tasks, whereas static task-level signals provide substantially weaker separation.

Correspondingly, 45 of the 56 additional solved examples arise from the Learn subset, showing that the advantage of co-training is dominated by capability acquisition rather than improved preservation of already mastered behavior. This pattern is consistent across all four benchmarks, indicating that adapting the synthesis policy does not merely reinforce the current policy but reallocates training experience toward regions where unresolved yet learnable problems remain. Together with the curriculum analysis in Section 4.2, these results suggest that learner-adaptive synthesis is particularly valuable near the evolving capability frontier, where informative training experience can be converted into new problem-solving competence. A complete benchmark-level decomposition is provided in Appendix D.3 and Figure 7.

4.4 RQ4: Which Synthesis Signals Align with Gradient-Bearing Tasks?

Figure 3 shows that outcome-grounded teachability is the synthesis signal most strongly aligned with optimization utility. Relative to the 16.2% gradient-bearing prevalence in the full task pool, ranking by the SYNCO gated reward increases this proportion to 62.3%, while teachability alone reaches 64.6%. In contrast, quality and novelty alone yield only 12.8% and 12.0%, respectively, both below the pool prevalence. The signal-level statistics in Table 6 further show that teachability strongly correlates with the gradient-bearing indicator ($r = 0.934$), whereas static task-level signals exhibit weak alignment. These results separate task feasibility from learning utility: quality and reliability help ensure trustworthy tasks, but the Reasoner’s realized rollout outcomes determine whether a task provides informative group-relative feedback. The gated reward therefore preserves the strong outcome signal while constraining optimization to valid and reliable tasks, supporting the central design of SYNCO. Because this analysis reranks an already collected task pool, it should be interpreted as a diagnostic analysis rather than a causal training ablation; additional results are provided in Appendix D.4.

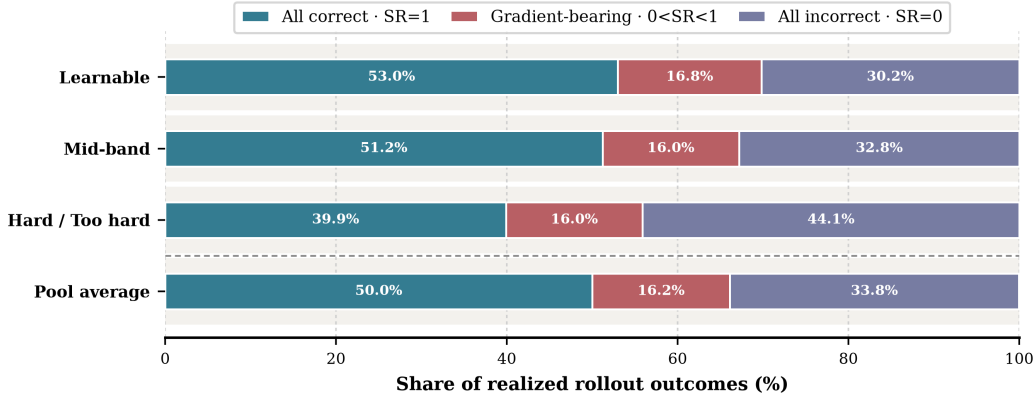


Figure 4: Realized Reasoner outcomes under the Synthesizer’s ex-ante difficulty estimates. Harder labels shift tasks from uniformly correct toward uniformly incorrect outcomes, indicating coarse difficulty awareness. However, the gradient-bearing proportion remains nearly unchanged across difficulty categories, showing that nominal difficulty does not predict realized optimization utility.

Table 3: Training cost and gradient utilization for Frozen Synthesizer and SYNCO. Times are measured per training step, and productive rate denotes the fraction of gradient-bearing tasks.

Variant	Update	Wall	Grad. Rate	Tasks/Batch
Frozen	45.8 s	341.6 s	13.7%	4.38
SYNCO	45.1 s	343.5 s	14.5%	4.64
Δ	-1.5%	+0.6%	+0.8 pp	+0.26

4.5 RQ5: Can the Synthesizer Anticipate Gradient Utility Before Rollout?

Figure 4 shows that the Synthesizer’s ex-ante estimates capture coarse task difficulty but fail to identify tasks with useful realized learning signal. As the label shifts from *learnable* to *hard/too hard*, the all-correct rate decreases from 53.0% to 39.9%, while the all-incorrect rate increases from 30.2% to 44.1%. However, this shift occurs almost entirely between the two zero-gradient extremes: the gradient-bearing rate remains nearly constant at 16.8%, 16.0%, and 16.0% across the three major difficulty groups, closely matching the 16.2% pool prevalence. Table 7 further shows that treating the *learnable* label as a predictor yields only $r = 0.009$, 16.8% precision, and 22.3% recall. In contrast, the ex-post teachability signal studied in Section 4.4 reaches $r = 0.934$ after observing the Reasoner’s rollouts. Thus, the Synthesizer can broadly anticipate whether tasks will be easier or harder, but cannot reliably identify the intermediate region that produces non-degenerate learning signals. This gap shows that nominal difficulty alone cannot substitute for learner interaction, supporting the outcome-grounded feedback design of SYNCO. Additional calibration results are provided in Appendix D.5.

4.6 RQ6: How Efficient Is Online Synthesizer–Reasoner Co-Training?

Table 3 shows that jointly optimizing the Synthesizer introduces negligible marginal overhead relative to the matched frozen-Synthesizer pipeline. The mean update time remains essentially unchanged (45.1 s vs. 45.8 s), while median end-to-end time differs by only 0.6%. Under this comparable cost, SYNCO increases the productive-task rate from 13.7% to 14.5%, corresponding to 4.64 versus 4.38 productive tasks per batch. This indicates that the benefit of co-training comes from reallocating the training distribution toward more informative interactions rather than simply increasing computation or rollout volume. Because both variants share the same Reasoner optimization budget and per-task rollout configuration, the comparison isolates the effect of adapting the synthesis policy itself. The higher productive-task rate further suggests that this adaptation improves how efficiently the fixed interaction budget is converted into useful learning signal. Thus, the gains from co-training are not

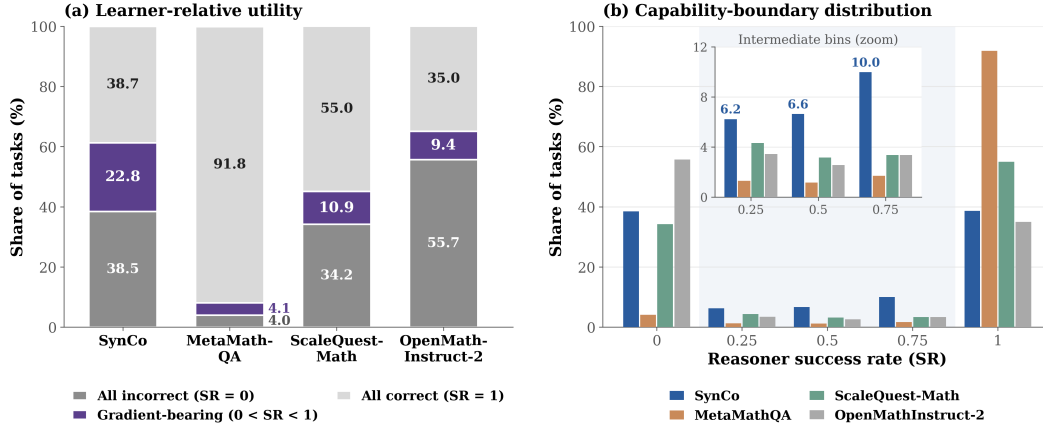


Figure 5: Learner-relative comparison of synthetic data from SYNCO, MetaMathQA, ScaleQuest-Math, and OpenMathInstruct-2 under the same Vanilla Qwen3-8B Reasoner and rollout protocol. **(a)** Distribution of realized rollout outcomes, highlighting the proportion of gradient-bearing tasks. **(b)** Full Reasoner success-rate distribution, showing how each data source is positioned relative to the learner’s capability boundary.

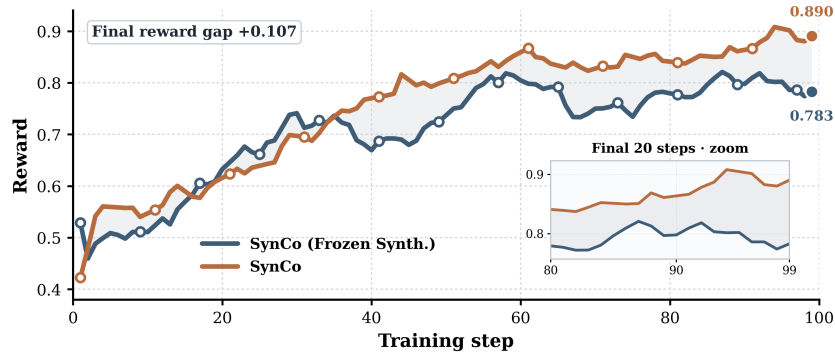


Figure 6: Reasoner reward trajectories under a frozen Synthesizer and joint Synthesizer–Reasoner co-training. Curves report 10-step moving averages. After comparable early progress, SYNCO maintains a consistently higher reward trajectory.

explained by additional per-step computation, but by a more effective online training distribution. Complete timing and data-production statistics are provided in Appendix D.6.

