REPRSI: CAN INTERNAL LEARNING PROGRESS GUIDE RECURSIVE SELF-IMPROVEMENT?

Anonymous authors
Paper under double-blind review

ABSTRACT

Recursive self-improvement relies on generated training experiences that support further learning. Immediate task performance can overlook internal changes that predict later capability acquisition. We study *semantic cohesion*: agreement between hidden representations of equivalent intermediate computations across contexts, relative to representations of different computations. In a controlled compositional task, semantic patching links these representations to reusable computation, and curriculum interventions show that cross-context exposure shapes cohesion and subsequent learning. Adding cohesion progress to behavioral predictors reduces mean absolute error in predicting subsequent accuracy from 19.3 to 10.1 percentage points. We introduce **REPRSI**, a **Rep**resentation-guided **R**ecursive **S**elf-**I**mprovement framework that leverages curriculum-induced semantic cohesion progress to reward a curriculum-generating teacher and select student branches for continuation, while training the student with verifiable task rewards. At matched total training compute, it improves pass@32 over a teacher rewarded by target-performance gains by *13.8, 3.1, and 5.7* percentage points on Manufactoria–HAS, MATH, and HARP, respectively. Its curricula also improve fresh students, and the recursive system reaches the comparison method's final greedy performance using *65–75% of the common full budget*. Cohesion progress thus provides useful feedback for recursive self-improvement beyond immediate task performance. The code is available at https://anonymous.4open.science/r/RERSI-BB36/

1 INTRODUCTION

Reinforcement learning with verifiable rewards (RLVR) (Wen et al., 2025; Mroueh, 2025; Liang et al., 2025; Yu et al., 2025; Zhu et al., 2025; Meng et al., 2026) has improved the mathematical reasoning and programming abilities of large language models (LLMs) (Guo et al., 2025; Team et al., 2025). Extending these gains through recursive self-improvement (RSI) requires systems to generate training experiences that support their evolving capabilities. Curriculum learning provides a foundation for this process by adapting the content and organization of training (Bengio et al., 2009; Narvekar et al., 2020; Zhou et al., 2024; Gao et al., 2026; Zhang et al., 2026). For example, SOAR trains a teacher to generate curricula by rewarding improvements in the student's performance on difficult target problems (Sundaram et al., 2026). A key question is therefore how a system should evaluate its own learning progress to decide which experiences to generate next.

Immediate task accuracy, measured after the current round of curriculum training, may not fully capture learning progress. In our experiments, curricula starting from the same student checkpoint can yield similar accuracy at this stage but lead to substantially different learning outcomes after further training. As illustrated in Figure 1, these curricula can induce different internal representation changes even when current accuracy offers little basis for choosing between them. For curriculum-based RSI, overlooking such changes can affect both what the teacher learns to generate and which student branch continues training. This motivates a concrete interpretation of *learning how to learn*: using changes in the learner's internal representations to guide the generation of future training experiences. We ask: ***Can LLMs recursively improve themselves by learning from their own representational progress?***

To investigate this question, we study *semantic cohesion*, building on work on implicit reasoning (Ye et al., 2025; Wang et al., 2024a; Lin et al., 2025; Kohli et al., 2026; Deng et al., 2023; Li et al.,