4.7 RQ7: Does SynCo Better Match the Learner’s Capability Boundary?

Figure 5 compares the learner-relative utility of SYNCO data with three representative static synthetic-data sources under the same Vanilla Qwen3-8B Reasoner, rollout budget, and answer checker. SYNCO places 22.8% of its tasks in the gradient-bearing region, compared with 4.1% for MetaMathQA, 10.9% for ScaleQuest-Math, and 9.4% for OpenMathInstruct-2, corresponding to a 2.1–5.5 $\times$ increase in potentially informative group-relative interactions. Panel (b) further shows that this advantage does not arise simply from generating harder problems: MetaMathQA is overwhelmingly concentrated at $SR = 1$, whereas OpenMathInstruct-2 places substantially more mass at $SR = 0$, and both therefore contain relatively few intermediate-outcome tasks. In contrast, SYNCO distributes considerably more probability mass across $SR \in \{0.25, 0.5, 0.75\}$, indicating stronger concentration near the current learner’s capability boundary. These results provide a data-side explanation for the downstream gains in Section 4.1: the benefit of learner-conditioned synthesis is not merely producing difficult or diverse problems, but generating more training experiences that are appropriately matched to the current Reasoner and yield non-degenerate learning signal.

4.8 RQ8: How Does Co-Training Affect the Reasoner’s Optimization Dynamics?

Figure 6 compares the Reasoner’s optimization trajectories under the two training regimes. Both variants improve rapidly at the beginning, indicating that online synthesized tasks already provide useful training signal with a frozen Synthesizer. The trajectories begin to separate after approximately 35 steps, after which SYNCO maintains a consistently higher moving-average reward. By the end of training, SYNCO reaches 0.890 compared with 0.783 for the frozen-Synthesizer baseline. This sustained divergence shows that the benefit of co-training emerges progressively as the synthesis policy adapts to the evolving Reasoner, rather than only appearing at a particular evaluation checkpoint. Together with the downstream gains in Section 4.1, these dynamics provide process-level evidence that joint Synthesizer–Reasoner optimization produces a more effective training trajectory.

5 Conclusion

In this paper, we introduced SYNCO, a multi-agent framework that jointly optimizes task synthesis and reasoning through shared online interaction. Across eight mathematical benchmarks, SYNCO consistently improves over a frozen Synthesizer, with gains primarily arising from previously unsolved problems. Our analyses further show that static task properties such as quality or novelty are insufficient to identify useful training experience, whereas outcome-grounded teachability is strongly aligned with gradient-bearing interactions throughout training. Co-training achieves these gains with little measurable marginal overhead, establishing adaptive data synthesis as a learnable and scalable component of self-evolving LLM training.

References

- Shuo Tang, Xianghe Pang, Zexi Liu, Bohan Tang, Rui Ye, Tian Jin, Xiaowen Dong, Yanfeng Wang, and Siheng Chen. Synthesizing post-training data for llms through multi-agent simulation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23306–23335, 2025.
- Jiaxuan Gao, Jiaao Chen, Chuyi He, Wei-Chen Wang, Shusheng Xu, Hanrui Wang, Di Jin, and Yi Wu. From self-evolving synthetic data to verifiable-reward rl: Post-training multi-turn interactive tool-using agents. *arXiv preprint arXiv:2601.22607*, 2026.
- Vardaan Pahuja, Yadong Lu, Corby Rosset, Boyu Gou, Arindam Mitra, Spencer Whitehead, Yu Su, and Ahmed Hassan. Explorer: Scaling exploration-driven web trajectory synthesis for multimodal web agents. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 6300–6323, 2025.
- Xiao Liang, Zhong-Zhi Li, Yeyun Gong, Yang Wang, Hengyuan Zhang, Ying Nian Wu, Weizhu Chen, et al. Sws: Self-aware weakness-driven problem synthesis in reinforcement learning for llm reasoning. *Advances in Neural Information Processing Systems*, 38:56801–56839, 2026.
- Yuhao Cheng, Liang Tang, Shuxian Li, Yukang Huo, Tiaonan Duan, Kaer Huang, Yanzhe Jing, and Yiqiang Yan. Evolving in tasks: Empowering the multi-modality large language model as the computer use agent. *arXiv preprint arXiv:2508.04037*, 2025.
- Shuo Zhang, Chaofa Yuan, Ryan Guo, Xiaomin Yu, Rui Xu, Zhangquan Chen, Zinuo Li, Zhi Yang, Shuhao Guan, Zhenheng Tang, et al. Evofsm: Controllable self-evolution for deep research with finite state machines. *arXiv preprint arXiv:2601.09465*, 2026a.
- Zhenrui Yue, Kartikeya Upasani, Xianjun Yang, Suyu Ge, Shaoliang Nie, Yuning Mao, Zhe Liu, and Dong Wang. Dr. zero: Self-evolving search agents without training data. *arXiv preprint arXiv:2601.07055*, 2026.
- Yunpeng Zhai, Shuchang Tao, Cheng Chen, Anni Zou, Ziqian Chen, Qingxu Fu, Shinji Mai, Li Yu, Jiaji Deng, Zouying Cao, et al. Agentevolver: Towards efficient self-evolving agent system. *arXiv preprint arXiv:2511.10395*, 2025.

- Shidong Yang, Ziyu Ma, Tongwen Huang, Yiming Hu, Yong Wang, and Xiangxiang Chu. Coevolve: Training llm agents via agent-data mutual evolution. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23015–23036, 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. *arXiv preprint arXiv:2508.05004*, 2025.
- Shaobo Wang, Zhengbo Jiao, Zifan Zhang, Yilang Peng, Xu Ze, Boyu Yang, Wei Wang, Hu Wei, and Linfeng Zhang. Socratic-zero: Bootstrapping reasoning via data-free agent co-evolution. *arXiv preprint arXiv:2509.24726*, 2025a.
- Wai-Chung Kwan, Aryo Pradipta Gema, Joshua Ong Jun Leang, and Pasquale Minervini. Scope: Self-play via co-evolving policies for open-ended tasks. *arXiv preprint arXiv:2605.31433*, 2026.
- Yizhong Wang, Yeganeh Kordi, Swaroop Mishra, Alisa Liu, Noah A Smith, Daniel Khashabi, and Hannaneh Hajishirzi. Self-instruct: Aligning language models with self-generated instructions. In *Proceedings of the 61st annual meeting of the association for computational linguistics (volume 1: long papers)*, pages 13484–13508, 2023.
- Can Xu, Qingfeng Sun, Kai Zheng, Xiubo Geng, Pu Zhao, Jiazhan Feng, Chongyang Tao, Qingwei Lin, and Daxin Jiang. Wizardlm: Empowering large pre-trained language models to follow complex instructions. In *International Conference on Learning Representations*, volume 2024, pages 30745–30766, 2024.
- Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah Goodman. Star: Bootstrapping reasoning with reasoning. *Advances in Neural Information Processing Systems*, 35:15476–15488, 2022.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James T Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. *arXiv preprint arXiv:2309.12284*, 2023.
- Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.
- Zhongzhi Li, Xuansheng Wu, Yijiang Li, Lijie Hu, and Ninghao Liu. Less is enough: synthesizing diverse data in feature space of llms. *arXiv e-prints*, pages arXiv–2602, 2026a.
- Zhibin Gou, Zhihong Shao, Yeyun Gong, Yujiu Yang, Minlie Huang, Nan Duan, Weizhu Chen, et al. Tora: A tool-integrated reasoning agent for mathematical problem solving. In *International Conference on Learning Representations*, volume 2024, pages 48362–48395, 2024.
- Ram Ramrakhya, Andrew Szot, Omar Attia, Yuhao Yang, Anh Nguyen, Bogdan Mazouze, Zhe Gan, Harsh Agrawal, and Alexander Toshev. Scaling synthetic task generation for agents via exploration. *arXiv preprint arXiv:2509.25047*, 2025.
- Ziyi Wang, Yuxuan Lu, Yimeng Zhang, Pei Chen, Ziwei Dong, Jing Huang, Jiri Gesi, Xianfeng Tang, Chen Luo, Qun Liu, et al. Trajectory2task: Training robust tool-calling agents with synthesized yet verifiable data for complex user intents. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 44021–44044, 2026a.
- Jiaao Chen, Jingyuan Qi, Mingye Gao, Wei-Chen Wang, Hanrui Wang, and Di Jin. Eigendata: A self-evolving multi-agent platform for function-calling data synthesis, auditing, and repair. *arXiv preprint arXiv:2603.05553*, 2026a.
- Peng Xia, Kaide Zeng, Jiaqi Liu, Can Qin, Fang Wu, Yiyang Zhou, Caiming Xiong, and Huaxiu Yao. Agent0: Unleashing self-evolving agents from zero data via tool-integrated reasoning. *arXiv preprint arXiv:2511.16043*, 2025.
- Guanting Dong, Junting Lu, Junjie Huang, Wanjuan Zhong, Longxiang Liu, Shijue Huang, Zhenyu Li, Yang Zhao, Xiaoshuai Song, Xiaoxi Li, et al. Agent-world: Scaling real-world environment synthesis for evolving general agent intelligence. *arXiv preprint arXiv:2604.18292*, 2026.

- Yueqin Dai, Ning Gao, Wei Zhang, Jie Wang, Zichen Luo, Jinpeng Wang, Yujie Wang, Ruiyuan Wu, and Chaozheng Wang. Sead: Self-evolving agent for multi-turn service dialogue. *arXiv preprint arXiv:2602.03548*, 2026.
- Rong Wu, Xiaoman Wang, Jianbiao Mei, Pinlong Cai, Daocheng Fu, Cheng Yang, Licheng Wen, Xuemeng Yang, Yufan Shen, Yuxin Wang, et al. Evolver: Self-evolving llm agents through an experience-driven lifecycle. *arXiv preprint arXiv:2510.16079*, 2025.
- Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.
- Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.
- Wujiang Xu, Zujie Liang, Kai Mei, Hang Gao, Juntao Tan, and Yongfeng Zhang. A-mem: Agentic memory for llm agents. *Advances in Neural Information Processing Systems*, 38:17577–17604, 2026.
- Erhan Zhang, Yiqun Chen, Zechun Niu, Wei Yang, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, and Jiaxin Mao. Oases: Outcome-aligned search-evaluation co-training for agentic search. *arXiv preprint arXiv:2604.03675*, 2026b.
- Heng Ping, Arijit Bhattacharjee, Peiyu Zhang, Shixuan Li, Wei Yang, Anzhe Cheng, Xiaole Zhang, Jesse Thomason, Ali Jannesari, Nesreen Ahmed, et al. Verimoo: A mixture-of-agents framework for spec-to-hdl generation. *arXiv preprint arXiv:2510.27617*, 2025.
- Yuehan Qin, Li Li, Linxin Song, Wei Yang, Jiate Li, Yuqing Yang, and Yue Zhao. Memory retrieval for changing preferences. *arXiv preprint arXiv:2606.02976*, 2026.
- Wei Yang, Bryce Kan, Shixuan Li, Li Li, Yuehan Qin, Jiate Li, Paul Bogdan, and Jesse Thomason. Ramem: Contextual reinstatement for long-term agentic memory. *arXiv preprint arXiv:2606.22844*, 2026b.
- Wei Yang, Shixuan Li, Heng Ping, Peiyu Zhang, Paul Bogdan, and Jesse Thomason. Auditing multi-agent llm reasoning trees outperforms majority vote and llm-as-judge. *arXiv preprint arXiv:2602.09341*, 2026c.
- Kechi Zhang, Jia Li, Ge Li, Xianjie Shi, and Zhi Jin. Codeagent: Enhancing code generation with tool-integrated agent systems for real-world repo-level coding challenges. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13643–13658, 2024.
- Hongliang He, Wenlin Yao, Kaixin Ma, Wenhao Yu, Yong Dai, Hongming Zhang, Zhenzhong Lan, and Dong Yu. Webvoyager: Building an end-to-end web agent with large multimodal models. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 6864–6890, 2024.
- Ching Chang, Yidan Shi, Defu Cao, Wei Yang, Jeehyun Hwang, Haixin Wang, Jiacheng Pang, Wei Wang, Yan Liu, Wen-Chih Peng, et al. A survey of reasoning and agentic systems in time series with large language models. *arXiv preprint arXiv:2509.11575*, 2025.
- Shawn Li, Chenxiao Yu, Han Wang, Wei Yang, Ryan Rossi, Franck Dernoncourt, Xiyang Hu, Philip Yu, Chaowei Xiao, Huan Zhang, et al. Fortis: Benchmarking over-privilege in agent skills. *arXiv preprint arXiv:2605.09163*, 2026b.
- Wei Yang, Muyan Weng, Jiacheng Pang, Defu Cao, Heng Ping, Peiyu Zhang, Shixuan Li, Yue Zhao, Qiang Yang, Mengdi Wang, et al. Toward evolutionary intelligence: Llm-based agentic systems with multi-agent reinforcement learning. *Available at SSRN 5819182*, 2025.
- Wei Yang, Defu Cao, Jiacheng Pang, Muyan Weng, and Yan Liu. Adaptive collaboration with humans: Metacognitive policy optimization for multi-agent llms with continual learning. *arXiv preprint arXiv:2603.07972*, 2026d.

- Yiqun Chen, Jinyuan Feng, Wei Yang, Meizhi Zhong, Zhengliang Shi, Rui Li, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, et al. Self-compression of chain-of-thought via multi-agent reinforcement learning. *arXiv preprint arXiv:2601.21919*, 2026b.
- Heng Ping, Peiyu Zhang, Zhenkun Wang, Shixuan Li, Anzhe Cheng, Wei Yang, Paul Bogdan, and Shahin Nazarian. Poet: Power-oriented evolutionary tuning for llm-based rtl ppa optimization. *arXiv preprint arXiv:2603.19333*, 2026.
- Zixuan Yang, Yiqun Chen, Wei Yang, Erhan Zhang, Zihan Shen, Xiaochi Wei, Yan Gao, Yi Wu, Yao Hu, and Jiaxin Mao. Tournament-grpo: Group-wise tournament rewards for reinforcement learning in open-ended long-form generation. *arXiv preprint arXiv:2605.26958*, 2026e.
- Xian Li, Ping Yu, Chunting Zhou, Timo Schick, Omer Levy, Luke Zettlemoyer, Jason E Weston, and Mike Lewis. Self-alignment with instruction backtranslation. In *The twelfth international conference on learning representations*, 2023.
- Maosong Cao, Taolin Zhang, Mo Li, Chuyu Zhang, Yunxin Liu, Haodong Duan, Songyang Zhang, and Kai Chen. Condor: Enhance llm alignment with knowledge-driven data synthesis and refinement. *arXiv preprint arXiv:2501.12273*, 2(4), 2025.
- Zeao Tu, Xiangdi Meng, Yu He, Zihan Yao, Tianyu Qi, Jun Liu, and Ming Li. Resofilter: Fine-grained synthetic data filtering for large language models through data-parameter resonance analysis. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pages 5414–5428, 2025.
- Haipeng Luo, Qingfeng Sun, Can Xu, Pu Zhao, Jian-Guang Lou, Chongyang Tao, Xiubo Geng, Qingwei Lin, Shifeng Chen, Yansong Tang, et al. Wizardmath: Empowering mathematical reasoning for large language models via reinforced evol-instruct. In *International Conference on Learning Representations*, volume 2025, pages 49573–49609, 2025.
- Xiang Yue, Xingwei Qu, Ge Zhang, Yao Fu, Wenhao Huang, Huan Sun, Yu Su, and Wenhua Chen. Mammoth: Building math generalist models through hybrid instruction tuning. In *International Conference on Learning Representations*, volume 2024, pages 40320–40341, 2024.
- Jia Li, Edward Beeching, Lewis Tunstall, Ben Lipkin, Roman Soletskyi, Shengyi Huang, Kashif Rasul, Longhui Yu, Albert Q Jiang, Ziju Shen, et al. Numinamath: The largest public dataset in ai4maths with 860k pairs of competition math problems and solutions. *Hugging Face repository*, 13(9):9, 2024.
- Qiushi Sun, Kanzhi Cheng, Zichen Ding, Chuanyang Jin, Yian Wang, Fangzhi Xu, Zhenyu Wu, Chengyou Jia, Liheng Chen, Zhoumianze Liu, et al. Os-genesis: Automating gui agent trajectory construction via reverse task synthesis. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 5555–5579, 2025.
- Rui Wang, Ce Zhang, Jun-Yu Ma, Jianshu Zhang, Hongru Wang, Yi Chen, Boyang Xue, Tianqing Fang, Zhisong Zhang, Hongming Zhang, et al. Webaggregator: Enhancing compositional reasoning capabilities of deep research agent foundation models. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 24486–24517, 2026b.
- Zhaorun Chen, Zhuokai Zhao, Kai Zhang, Bo Liu, Qi Qi, Yifan Wu, Tarun Kalluri, Sara Cao, Yuanhao Xiong, Haibo Tong, et al. Scaling agent learning via experience synthesis. *arXiv preprint arXiv:2511.03773*, 2025.
- Chanwoo Park, Seungju Han, Xingzhi Guo, Asuman E Ozdaglar, Kaiqing Zhang, and Joo-Kyung Kim. Maporl: Multi-agent post-co-training for collaborative large language models with reinforcement learning. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 30215–30248, 2025.
- 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.