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

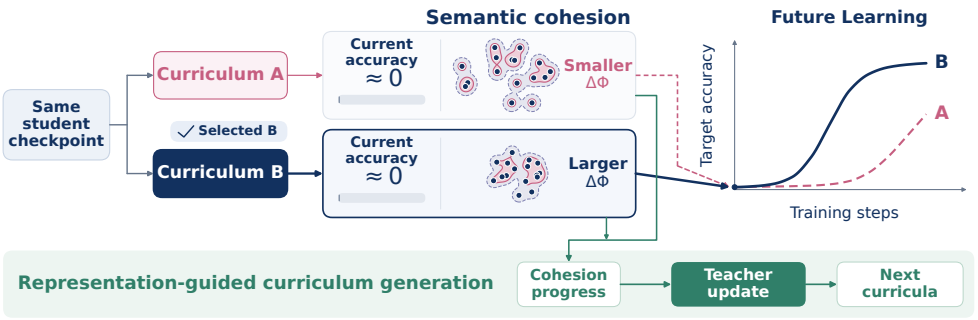

Figure 1: **Learning to generate curricula from representation feedback.**

2025). Cohesion measures agreement between hidden representations of equivalent intermediate computational states across contexts, relative to agreement between representations of different states. We examine whether progress in this quantity captures changes that support subsequent learning. In a controlled compositional setting, semantic patching provides functional evidence that the measured representations support reusable intermediate computation. Curriculum interventions further show that changing cross-context exposure affects cohesion, semantic reuse, and subsequent learning. Predictive analyses in this setting and on MATH (Hendrycks et al., 2021) show that cohesion progress adds information beyond the measured behavioral feedback. In the controlled setting, adding cohesion progress reduces the mean absolute error in predicting subsequent accuracy from 19.3 to 10.1 percentage points; on MATH, its predictive value persists beyond accuracy and likelihood-based features. These findings motivate using cohesion progress to evaluate curricula whose learning benefits are not fully reflected in current performance.

We introduce REPRSI, a ***representation-guided recursive self-improvement framework*** that turns this signal into feedback for curriculum generation. At each round, a teacher proposes candidate curricula that train student branches from a shared checkpoint. Their cohesion progress, measured on common diagnostic examples, both rewards the teacher and determines which student branch continues into the next round. The student learns from verifiable task rewards throughout. By repeating this process, the teacher learns to generate curricula according to how they change the student's representations. REPRSI thus makes the student's internal learning progress an explicit part of the recursive improvement loop.

We evaluate REPRSI across programming and mathematical reasoning under matched total training compute, accounting for teacher generation and updates, all candidate student branches, and diagnostic computation. Relative to a teacher rewarded by target-performance gains, REPRSI improves pass@32 by 13.8, 3.1, and 5.7 percentage points on Manufactoria–HAS (Sun et al., 2025), MATH (Hendrycks et al., 2021), and HARP (Yue et al., 2024), respectively. The learned curricula also improve fresh students trained from a common initialization, and the recursive system reaches the comparison method's final greedy performance using 65–75% of the common full training budget. These results show that learning to generate curricula from representation changes can improve both capability acquisition and the efficiency of recursive self-improvement.

Our main contributions are as follows:

- **Semantic cohesion as a predictor of learning.** We provide functional evidence connecting semantic cohesion to reusable intermediate computation and show that cohesion progress predicts subsequent learning beyond measured behavioral feedback. Controlled curriculum interventions further establish that cross-context exposure affects cohesion, reuse, and later learning.
- **Representation-guided recursive self-improvement.** We introduce a teacher–student framework that uses curriculum-induced cohesion progress to guide teacher updates and student continuation, while training the student with verifiable task rewards.
- **Capability acquisition and learning efficiency.** Across programming and mathematical reasoning, we demonstrate stronger capability acquisition under matched total training compute, effective curricula for fresh students, and reduced compute to reach the comparison method's final performance.

Due to space limits, we provide more related work in Appendix A.

2 SEMANTIC COHESION DURING LEARNING

2.1 THE EMERGENCE OF SEMANTIC COHESION

Intermediate semantics. A verifiable reference procedure defines intermediate state s and continuation:

$$s = F_\tau(u), \qquad y = G_\tau(s, v), \tag{1}$$

where u, v supply subcomputation and continuation inputs, and τ specifies the computation type and interface. The state is an entity, value, expression, or tuple, with canonical label $\ell = (\tau, \mathrm{can}_\tau(s))$. Equal canonical states preserve all admissible continuations:

$$\mathrm{can}_\tau(s_i) = \mathrm{can}_\tau(s_j) \implies G_\tau(s_i, v) = G_\tau(s_j, v). \tag{2}$$

Cross-context geometry and semantic patching test whether model representations respect these labels (Geiger et al., 2021; Ye et al., 2025) (Appendix B).

Analysis setting. We use `google/gemma-4-E4B-it` (Team, 2026) with explicit thinking disabled. Following Ye et al. (2025), we construct deterministic atomic relations and two-hop queries:

$$q = (e, r_1, r_2), \qquad s = r_1(e), \qquad a = r_2(s). \tag{3}$$

The student generates a; s is the intermediate state, with $u = (e, r_1)$ and $v = r_2$ in Equation (1). After single-relation training, all curriculum trials start from the same student checkpoint. Training, diagnostic, and evaluation queries are disjoint but share atomic facts. Cohesion uses fixed diagnostic examples, and $\widehat{S}_T(M)$ is target accuracy under fixed decoding (Appendix B.4).