- Arkil Patel, Satwik Bhattamishra, and Navin Goyal. Are nlp models really able to solve simple math word problems? In *Proceedings of the 2021 conference of the North American chapter of the association for computational linguistics: human language technologies*, pages 2080–2094, 2021.
- Shen-Yun Miao, Chao-Chun Liang, and Keh-Yih Su. A diverse corpus for evaluating and developing english math word problem solvers. In *Proceedings of the 58th annual meeting of the Association for Computational Linguistics*, pages 975–984, 2020.
- Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pages 10764–10799. PMLR, 2023.
- 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.
- 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, pages 39578–39601, 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.
- Avi Singh, John D Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J Liu, James Harrison, Jaehoon Lee, Kelvin Xu, et al. Beyond human data: Scaling self-training for problem-solving with language models. *arXiv preprint arXiv:2312.06585*, 2023.
- Yuanjie Lyu, Chengyu Wang, Haonan Zheng, Yuanhao Yue, Junbing Yan, Ming Wang, and Jun Huang. Agenticqwen: Training small agentic language models with dual data flywheels for industrial-scale tool use. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (ACL 2026)*, pages 535–551, 2026.
- Xiaoshuai Song, Haofei Chang, Guanting Dong, Yutao Zhu, Ji-Rong Wen, and Zhicheng Dou. Envscaler: Scaling tool-interactive environments for llm agent via programmatic synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, pages 8326–8357, 2026.
- Zexu Sun, Bokai Ji, Hengyi Cai, Shuaiqiang Wang, Lei Wang, Guangxia Li, and Xu Chen. Agentskiller: Scaling generalist agent intelligence through semantically integrated cross-domain data synthesis. *arXiv preprint arXiv:2602.09372*, 2026.
- Qi Wang, Hongzhi Zhang, Jia Fu, Kai Fu, Yuhui Liu, Tinghai Zhang, Chenxi Sun, Gangwei Jiang, Jingyi Tang, Xingguang Ji, et al. Klear-agentforge: Forging agentic intelligence through posttraining scaling. *arXiv preprint arXiv:2511.05951*, 2025b.
- Longhui Yu, Weisen Jiang, Han Shi, Jincheng Yu, Zhengying Liu, Yu Zhang, James Kwok, Zhenguo Li, Adrian Weller, and Weiyang Liu. Metamath: Bootstrap your own mathematical questions for large language models. In *International Conference on Learning Representations*, volume 2024, pages 45040–45061, 2024.
- Yuyang Ding, Xinyu Shi, Xiaobo Liang, Juntao Li, Zhaopeng Tu, Qiaoming Zhu, and Min Zhang. Unleashing llm reasoning capability via scalable question synthesis from scratch. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 13414–13438, 2025.
- Shubham Toshniwal, Wei Du, Ivan Moshkov, Branislav Kisacanin, Alexan Ayrapetyan, and Igor Gitman. Openmathinstruct-2: Accelerating ai for math with massive open-source instruction data. In *International Conference on Learning Representations*, volume 2025, pages 19243–19275, 2025.
- Yiqun Chen, Wei Yang, Erhan Zhang, Shijie Wang, Qi Liu, Zechun Niu, Bin Zhang, Haitao Li, Rui Li, Lingyong Yan, et al. Unitymas-o: A general rl optimization framework for llm-based multi-agent systems. *arXiv preprint arXiv:2605.26646*, 2026c.

Appendix

A Related Work

Large language models have increasingly been extended into autonomous agents capable of planning, tool use, memory, and multi-step interaction with external environments [Shinn et al., 2023, Guo et al., 2025, Xu et al., 2026, Zhang et al., 2026b, Ping et al., 2025, Qin et al., 2026, Yang et al., 2026b,c]. Building on these capabilities, multi-agent systems coordinate multiple specialized agents through communication, collaboration, and iterative feedback, and have been explored across reasoning, coding, web interaction, and other complex decision-making tasks [Zhang et al., 2024, He et al., 2024, Chang et al., 2025, Li et al., 2026b, Yang et al., 2025, 2026d, Chen et al., 2026b, Ping et al., 2026, Yang et al., 2026e]. These developments motivate learning frameworks in which agents not only solve tasks, but also actively construct and adapt the experience used for subsequent training.

A.1 Synthetic Data for LLM Post-Training

Synthetic data has become an important source of supervision for post-training large language models [Gao et al., 2026]. Early instruction-synthesis methods, including Self-Instruct [Wang et al., 2023], WizardLM [Xu et al., 2024], and instruction backtranslation [Li et al., 2023], bootstrap instruction–response pairs from seed demonstrations, teacher models, or unlabeled corpora. Subsequent work improves the realism, diversity, and coverage of generated data. MATRIX [Tang et al., 2025] constructs realistic post-training scenarios through multi-agent simulation, while FAC Synthesis [Li et al., 2026a] explicitly targets missing features in the model’s representation space. Other approaches employ iterative refinement, filtering, and autonomous data engineering to better align synthetic corpora with downstream learning objectives [Cao et al., 2025, Tu et al., 2025]. These studies establish data generation as an important dimension of LLM post-training, but typically construct an offline corpus that is subsequently consumed by a separately optimized learner.

Reasoning-oriented synthesis places additional emphasis on solution quality and answer verifiability. STaR [Zelikman et al., 2022] bootstraps reasoning traces from successful model generations, while WizardMath [Luo et al., 2025] and MetaMath [Yu et al., 2023] expand mathematical supervision through instruction evolution and multi-perspective problem augmentation. MAMmoTH [Yue et al., 2024] combines natural-language and program-based rationales, whereas ToRA [Gou et al., 2024] constructs tool-integrated reasoning trajectories. DeepSeekMath [Shao et al., 2024], NuminaMath [Li et al., 2024], and related mathematical post-training pipelines further demonstrate the effectiveness of large-scale curated or synthesized reasoning corpora paired with outcome verification. More recently, SwS [Liang et al., 2026] identifies persistent learner weaknesses and synthesizes new problems targeting the corresponding concepts. Together with reinforcement learning from verifiable rewards, these methods enable scalable reasoning supervision without requiring process-level human preference annotations.

Synthetic experience has also become central to training tool-using and interactive agents. Explorer [Pahuja et al., 2025] synthesizes diverse multimodal web trajectories through large-scale environment exploration, while OS-Genesis [Sun et al., 2025] derives executable GUI tasks from successful interaction trajectories. AutoPlay [Ramrakhya et al., 2025] explores application states before generating environment-grounded tasks; Trajectory2Task [Wang et al., 2026a] transforms valid tool trajectories into verifiable user-facing tasks; and WebAggregator [Wang et al., 2026b] constructs search tasks requiring compositional evidence aggregation. EigenData [Chen et al., 2026a] synthesizes multi-turn tool-use dialogues together with executable checkers, whereas DreamGym [Chen et al., 2025] uses a learned experience model to scale synthetic tasks and interactions. These pipelines improve feasibility, environmental grounding, diversity, and verifiability, but their generators are commonly fixed, prompt-optimized, or updated separately from the policy trained on the resulting experience.

In group-relative reinforcement learning, however, the value of a task is not determined by validity, novelty, or nominal difficulty alone. Even a well-formed task provides no discriminative policy signal when all sampled solutions are correct or all are incorrect, and its learning utility changes as the Reasoner evolves. SYNCO therefore formulates data synthesis as a trainable policy conditioned on the current learner state. By optimizing the Synthesizer through verified downstream interaction, the

training distribution can evolve according to the realized teachability of generated tasks rather than only their static data properties.

A.2 Self-Evolving LLMs and Agent–Data Co-Training

Self-evolving LLMs seek to reduce dependence on externally constructed supervision by generating and learning from their own experience [Cheng et al., 2025]. R-Zero [Huang et al., 2025] initializes independently parameterized Challenger and Solver policies that co-evolve through task generation and solution feedback. Socratic-Zero [Wang et al., 2025a] extends this paradigm with Teacher, Solver, and Generator roles, using diagnosed learner weaknesses to construct an increasingly challenging mathematical curriculum. Agent0 [Xia et al., 2025] further incorporates tool-integrated reasoning into curriculum–executor co-evolution, allowing the generated tasks to advance with the executor’s capabilities. These methods demonstrate that task proposal and task solving can form an autonomous feedback loop rather than relying on a fixed, human-authored problem distribution.

Related frameworks extend self-evolution beyond standard mathematical reasoning. Dr. Zero [Yue et al., 2026] co-evolves a question proposer and a multi-turn search agent while reducing rollout cost through hop-grouped relative optimization. SCOPE [Kwan et al., 2026] trains a Challenger and retrieval-based Solver on open-ended, document-grounded tasks, using a frozen self-judge to construct task-specific evaluation rubrics. CoEvolve [Yang et al., 2026a] extracts forgetting and uncertainty signals from agent trajectories to guide subsequent tool-use task synthesis. Agent-World [Dong et al., 2026], SEAD [Dai et al., 2026], AgentEvolver [Zhai et al., 2025], EvolveR [Wu et al., 2025], and EvoFSM [Zhang et al., 2026a] similarly evolve environments, curricula, interaction experiences, memories, or agent workflows across tool use, service dialogue, search, and general agentic settings. Collectively, this literature establishes agent–data feedback as a promising basis for autonomous and continual improvement.

The closest prior approaches adapt a proposer, teacher, or curriculum policy according to an evolving solver, commonly emphasizing frontier difficulty, weakness coverage, uncertainty, diversity, or verifier-based task quality. Such adaptation is often implemented through round-wise curriculum reconstruction, stage-wise policy optimization, or a fixed generation module augmented with learner feedback. SYNCO instead adopts a more tightly coupled formulation: the same online interaction batch produces rollout-level correctness rewards for the Reasoner and task-level, interaction-grounded rewards for the Synthesizer, after which both independently parameterized policies are updated within a unified multi-agent training process. This setting also differs from multi-agent post-co-training methods such as MAPoRL [Park et al., 2025], which jointly optimize multiple agents for collaborative response generation rather than training one policy to construct another policy’s future learning experience. Beyond final performance, we explicitly analyze whether synthesized tasks are gradient-bearing, thereby distinguishing coarse difficulty and static data quality from the realized optimization utility observed under the current Reasoner.

B Method

B.1 Problem Formulation: Agentic Data Synthesis Co-Training

We consider a self-evolving learning system composed of two trainable agents. The *Synthesizer* is parameterized by θ_S and defines a data-synthesis policy $\pi_S(\cdot; \theta_S)$, while the *Reasoner* is parameterized by θ_R and defines a task-solving policy $\pi_R(\cdot; \theta_R)$. The two agents have fully separated parameters and are optimized for different but mutually dependent objectives. The Synthesizer controls which training experiences are presented to the Reasoner, whereas the Reasoner’s behavior on those experiences determines the task-level feedback received by the Synthesizer.