Measuring cross-context agreement. At a fixed readout anchor (token position) and decoder block, we normalize the residual state as $z_\theta(q_i) = h_\theta(q_i)/\max\{\|h_\theta(q_i)\|_2, \epsilon\}$. Within each computation type, $\mathcal{P}^+$ pairs distinct contexts with the same semantic label; $\mathcal{P}^-$ pairs different labels. State-balanced sampling, type weights, and the layer selected by semantic patching on separate calibration trajectories are fixed (Appendix B.2; Ye et al., 2025). Define

$$C^+(\theta; \mathcal{D}) = \mathbb{E}_{(i,j)\sim\mathcal{P}^+}[z_\theta(q_i)^\top z_\theta(q_j)], \tag{4}$$

$$C^-(\theta; \mathcal{D}) = \mathbb{E}_{(i,j)\sim\mathcal{P}^-}[z_\theta(q_i)^\top z_\theta(q_j)], \tag{5}$$

$$\Phi(\theta; \mathcal{D}) = C^+(\theta; \mathcal{D}) - C^-(\theta; \mathcal{D}). \tag{6}$$

Thus Φ measures within-state relative to between-state mean cosine similarity. *Cohesion progress*, $\Delta\Phi$, is its checkpoint change on the same diagnostic examples and pairing rule.

Development during learning. By step 300, both the 50% and 90% representation-mass contours contract for one intermediate state in the shared principal-component projection (Figure 2), indicating increased cross-context concentration. This contraction is visible while target accuracy remains low, before the later improvement in task performance.

2.2 FUNCTIONAL AND LEARNING SIGNIFICANCE OF COHESION

Semantic function. Semantic patching tests whether an intermediate-state representation can support a different query's continuation. At a fixed checkpoint, we replace a recipient's hidden state with a source state for s_i at the same anchor (Ye et al., 2025). The recipient continuation r determines the counterfactual answer $r(s_i)$, while the source query has a different final answer. Relative to the unpatched recipient, semantic patching increases state-consistent answer probability by 11.2 percentage points on average, supporting reuse across query contexts.

Information about subsequent learning. In the controlled task, each curriculum trains a branch from the common M_{θ_0} with equal budget, yielding M_{θ_j} and

$$\Delta\Phi_j = \Phi(\theta_j; \mathcal{D}) - \Phi(\theta_0; \mathcal{D}). \tag{7}$$

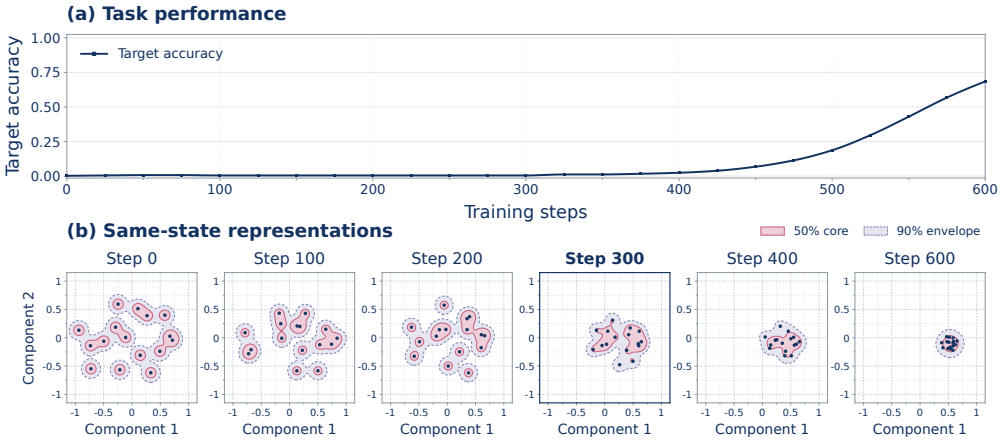

Figure 2: **Semantic cohesion during curriculum learning.** Top: target accuracy (mean ± sample standard deviation, five runs). Bottom: 50% and 90% representation-mass contours for one intermediate state in a shared principal-component projection. Cohesion is measured in the original hidden space (Appendix B.6).

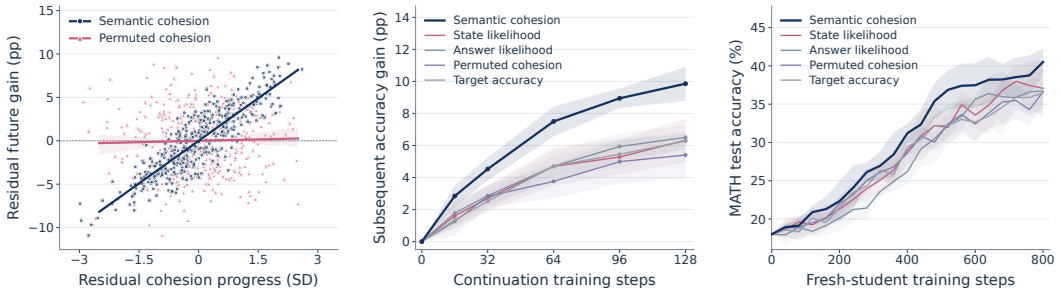

Figure 3: **Cohesion predicts subsequent learning and supports curriculum selection on MATH.** Left: residual cohesion progress and learning gain after adjusting for behavioral features. Center: continuation-training accuracy gains under different selection signals. Right: fresh-student MATH accuracy. Shading shows pointwise uncertainty intervals.

All endpoints receive the same target-training continuation and budget. Predictors use current target, curriculum-training, and intermediate-state accuracy; MATH also includes changes in reference-answer and intermediate-state log likelihood on the cohesion diagnostic items. Holding out complete runs, we predict subsequent accuracy in the controlled task and relative gains within MATH curriculum groups. Adding $\Delta\Phi$ raises ranking accuracy from 53.2% to 87.2% and from 67.4% to 86.3%, respectively, while reducing MAE (Table 1). Shuffled cohesion and permuted semantic labels provide little benefit, supporting curriculum-specific semantic progress as information for selecting curricula. Appendix B.4 gives the controlled-task protocol.

Table 1: Learning prediction: MAE (pp) and rank accuracy (%).

Predictor	MAE ↓	Rank ↑
Controlled task		
Behavioral	19.3	53.2
+ Shuffled $\Delta\Phi$	18.7	51.9
+ $\Delta\Phi$	**10.1**	**87.2**
MATH		
Behav. + likelihood	3.08	67.4
+ Label-perm. $\Delta\Phi$	3.06	66.9
+ $\Delta\Phi$	**1.41**	**86.3**

Curriculum selection under short trials. On MATH, cohesion progress retains an association with subsequent gains after behavioral adjustment using predictors fitted on separate runs; permuted labels show little residual association (Figure 3, left). Cohesion-based selection from common candidate pools improves both continuation gains and fresh-student performance over behavioral and permuted-label feedback (center and right).

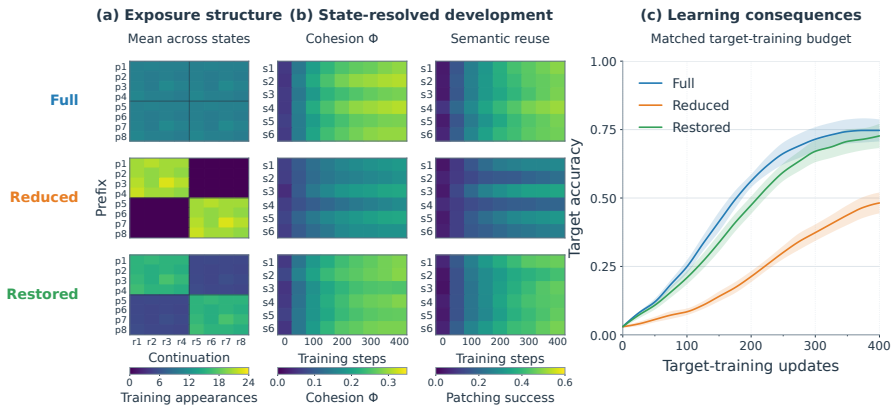

Figure 4: **Effects of cross-context exposure on semantic cohesion, reuse, and subsequent learning.** Cohesion is resolved by state as $\Phi_s = C_s^+ - C_s^-$. Reuse is greedy counterfactual accuracy after semantic patching, shown as Equation (18) divided by 100. Target accuracy is measured after 400 continuation updates. Atomic accuracy is 97.8%, 97.5%, and 97.8% for Full, Reduced, and Restored, respectively.

2.3 How Curricula Shape Cohesion

We test how the joint exposure of prefixes and continuations around shared intermediate states affects learning. Full covers all selected connections, Reduced replaces some with valid alternatives, and Restored recovers Full's connection coverage with different joint frequencies. All conditions start from the same checkpoint and train for 500 optimizer steps with five paired seeds, preserving query format, atomic facts, example count, and prefix, continuation, and state marginals (Appendix B.5).