At training iteration t , the system maintains an evolving synthesis context c_t that summarizes the current learning state. Conceptually, c_t captures the Reasoner’s recent task performance, capability weaknesses, the coverage of the synthetic task distribution, and feedback from previously rejected or redundant tasks. Given this context, the Synthesizer samples a structured task

$$x_t \sim \pi_S(\cdot \mid c_t; \theta_S^t), \quad (12)$$

where x_t contains a problem specification together with the information required to verify its validity and answer. Accordingly, the training distribution at iteration t is not externally fixed, but is induced

by the current synthesis policy and the current learner state:

$$\mathcal{D}_t = \pi_S(\cdot \mid c_t; \theta_S^t). \quad (13)$$

For each valid synthesized task x_t , the Reasoner produces a group of K independent solution rollouts,

$$y_t^{(k)} \sim \pi_R(\cdot \mid x_t; \theta_R^t), \quad k = 1, \dots, K. \quad (14)$$

Let a_t^* denote the verified reference answer associated with x_t . An outcome verifier evaluates each rollout and returns

$$z_t^{(k)} = \text{Check}(y_t^{(k)}, a_t^*), \quad (15)$$

where $z_t^{(k)}$ indicates whether the rollout successfully solves the task. The resulting interaction record is

$$\mathcal{I}_t = \left(c_t, x_t, \left\{ y_t^{(k)}, z_t^{(k)} \right\}_{k=1}^K \right). \quad (16)$$

The same interaction $\mathcal{I}_t$ induces agent-specific rewards for both policies. For the Reasoner, each rollout receives a task-solving reward

$$r_{R,t}^{(k)} = R_R(x_t, y_t^{(k)}, z_t^{(k)}), \quad (17)$$

which encourages correct and properly formatted solutions. For the Synthesizer, the task receives a synthesis reward

$$r_{S,t} = R_S\left(c_t, x_t, \left\{ y_t^{(k)}, z_t^{(k)} \right\}_{k=1}^K\right). \quad (18)$$

Unlike a static task-quality score, $r_{S,t}$ depends on how the *current* Reasoner actually behaves on the synthesized task. It therefore evaluates not only whether the task is valid and reliable, but also whether it provides useful training experience for the present Reasoner.

The two policies optimize their respective expected returns,

$$J_R(\theta_R) = \mathbb{E}_{x_t \sim \pi_S, y_t^{(k)} \sim \pi_R} \left[\frac{1}{K} \sum_{k=1}^K r_{R,t}^{(k)} \right], \quad (19)$$

$$J_S(\theta_S) = \mathbb{E}_{x_t \sim \pi_S, \{y_t^{(k)}\}_{k=1}^K \sim \pi_R} [r_{S,t}]. \quad (20)$$

Although the agents receive distinct rewards and do not share parameters, their optimization problems are intrinsically coupled. The distribution underlying J_R is determined by the Synthesizer through Eq. (13), while the reward underlying J_S is determined partly by the Reasoner’s rollout distribution. We jointly update the two policies from the same online interaction process:

$$\theta_R^{t+1} = \mathcal{U}_R(\theta_R^t, \mathcal{I}_t, \{r_{R,t}^{(k)}\}_{k=1}^K), \quad (21)$$

$$\theta_S^{t+1} = \mathcal{U}_S(\theta_S^t, \mathcal{I}_t, r_{S,t}), \quad (22)$$

where $\mathcal{U}_R$ and $\mathcal{U}_S$ denote the corresponding group-relative policy optimization updates.

This formulation differs fundamentally from training a task model on an independently synthesized dataset. Here, the Synthesizer directly controls the Reasoner’s online learning distribution, and the Reasoner’s realized behavior directly supervises the Synthesizer. Consequently, data synthesis and task learning form a coupled multi-agent policy optimization problem rather than two sequential training stages. The following sections instantiate this formulation by describing how the evolving Reasoner state conditions task synthesis, how shared task interactions jointly optimize both agents, and how their updates continuously reconstruct the subsequent training distribution.

B.2 Reward and Verification Details

Each synthesized task is verified before entering Reasoner training. The verifier checks schema completeness, problem well-posedness, answer consistency, output validity, and redundancy with recently generated tasks. We additionally apply an independent consensus check to filter tasks whose reference answers are insufficiently supported by independent solutions. Throughout training and

evaluation, Reasoner answers are judged using the same checker, which supports normalized exact, numerical, symbolic, and set- or tuple-level equivalence.

For the Reasoner, each rollout receives a correctness-based reward: 1.0 for a correct answer in the prescribed format, 0.9 for a correct answer recovered without that format, and 0 otherwise. For the Synthesizer, the main experiments use an outcome-grounded gated reward:

$$r_{S,t}^{\text{imm}} = G_q(r_t^{\text{quality}}) G_r(r_t^{\text{reliability}}) T(\mu_t), \quad (23)$$

where the quality and reliability gates follow

$$G(v) = \text{clip}\left(\frac{v - 0.5}{0.5}, 0, 1\right). \quad (24)$$

The task-quality score is a deterministic composite of verifier signals measuring schema completeness, problem validity, answer consistency, output format, and redundancy. The reliability score specifically reflects the internal consistency between the Synthesizer-provided reference solution and final answer. Both are computed using rule-based checks rather than an LLM judge and primarily serve as feasibility signals; exact verifier coefficients and thresholds are included in the supplementary implementation.

Learner-dependent utility is determined by realized teachability. Given the Reasoner’s empirical success rate $\mu_t = \frac{1}{K} \sum_{k=1}^K z_t^{(k)}$, we define

$$T(\mu_t) = \begin{cases} 2\mu_t, & \mu_t \leq 0.5, \\ 2(1 - \mu_t), & \mu_t > 0.5. \end{cases} \quad (25)$$

Teachability is therefore maximized when the Reasoner succeeds on half of its rollouts and decreases toward zero as the rollout group becomes uniformly correct or uniformly incorrect. The gated objective consequently separates task feasibility from learner-relative utility: quality and reliability constrain optimization toward trustworthy tasks, while the current Reasoner’s realized behavior determines whether those tasks provide informative group-relative feedback.

Tasks rejected by the verifier receive an immediate Synthesizer reward of -0.5 and are excluded from Reasoner training. The training framework additionally applies a constant 0.1 bonus to non-empty model outputs. Novelty and diagnostic-alignment signals are logged for analysis but do not directly contribute to the gated Synthesizer reward used in the main experiments.

C Experimental Details

C.1 Benchmarks

We evaluate mathematical reasoning across eight benchmarks spanning elementary word problems, competition mathematics, and higher-level quantitative reasoning. GSM8K [Cobbe et al., 2021] contains grade-school math word problems requiring multi-step arithmetic reasoning, while SVAMP [Patel et al., 2021] stresses robustness to variations in quantities and problem structure. ASDiv [Miao et al., 2020] provides a more diverse collection of arithmetic word problems with heterogeneous linguistic and reasoning patterns. GSM-Hard [Gao et al., 2023] increases the numerical difficulty of GSM8K-style problems while largely preserving their underlying reasoning structure. MATH-500 [Hendrycks et al., 2021, Lightman et al., 2024] covers competition-level algebra, geometry, number theory, counting, probability, and related topics, requiring more involved symbolic and multi-step reasoning. AIME24 and AIME25 each contain problems from the corresponding American Invitational Mathematics Examination and provide a challenging evaluation of competition-style reasoning with exact answers. Minerva [Lewkowycz et al., 2022] contains higher-level quantitative problems spanning advanced mathematics and scientific reasoning. Together, these benchmarks cover a broad difficulty range, from relatively saturated elementary reasoning to substantially more challenging problems with greater learning headroom. We use the standard test splits and report answer accuracy under the same answer-extraction and verification pipeline for all methods.

C.2 Baselines

We compare SYNCO with three representative offline alternatives, an iterative self-evolution baseline, and a matched frozen-Synthesizer baseline. *Static-Pool GRPO* constructs a synthetic task pool

before training and optimizes only the Reasoner using GRPO [Shao et al., 2024]. It isolates whether reinforcement learning over fixed synthetic data can match an adaptively evolving training distribution. *Offline-Augmented GRPO* constructs an offline task pool through mathematical question rewriting and variation inspired by MetaMath [Yu et al., 2023]. The Reasoner is subsequently trained with GRPO while the augmented task distribution remains fixed. *Rejection-Sampling SFT* samples Reasoner trajectories, retains those with verified correct answers, and fine-tunes the model on the resulting successful solutions. This baseline follows the generate–filter–fine-tune self-training paradigm of ReST^{EM} [Singh et al., 2023], without online policy optimization or adaptive synthesis. *R-Zero* [Huang et al., 2025] co-evolves a Challenger and a Solver by iteratively generating tasks near the Solver’s capability boundary and optimizing the two roles through reinforcement learning. Unlike SYNCO’s same-batch joint optimization, R-Zero follows an alternating self-evolution process in which task generation and solver improvement are updated across successive stages. Finally, SYNCO (*Frozen Synth.*) follows the same online generation and Reasoner-training pipeline as SYNCO, but keeps the Synthesizer parameters fixed. This matched comparison isolates the benefit of jointly optimizing the synthesis policy.

Further, we include two complementary groups of external baselines to provide a broader comparison with existing synthetic-data approaches.