With atomic accuracy remaining within 97.5–97.8% across conditions, Reduced yields lower cohesion, weaker semantic reuse, and lower target accuracy after 400 continuation updates than Full (48.2% versus 74.7%; Figure 4). Restoring connection coverage recovers cohesion and reuse and brings target accuracy to 72.7%, despite different joint frequencies from Full. This recovery supports cross-context coverage as a contributor to reusable computation and subsequent learning under shared atomic facts, matched marginal exposure, and equal training budgets.

> ***Reusable computation and learning potential.*** *Semantic cohesion reflects reusable intermediate computation and helps predict subsequent learning beyond the measured behavioral feedback.*

3 Representation-Guided Curriculum Generation

Overview. RepRSI uses curriculum-induced cohesion progress to update the teacher and select a student branch for continuation. The student learns from verifiable task rewards (Figure 5). Teacher $\pi^T_{\phi_0}$ and student $\pi^S_{\theta_0}$ start from the same checkpoint with separate parameters. At round t, the teacher samples $K \geq 2$ ordered curricula under fixed instructions p. It selects task families, parameters, composition, presentation, and order; fixed constructors produce questions and verifiers from these specifications. Construction rules and example count remain unchanged (Appendix C).

Each valid curriculum $\mathcal{C}_{t,k}$ trains R copies of the current student for B optimizer steps, producing endpoints $\theta'_{t,k,a}$, $a = 1, \ldots, R$.

Outer Loop: Learning from Cohesion Progress.

A shared batch $\mathcal{D}_t$ from the fixed reward-diagnostic split evaluates the starting student and all endpoints using Section 2.1's cohesion measure. Each valid curriculum receives mean progress across replicates:

$$r^{\text{rep}}_{t,k} = \frac{1}{R} \sum_{a=1}^{R} \left[\Phi(\theta'_{t,k,a}; \mathcal{D}_t) - \Phi(\theta_t; \mathcal{D}_t) \right]. \tag{8}$$

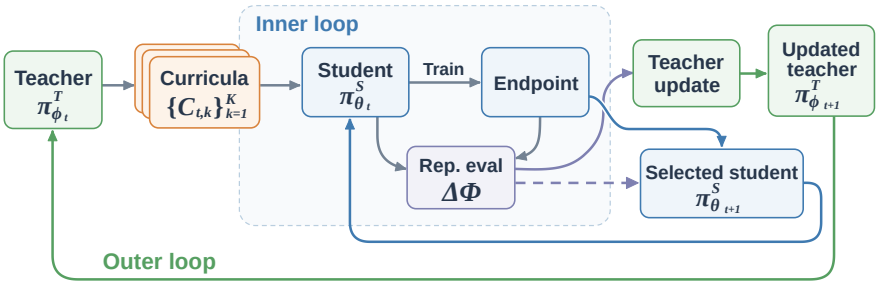

Figure 5: **Representation-guided curriculum generation.**

Shared checkpoints and diagnostics make progress comparable across curricula; replicate averaging reduces sensitivity to training randomness.

REINFORCE Leave-One-Out (RLOO) (Ahmadian et al., 2024) updates the teacher to maximize expected cohesion progress. Valid rewards are normalized before invalid proposals receive a fixed penalty. The leave-one-out advantage weights each complete curriculum's log-probability. Appendix D.5 gives the training map, objective, normalization, and update equations.

Inner Loop: Student Learning and Recursive Updates.

Branches copy the current student and optimizer state, then train with RLOO on final-answer correctness. Candidates share budget B, schedule, decoding, and optimizer settings, with replicate seeds paired across curricula. Initialization and atomic preparation are specified in Appendix D.2. Among valid candidates, the curriculum with highest mean cohesion progress determines the continuing student:

$$k_t^* = \arg\max_k r_{t,k}^{\text{rep}}, \qquad \theta_{t+1} = \theta'_{t,k_t^*,a_0}. \tag{9}$$

Replicate a_0 and tie-breaking are fixed before observing rewards. The selected endpoint and optimizer state initialize the next round; the updated teacher generates its curricula.

Training, reward diagnostics, and evaluation use disjoint splits; Appendix D.7 specifies total compute accounting.

Cohesion-guided curriculum adaptation. *Curriculum-induced cohesion progress guides teacher updates and student continuation according to the student's evolving learning state.*

4 Experiments

4.1 Experimental Setup

Models and benchmarks. We use separate Gemma 4 E4B-IT teacher and student copies with non-thinking generation. We evaluate DELTA–Manufactoria (targeting HAS) (Sun et al., 2025), MATH (Hendrycks et al., 2021), and HARP (Yue et al., 2024). Model configurations, data partitions, and representation diagnostics appear in Appendices D.2, D.3, and B.3.

Baselines. We compare with Direct target RL (Ahmadian et al., 2024), Prompted Teacher, Target-grounded Teacher (Sundaram et al., 2026), Uncertainty-driven RSI (Huang et al., 2026), and our Matched recursive search control. We also report the initial checkpoint and human-designed curricula (Sun et al., 2025; Bengio et al., 2009). Baseline sources, implementations, and adaptations are detailed in Appendix D.6.

Evaluation. We report pass@32 on the full test sets using the estimator of Chen et al. (2021). On the fixed MATH-fail@128 and HARP-fail@128 subsets, we report pass@k gains over Direct target RL in percentage points. Curriculum and compute curves use accuracy under greedy decoding. Comparisons match total training compute across five independent runs. Subset construction, evaluation protocols, compute accounting, and uncertainty estimates are detailed in Appendices D.4, D.7, and D.10.

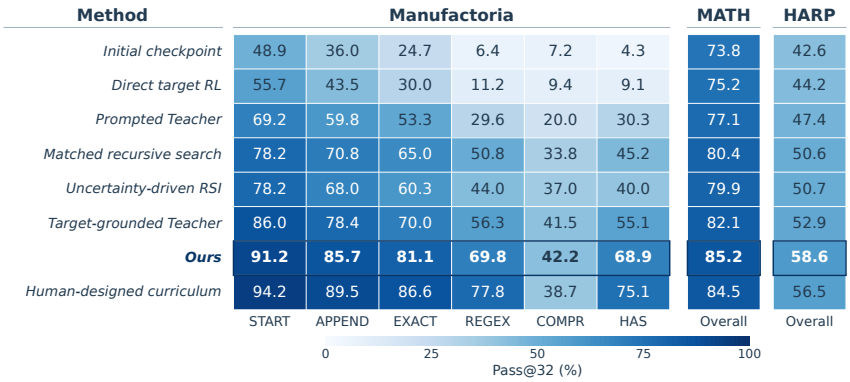

Figure 6: **Final capability at matched total training compute.** Scores are full-test pass@32. The Manufactoria target is HAS; the remaining Manufactoria columns report supporting task families.

4.2 CAPABILITY ACQUISITION ACROSS TASKS

Performance on the full test sets. At matched total training compute, representation-guided feedback improves full-test pass@32 over the Target-grounded Teacher by 13.8, 3.1, and 5.7 percentage points on Manufactoria–HAS, MATH, and HARP, respectively (Figure 6). The improvement therefore persists after accounting for the cost of obtaining representation feedback.

Acquisition on initially unsuccessful problems. The gains extend to MATH-fail@128 and HARP-fail@128, whose problems had zero successful solutions in the initial 128-sample screening (Figure 7). Against Direct target RL at equal sampling budgets, the advantage is larger at $k = 32$ (8.32 and 6.51 percentage points on MATH and HARP) than at $k = 1$ (1.00 and 0.80 points). At $k = 32$, REPRSI also exceeds the Target-grounded Teacher by 4.85 and 2.55 points, respectively. This pattern supports broader solution coverage under repeated sampling, with more modest gains in single-sample success.

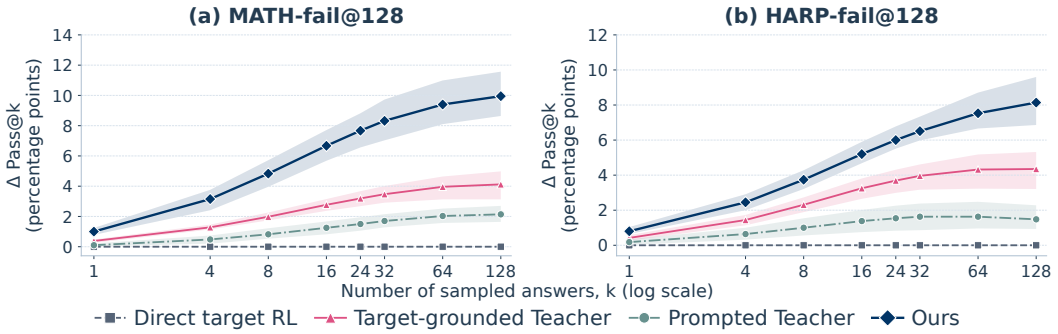

Figure 7: Capability acquisition on **(a) MATH-fail@128** and **(b) HARP-fail@128**. Gains compare the same final checkpoints with Direct target RL at each sampling budget k.