Released-checkpoint baselines. We first evaluate four publicly released models that have undergone synthetic-data-driven agentic post-training. **AgenticQwen** [Lyu et al., 2026] trains compact agentic models through multi-round reinforcement learning with dual data flywheels, where reasoning and agentic tasks are progressively synthesized and increased in difficulty. **EnvScaler** [Song et al., 2026] programmatically synthesizes executable tool-interactive environments together with scenarios and rule-based verification functions, and uses the resulting interaction data for SFT and RL of Qwen3 models. **AgentSkiller** [Sun et al., 2026] automatically constructs executable cross-domain environments and synthesizes multi-turn interaction trajectories through structured domain modeling, semantic cross-domain composition, and execution-based validation. **Klear-AgentForge** [Wang et al., 2025b] provides an open post-training pipeline that combines large-scale synthetic-data SFT with subsequent multi-turn reinforcement learning for general-purpose agentic capabilities. For these baselines, we directly use the authors’ publicly released checkpoints without additional training or adaptation, and evaluate them using exactly the same mathematical evaluation protocol as SynCo.

Released synthetic-data baselines. We further compare against three representative publicly released mathematical synthesis corpora while controlling the downstream learner and optimization budget. **MetaMathQA** [Yu et al., 2024] expands mathematical supervision through question rephrasing and backward-reasoning-based question generation, producing a large collection of augmented mathematical problems. **ScaleQuest-Math** [Ding et al., 2025] synthesizes mathematical questions from scratch using lightweight question generators optimized through Question Fine-Tuning and Question Preference Optimization, yielding a large-scale corpus of synthetic problem–solution pairs. **OpenMathInstruct-2** [Toshniwal et al., 2025] constructs massive open mathematical supervision through problem and solution synthesis; for our comparison, we use its newly augmented synthetic problems rather than unchanged original benchmark questions. For each corpus, we sample exactly 2,940 unique synthetic problems, matching the number of verified training tasks used by SynCo, and train the same Qwen3 learner using the same GRPO implementation, rollout configuration, optimizer settings, and total optimization budget as our controlled static-data baseline.

C.3 Implementation Details

We initialize the Synthesizer and the Reasoner as two independently parameterized copies of Qwen3-8B, and additionally repeat the main comparison with Qwen3-4B. Both agents are optimized using GRPO through the UnityMAS-O [Chen et al., 2026c] training framework, with vLLM serving as the rollout backend. Each training iteration contains 32 synthesized tasks, and the Reasoner samples $K = 4$ independent trajectories for every admitted task. We use an actor learning rate of 1×10^{-6} , a 10-step warm-up, weight decay of 0.1, and no KL penalty in the reward. During training, the Synthesizer and Reasoner use temperatures of 1.0 and 0.9, respectively, with $p = 0.95$ nucleus sampling. The Qwen3-8B experiments are distributed across two 8-GPU worker groups, one for each agent. The Qwen3-4B experiments follow the same training protocol and hyperparameters except for backbone-specific inference configurations. Each synthesized task is verified for validity and answer reliability before entering Reasoner training, with an additional consensus check used to filter

unreliable reference answers. For evaluation, we use temperature 0.6 and top- p 0.95, and apply the same answer checker to all methods, supporting exact, numerical, symbolic, and set- or tuple-level equivalence. The frozen-Synthesizer baseline uses the same backbone, Reasoner optimization, rollout budget, verifier, and evaluation protocol, while keeping the Synthesizer parameters fixed.

C.4 Agent Prompt Templates

We provide the prompt templates used by the Synthesizer and the Reasoner. At each synthesis iteration, the placeholders in the Synthesizer user prompt are populated using the current synthetic-pool statistics, the Reasoner’s recent capability diagnostics, and feedback from rejected tasks. The Reasoner receives only the generated problem statement; all pedagogical metadata, reference solutions, and verification information are reserved for task validation and reward computation.

Synthesizer Prompt

Role. The Synthesizer generates exactly one original, valid, and pedagogically useful mathematical reasoning problem at each synthesis iteration. The generated task should target the current Reasoner’s learning needs, remain independently verifiable, and provide a useful group-relative learning signal.

System prompt.

You are the Synthesizer in a two-agent mathematical self-evolution system.

Your task is to generate exactly one original, valid, and pedagogically useful mathematical reasoning problem for training a Reasoner model.

The generated task should:

1. be well-posed and self-contained;
2. have a unique and independently verifiable final answer;
3. require non-trivial, multi-step reasoning;
4. be reliable enough to serve as training data;
5. differ meaningfully from recently generated tasks;
6. target weaknesses or under-explored reasoning behaviors identified in the diagnostic reports; and
7. be likely to provide a useful learning signal for the current Reasoner.

Use the following broad capabilities as inspiration, but do not mechanically cycle through them or treat them as fixed categories:

- Multi-step arithmetic word reasoning:
quantity tracking, rates, ratios, unit conversion, work and time, proportional reasoning, money or counting scenarios, and hidden intermediate quantities.
- Competition-style algebra:
equations, inequalities, substitutions, systems, functional expressions, hidden constraints, and exact simplification.
- Number theory:
divisibility, modular arithmetic, primes, remainders, parity, greatest common divisors, least common multiples, and Diophantine constraints.
- Counting, probability, and combinatorics:
case analysis, arrangements, combinations, overcounting, conditional probability, and recurrence-like reasoning.
- Geometry without diagrams:
angles, lengths, areas, similarity, coordinate geometry, circles, polygons, and algebraic geometric constraints.
- Symbolic and advanced reasoning:
sequences, functions, polynomial identities, transformations, exact symbolic computation, and calculus-like manipulation when appropriate.

Important restrictions:

- Do not copy or paraphrase known benchmark questions.
- Do not mention benchmark names in the generated problem.
- Do not generate famous contest problems or standard textbook examples.
- Do not generate ambiguous or underspecified problems.
- Do not require diagrams, external knowledge, or subjective judgment.
- Do not include the solution or answer in the problem statement.
- Do not output multiple problems.
- Do not output anything except valid JSON matching the required schema.

Difficulty calibration:

The Reasoner is already capable of solving many standard mathematical reasoning problems. Prefer tasks for which the Reasoner is likely to succeed on some, but not all, independent attempts. Tasks that are solved uniformly or failed uniformly provide little group-relative learning signal. Avoid direct applications of a single standard formula, single-step setups, or problems with an immediately obvious solution procedure.

Prefer tasks involving at least one of the following:

- a non-obvious case distinction or boundary condition;
- a hidden constraint that must be checked after an initial solution;
- multiple interacting constraints;
- a subtle parity, indexing, or limiting-case consideration; or
- symbolic or combinatorial structure requiring careful manipulation.

User prompt structure.

Current synthetic pool report:
{pool_report}

Current Reasoner diagnostic report:
{reasoner_diagnostic_report}

Recent rejection summary:
{recent_rejection_summary}

Generate exactly one new mathematical reasoning task as a JSON object with the following schema:

{output_schema}

Requirements:

- Output valid JSON only.
- Generate exactly one task.
- The problem must be original, well-posed, and self-contained.
- The problem must have a unique and verifiable final answer.
- The task should target the current Reasoner's learning needs rather than merely maximize difficulty.
- The problem statement must not reveal the solution or final answer.

Output schema.

```
{
  "teaching_intent": "...",
  "target_reasoning_style": "...",
  "expected_reasoner_weakness": "...",
  "difficulty_estimate": "...",
  "problem": "...",
  "solution": "...",
  "final_answer": "...",
  "verification_notes": "...",
  "novelty_notes": "..."
}
```

When no interaction history is available, the context fields are initialized with neutral descriptions indicating that no historical pool, Reasoner diagnostics, or rejection records are yet available.

Reasoner Prompt

Role. The Reasoner solves the mathematical problem generated by the Synthesizer. It receives only the problem statement and does not observe pedagogical metadata, reference solutions, verification notes, or other synthesis-time information.

System prompt.

You are the Reasoner in a mathematical reasoning training system.

Solve the given problem carefully.

Reason step by step.

Put the final answer inside \boxed{ }.

Do not provide multiple final answers.

User prompt structure.

Problem:
{problem}

Solve the problem step by step and put the final answer inside \boxed{ }.

Table 4: Reasoning accuracy (%) across eight mathematical reasoning benchmarks using Qwen3-4B. Vanilla denotes the initial Qwen3-4B model, while SYNCO (Frozen Synth.) retains learner-conditioned online synthesis but disables Synthesizer updates. Avg. is the unweighted average across all benchmarks. Bold and underlined values indicate the best and second-best results, respectively.

Method	Primary Benchmarks					Advanced Benchmarks			Avg.
	GSM8K	SVAMP	ASDiv	GSM-Hard	MATH-500	AIME24	AIME25	Minerva	
Vanilla	92.34	86.25	96.63	52.01	82.60	63.33	56.67	31.25	70.14
EnvScaler	89.84	82.00	95.40	42.00	82.00	73.33	70.00	31.62	70.77
AgentSkinner	79.45	72.67	92.94	37.38	71.80	63.33	60.00	29.41	63.37
SYNCO (Frozen Synth.)	<u>93.10</u>	<u>92.00</u>	<u>97.78</u>	61.33	<u>86.20</u>	66.67	60.67	<u>31.99</u>	<u>73.72</u>
SYNCO	93.71	93.00	98.28	<u>60.65</u>	86.60	<u>70.00</u>	<u>66.67</u>	32.72	75.20

Shared prompt conventions. Across the Synthesizer and Reasoner prompts, the system follows four conventions: (1) the Synthesizer must generate exactly one original, well-posed, and independently verifiable task; (2) the Synthesizer output must be valid JSON following an explicitly specified schema; (3) task generation is conditioned on synthetic-pool statistics, Reasoner diagnostics, and rejection feedback when such interaction history is available; and (4) the Reasoner receives only the generated problem statement, while pedagogical metadata, reference solutions, and verification information are reserved for task validation and reward computation.