> ***Acquisition of new capabilities.*** *Representation-guided curricula improve solution coverage, including on problems with no successful solutions in the initial finite-sample evaluation.*

4.3 CURRICULUM EFFECTIVENESS AND LEARNING EFFICIENCY

Effectiveness of the learned curricula. With final teachers frozen and student initialization, example counts, optimizers, and training budgets matched, representation-guided curricula improve step-800 greedy accuracy over Target-grounded Teacher curricula by 12.2, 4.5, and 5.6 percentage points on Manufactoria–HAS, MATH, and HARP, respectively (Figure 8, top). These fresh-student gains support reusable teaching value in the learned generator (Appendix D.8).

Efficiency of recursive learning. The recursive system first reaches the Target-grounded Teacher's final mean greedy score at 65%, 75%, and 65% of the common budget on Manufactoria–HAS, MATH, and HARP, respectively. At full budget, its gains are 9.9, 4.4, and 4.4 percentage points (Figure 8,

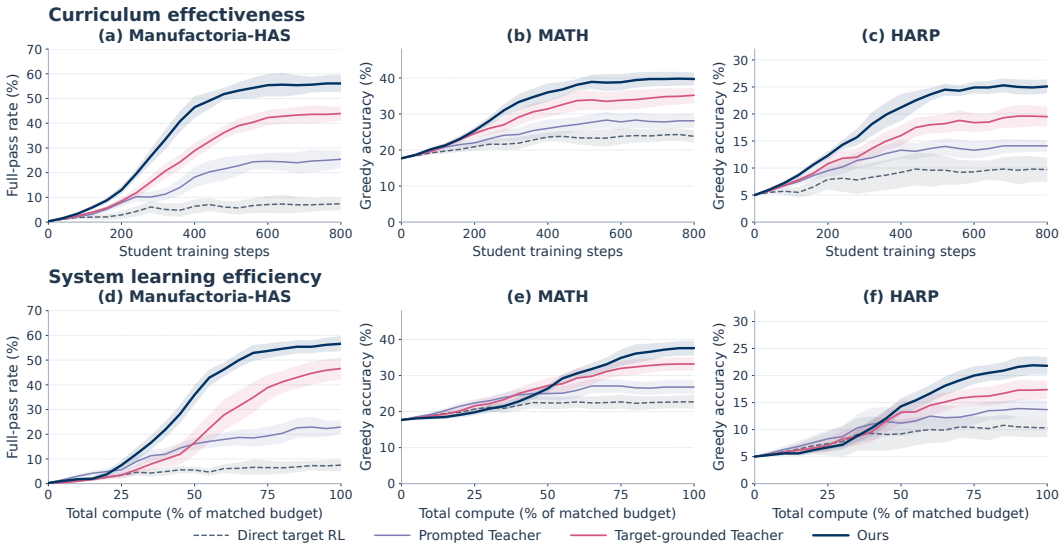

Figure 8: **Curriculum effectiveness and learning efficiency on the full test sets.** Top: fresh-student performance versus student training steps. Bottom: continuing-student performance versus cumulative total training compute, expressed as a percentage of the common budget.

bottom; Table 2). Comparing each method's first target-reaching checkpoint gives compute savings of 35.0%, 21.1%, and 35.0%. On MATH, the baseline first reaches its final mean score at 95% of the common budget, so the comparison is 75% versus 95% (Appendix D.7).

Advantage over likelihood-progress feedback. In the supplied MATH comparison, REPRSI exceeds Likelihood-progress Teacher by 3.51 percentage points in pass@32 and 2.88 points in greedy accuracy under a common 92.02 GPU-hour budget (Table 3); Appendix E provides paired results, compute breakdowns, and monitoring-pool diagnostics.

Table 2: **Full-budget performance and compute saved at matched performance.** Baseline: Target-grounded Teacher. Scores are greedy percentages. Compute compares each method's first checkpoint reaching the baseline's final mean score).

	Full budget (%) ↑		Matched performance	
Benchmark	Baseline	Ours	Ours (%)	Compute saved (%) ↑
Manufactoria–HAS	46.6	**56.5**	49.9	**35.0**
MATH	33.2	**37.6**	34.9	**21.1**
HARP	17.4	**21.8**	18.1	**35.0**

Table 3: **MATH feedback comparison at a common compute budget.** Scores are mean ± SD across five runs at the last checkpoint.