D Extended Experimental Analysis

D.1 Generalization Across Model Scales

We further examine whether the benefits of synthesis co-training generalize across model scales by repeating the main comparison with Qwen3-4B. As shown in Table 4, the same progression observed with Qwen3-8B is reproduced: conditioning a frozen Synthesizer on the evolving Reasoner raises the average accuracy from 70.14% to 73.72%, while jointly optimizing the Synthesizer further improves it to 75.20%. Thus, co-training contributes an additional 1.48 percentage points over the matched frozen variant and 5.06 points over Vanilla. SYNCO improves over its frozen counterpart on seven of eight benchmarks and achieves the strongest overall average, outperforming EnvScaler and AgentSkinner by 4.43 and 11.83 points, respectively. While EnvScaler attains stronger results on AIME24 and AIME25, SYNCO performs more consistently across the broader benchmark suite, achieving the best result on five benchmarks and ranking second on the remaining three. The improvement is therefore not driven by a single task family: gains appear across arithmetic reasoning, algebraic and competition-style benchmarks, and Minerva, with only a small regression on GSM-Hard. These results reproduce the main Qwen3-8B finding at a smaller scale and show that the advantage of jointly adapting the synthesis policy is robust to the Reasoner backbone rather than being specific to a single model size.

D.2 How Does the Synthesized Curriculum Evolve During Co-Training?

Table 5 provides a detailed decomposition of the 2,940 verified tasks collected during co-training. We define a task as *gradient-bearing* when its Reasoner rollout group receives non-identical rewards and therefore produces non-zero group-relative advantages. Empirically, this criterion is equivalent to $0 < SR < 1$ for all verified task groups in our collected training data. Only 475 tasks (16.2%) satisfy this condition, compared with 994 (33.8%) that are uniformly incorrect and 1,471 (50.0%) that are uniformly correct. Thus, the primary bottleneck is not producing valid tasks, but continually placing them in the region where the current Reasoner receives informative group-relative learning signal.

The temporal evolution in Figure 2 shows that this region becomes increasingly difficult to maintain. Gradient-bearing tasks decrease from 31.3% in the earliest bucket to 9.5% in the final bucket, while the all-correct proportion rises from 34.4% to 59.9%. The shift is therefore driven primarily by tasks becoming uniformly solvable as the Reasoner improves, rather than by an increase in universally failed tasks. Meanwhile, task quality remains high and novelty increases, showing that a curriculum can remain valid and diverse while progressively losing optimization value. This divergence highlights

Table 5: Mechanistic analysis of 2,940 verified tasks collected during co-training. Empirically, gradient-bearing groups coincide exactly with $0 < \text{SR} < 1$ across the collected training data. Panel (a) summarizes realized rollout outcomes, Panel (b) relates synthesis signals to optimization utility, and Panel (c) compares the three major intended-difficulty groups with realized learnability; these groups cover 2,919 tasks.

(a) Realized Rollout Outcomes			
Outcome Regime	Criterion	Number of Tasks	Percentage
All incorrect	$\text{SR} = 0$	994	33.8%
Gradient-bearing	$0 < \text{SR} < 1$	475	16.2%
All correct	$\text{SR} = 1$	1,471	50.0%
(b) Synthesis Signals vs. Realized Utility			
Synthesis Signal	Correlation Target	Correlation	Observed Alignment
Teachability	Gradient-bearing indicator	+0.934	Strong positive
Novelty	Gradient-bearing indicator	−0.041	Negligible
Teachability	Realized success rate	−0.028	Negligible
(c) Intended Difficulty vs. Realized Learnability			
Synthesizer Label	Number of Tasks	Gradient-Bearing Rate	Zero-Gradient Rate
Learnable	630	16.8%	83.2%
Mid-band	1,926	16.0%	84.0%
Hard / Too hard	363	16.0%	84.0%

the non-stationarity of synthetic-data utility: a task distribution that is useful for an earlier learner may gradually fall behind its evolving capability frontier.

Panel (b) further separates static synthesis properties from learner-dependent utility. Novelty is nearly uncorrelated with the gradient-bearing indicator ($r = -0.041$), whereas outcome-grounded teachability is strongly aligned with it ($r = 0.934$), as expected from its construction using realized rollout outcomes. Teachability is also nearly uncorrelated with raw success rate ($r = -0.028$), indicating that it does not simply favor easier or harder tasks but instead emphasizes heterogeneous outcomes. Panel (c) reaches the same conclusion from the Synthesizer’s ex-ante perspective: the *learnable*, *mid-band*, and *hard/too hard* groups exhibit gradient-bearing rates of 16.8%, 16.0%, and 16.0%, respectively, all close to the 16.2% pool prevalence. Thus, intended difficulty meaningfully shifts whether tasks are solved or failed, but provides little information about whether they lie in the optimization-relevant region. Together, these analyses show that validity, novelty, and nominal difficulty characterize different properties of synthetic data, but none substitutes for observing how the current learner actually interacts with it.

D.3 Attributing Gains to Capability Acquisition

Table 2 decomposes the terminal advantage of SYNCO over SYNCO (Frozen Synth.) according to whether each evaluation example is initially solved by Qwen3-8B. Both methods are evaluated at fixed terminal checkpoints, avoiding benchmark-specific checkpoint selection. On the 2,560 initially correct examples, the two variants remain close: SYNCO retains 2,533 examples (98.95%), compared with 2,522 (98.52%) for the frozen variant, yielding 11 additional retained solutions. The difference is substantially larger on the 878 initially incorrect examples, where SYNCO solves 283 (32.23%) compared with 238 (27.11%) for Frozen. Thus, the Learn subset contributes 45 of the 56 additional solved examples, accounting for 80.4% of the overall advantage.

Figure 7 shows that this pattern holds consistently across all four benchmarks. On GSM-Hard, the Learn subset contributes +21 of the +27 net gain, compared with +6 from Retain. Similarly, Learn contributes +9 of +11 on MATH-500, +11 of +13 on GSM8K, and +4 of +5 on SVAMP. In aggregate, co-training improves retention by only +0.43 percentage points, from 98.52% to 98.95%, while improving recovery on initially incorrect examples by +5.12 points, from 27.11% to 32.23%.

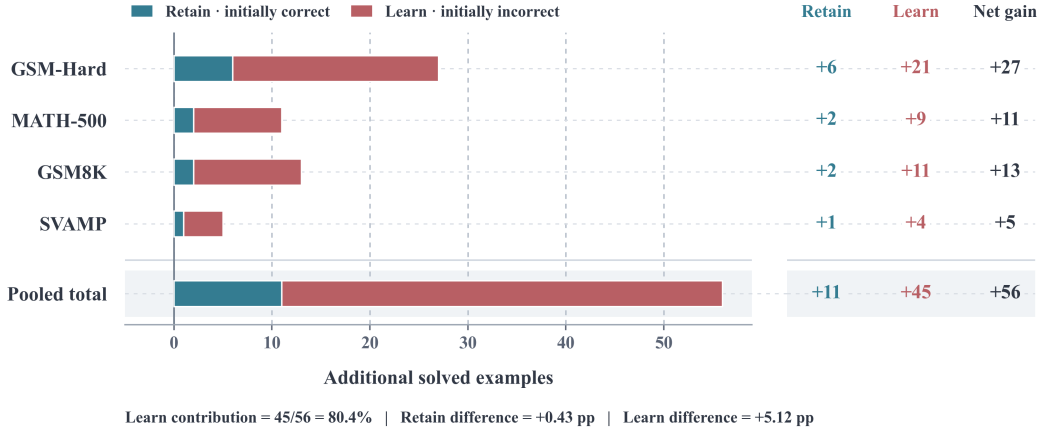


Figure 7: Attribution of SYNCO’s terminal advantage over SYNCO (Frozen Synth.). *Retain* measures additional solved examples among those initially solved by Qwen3-8B, whereas *Learn* measures gains on initially incorrect examples. Across all four benchmarks, most of the net improvement arises from the Learn subset, which contributes 45 of the 56 additional solved examples (80.4%).

The terminal advantage is therefore driven primarily by increased success on examples outside the initial model’s competence, rather than by preservation of already mastered behavior.

This decomposition complements the curriculum analysis in Section 4.2. Although only a minority of synthesized tasks produce non-degenerate group-relative feedback, the resulting optimization translates predominantly into newly solved evaluation examples rather than additional retention of previously solved ones. This pattern appears across all four benchmarks, including both relatively saturated datasets and benchmarks with greater unresolved headroom. Together with the aggregate improvements in Section 4.1, these results support the role of adaptive synthesis in capability acquisition: jointly evolving the synthesis policy helps the Reasoner expand beyond its initial competence rather than merely consolidate existing performance.

D.4 Synthesis Signals and Realized Gradient Utility

Table 6 examines how different synthesis signals relate to realized optimization utility over the 2,940 verified tasks. Quality and reliability are highly concentrated after verification, with means of 0.962 and 0.965, respectively. This concentration is expected: both signals participate in the gated synthesis reward and are also closely related to the admissibility checks that remove malformed or unreliable tasks before Reasoner training. Consequently, their weak correlations with the gradient-bearing indicator should not be interpreted as evidence that they are unnecessary; rather, once tasks pass verification, these signals provide limited discrimination over which remaining tasks will be useful for the current learner. Novelty exhibits substantially greater variation but remains nearly uncorrelated with gradient production ($r = -0.041$), showing that a task can be distinct from previous generations without providing useful optimization signal. Report alignment is similarly weakly associated with realized utility and is used as a diagnostic context signal rather than as part of the gated reward.

Outcome-grounded teachability behaves fundamentally differently. Its correlation with the gradient-bearing indicator reaches $r = 0.934$, while its correlation with raw Reasoner success rate is only $r = -0.028$. Thus, teachability does not simply favor easier or harder tasks; it identifies tasks whose rollout groups contain heterogeneous outcomes and therefore provide non-degenerate group-relative feedback. This distinction is central to the reward design of SYNCO: quality and reliability constrain optimization toward trustworthy tasks, while realized Reasoner behavior determines which of those tasks are useful for learning.

The reranking analysis in Figure 3 makes this separation explicit. Selecting the top quartile according to the actual SYNCO gated reward raises the gradient-bearing rate from the pool prevalence of 16.2% to 62.3%, a $3.9\times$ enrichment. Outcome teachability alone reaches 64.6% ($4.0\times$), whereas quality-

Table 6: Diagnostic statistics of synthesis signals over 2,940 verified tasks. Quality, reliability, and teachability constitute the gated reward used by SYNCO, whereas novelty and report alignment are recorded as diagnostic signals. Correlation is measured against the binary gradient-bearing indicator defined by the Reasoner’s group-relative learning signal.

Synthesis Signal	Mean	Std.	Corr. with Gradient-Bearing	Role	Interpretation within Verified Pool
Quality	0.962	0.030	+0.060	Reward gate	Nearly saturated after verification, with limited discrimination among valid tasks.
Reliability	0.965	0.096	+0.105	Reward gate	Ensures answer consistency and reliability, but weakly predicts realized gradient utility.
Report alignment	0.999	0.011	−0.021	Diagnostic	Nearly constant in the collected pool and weakly associated with realized utility.
Novelty	0.703	0.175	−0.041	Diagnostic	Varies substantially across tasks but does not identify gradient-bearing interactions.
Teachability	0.102	0.249	+0.934	Outcome utility	Strongly aligned with whether the task produces non-degenerate group-relative feedback.

Table 7: Calibration of the Synthesizer’s ex-ante difficulty estimates against realized Reasoner outcomes over 2,940 verified tasks. Panel (a) compares self-rated difficulty with the outcomes of $K = 4$ Reasoner rollouts, while Panel (b) contrasts the ex-ante *learnable* label with outcome-grounded teachability as signals of realized gradient utility.

(a) Ex-Ante Difficulty vs. Realized Rollout Outcomes					
Difficulty Label	Tasks	Mean SR	Gradient-Bearing	All Correct	All Incorrect
Learnable	630	0.625	16.8%	53.0%	30.2%
Mid-band	1,926	0.607	16.0%	51.1%	32.8%
Hard / Too hard	363	0.484	16.0%	39.9%	44.1%
Pool overall	2,940	0.594	16.2%	50.0%	33.8%

(b) Ex-Ante vs. Outcome-Grounded Signals					
Signal	Availability	Corr. with Gradient-Bearing	Precision	Recall	Interpretation
Self-rated “learnable”	Ex ante, before rollout	+0.009	16.8%	22.3%	Captures generation intent but provides little enrichment for gradient-bearing tasks.
Teachability	Ex post, after rollout	+0.934	–	–	Outcome-grounded signal aligned with non-degenerate group-relative feedback.

only and novelty-only rankings yield only 12.8% and 12.0%, respectively. The small difference between teachability alone and the full gated reward reflects their different roles: teachability identifies outcome-level learning utility, while the quality and reliability gates prevent optimization from favoring informative but invalid or unreliable tasks. Hence, the objective is not designed to maximize gradient yield in isolation, but to identify useful learning experiences within a feasible synthesis space.

D.5 Why Ex-Ante Difficulty Does Not Predict Gradient Utility

Table 7 compares the Synthesizer’s ex-ante difficulty estimates with the outcomes realized by the current Reasoner. The estimates exhibit meaningful coarse calibration: tasks labeled *learnable* have an average success rate of 0.625, whereas those labeled *hard/too hard* average 0.484. Consistently, the proportion of uniformly correct tasks decreases from 53.0% to 39.9%, while the uniformly incorrect proportion increases from 30.2% to 44.1%. The Synthesizer therefore captures the broad direction of task difficulty and can shift generation from easier toward harder regions of the task space.

However, calibration of average difficulty does not imply calibration of optimization utility. The gradient-bearing rates remain nearly unchanged at 16.8%, 16.0%, and 16.0% for the *learnable*, *mid-band*, and *hard/too hard* groups, respectively, all close to the 16.2% pool prevalence. Increasing nominal difficulty therefore primarily transfers probability mass from uniformly correct to uniformly incorrect outcomes rather than increasing the likelihood of mixed rollout groups. This distinction is important under group-relative optimization: a task can become substantially harder while remaining

Table 8: Detailed efficiency analysis of online Synthesizer–Reasoner co-training. Panel (a) compares per-step training cost, Panel (b) reports productive-task yield over the matched logging window, and Panel (c) summarizes task generation and verification during the 100-step SYNCO run.

(a) Marginal Training Cost				
Training Variant	Steps	Mean Update Time	Median Update Time	Median Wall Time
SYNCO (Frozen Synth.)	100	45.8 s	45.0 s	341.6 s
SYNCO	100	45.1 s	44.6 s	343.5 s
Relative difference	–	–1.5%	–0.9%	+0.6%

(b) Productive-Task Yield over the Matched Window				
Training Variant	Learnable	Mid-band	Productive	Productive Tasks / Batch
SYNCO (Frozen Synth.)	3.4%	10.3%	13.7%	4.38
SYNCO	3.9%	10.6%	14.5%	4.64
Relative change	1.14×	1.03×	1.06×	+0.26

(c) Online Data Production	
Quantity	Value
Generated task candidates	3,200
Verified tasks	2,940
Acceptance rate	91.9%
Human annotations	0
External LLM API calls	0

Update time denotes the synchronized policy-update phase. Wall-clock statistics are reported as secondary references because the two runs were executed independently. Productive-task statistics are computed over steps 13–70, the interval with valid logging for both variants. Task-generation, verification, and rollout computation are included in the reported runtime.

equally uninformative if all sampled trajectories still receive the same outcome. Ex-ante difficulty and realized gradient utility are therefore related but fundamentally different quantities.

Panel (b) quantifies this gap directly. Treating the Synthesizer’s *learnable* label as a binary predictor of gradient-bearing tasks yields a correlation of only 0.009, with 16.8% precision and 22.3% recall. Its precision is essentially identical to the underlying pool prevalence, indicating that the label provides almost no enrichment for optimization-relevant tasks. Moreover, most gradient-bearing tasks fall outside the *learnable* category, showing that useful interactions cannot be isolated reliably through self-estimated difficulty alone. The difficulty label is therefore better interpreted as a generation intent than as a prediction of downstream learning value.

By contrast, outcome-grounded teachability is computed after observing the Reasoner’s rollouts and is strongly aligned with the gradient-bearing indicator ($r = 0.934$) by construction. Its role is consequently different from ex-ante difficulty: the former identifies realized learning utility, whereas the latter only guides where in the task space synthesis should occur before interaction. Together with Section 4.4, this analysis explains why SYNCO closes the loop between generation and learner feedback. Static task attributes and self-estimated difficulty can steer exploration, but determining whether a task actually provides non-degenerate learning signal requires observing the current Reasoner’s behavior.

D.6 Training Efficiency, Gradient Utilization, and Data Production

Table 8 provides a detailed efficiency analysis of SYNCO and its frozen-Synthesizer counterpart. As shown in Panel (a), jointly optimizing the Synthesizer introduces little additional per-step computation. The mean synchronized update times are 45.1 s for SYNCO and 45.8 s for Frozen, with corresponding median update times of 44.6 s and 45.0 s. Median end-to-end step times are likewise comparable at 343.5 s and 341.6 s. Because the runs were executed independently, these small wall-clock differences should be interpreted as comparable computational profiles rather than evidence that either variant is intrinsically faster. The key result is that enabling Synthesizer optimization does not materially increase the cost of the otherwise matched online training pipeline.

Panel (b) examines how this comparable computation is converted into useful training signal over the matched logging window. The productive-task rate increases from 13.7% under Frozen to 14.5% with SYNCO, corresponding to 4.38 versus 4.64 productive tasks per batch of 32. The increase is modest, but it is obtained without increasing the rollout budget or Reasoner optimization cost. This suggests that adapting the synthesis policy improves the allocation of a fixed interaction budget: rather than generating more experience, SYNCO slightly increases the frequency with which generated tasks fall into regions that provide non-degenerate group-relative feedback. This complements the optimization dynamics in Section 4.7, where the same training process produces a progressively stronger Reasoner trajectory.

Panel (c) summarizes the resulting online data-production process. Across 100 training steps, SYNCO generates 3,200 task candidates, of which 2,940 pass verification, yielding a 91.9% acceptance rate. The resulting curriculum is generated and verified within the training loop without human annotation or external LLM API calls. These numbers should not be interpreted as zero synthesis cost: generation, verification, and Reasoner rollouts are already included in the measured runtime. Rather, they show that the evolving training distribution is constructed internally as part of the same optimization process, without requiring a separate data-generation stage.