Method	pass@32 (%)	Greedy (%)
Target-grounded Teacher	81.80 ± 1.61	33.82 ± 1.33
Likelihood-progress Teacher	81.99 ± 2.05	34.22 ± 1.78
REPRSI	$\mathbf{85.50 \pm 1.81}$	$\mathbf{37.09 \pm 1.71}$
Paired gain (pp)	+3.51	+2.88

Curriculum transfer and compute efficiency. *The learned curricula benefit fresh students, while the recursive system reaches target performance with less total training compute.*

5 DISCUSSION AND CONCLUSION

5.1 CONSISTENCY OF LEARNED TEACHING POLICIES

On Manufactoria–HAS, we evaluate five independently trained teachers per feedback condition, each using a fixed sampled curriculum to train five fresh students (Figure 9). Representation-guided teachers yield closely grouped trajectories and final scores; uncertainty-driven and target-grounded feedback shows greater variation and occasional late declines. Shuffled feedback reduces performance, supporting curriculum-specific assignment of learning progress.

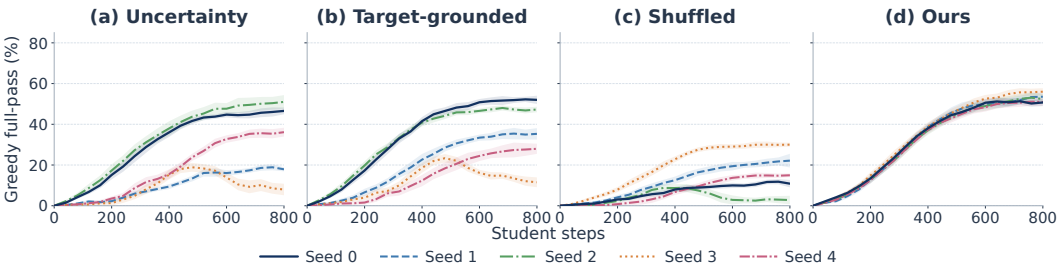

Figure 9: Teaching-policy consistency on Manufactoria–HAS. Each curve shows mean ± sample standard deviation across five fresh students using one teacher's fixed sampled curriculum. Between-curve variation includes teacher-training and curriculum-sampling differences (Appendix D.9).

Table 4: Reward-information ablations.

Teacher feedback	Fresh	Recursive
$\Delta\Phi$ (ours)	$\mathbf{56.1 \pm 3.6}$	$\mathbf{56.5 \pm 3.1}$
ΔC^+	43.1 ± 6.1	44.7 ± 4.3
Shuffled $\Delta\Phi$	24.5 ± 12.3	29.5 ± 8.5
Frozen teacher	33.8 ± 7.7	30.9 ± 6.9

Table 5: Feedback-location ablations.

Update	Select	Fresh	Recursive
$\Delta\Phi$	$\Delta\Phi$	$\mathbf{56.1 \pm 3.6}$	$\mathbf{56.5 \pm 3.1}$
$\Delta\Phi$	ΔS	50.9 ± 3.3	51.3 ± 2.4
ΔS	$\Delta\Phi$	45.2 ± 5.5	49.7 ± 8.6
ΔS	ΔS	43.9 ± 2.8	46.6 ± 4.4

5.2 What Makes Representation Feedback Useful?

Semantic specificity and curriculum credit assignment. With continuation selection fixed at $\Delta\Phi$, replacing the teacher reward with within-state progress ΔC^+ lowers Fresh and Recursive performance by 13.0 and 11.8 percentage points (Table 4), supporting the comparison between within-state and between-state agreement. Shuffling rewards across curricula reduces both outcomes, supporting curriculum-specific credit assignment. Freezing the teacher also reduces both scores, showing that learning the proposal policy contributes beyond cohesion-based selection.

Teacher updates and student continuation. With selection fixed at $\Delta\Phi$, changing the teacher reward from target-accuracy gain (ΔS) to cohesion progress raises Fresh and Recursive scores by 10.9 and 6.8 percentage points. With the teacher reward fixed at $\Delta\Phi$, changing selection from ΔS to $\Delta\Phi$ adds 5.2 points to both (Table 5). Both mixed configurations exceed all-behavioral feedback in mean performance, and using cohesion for both decisions yields the highest means.

5.3 Implications for Curriculum Learning

Cohesion progress links short curriculum trials to subsequent learning: it adds predictive information beyond the measured behavioral and likelihood features and helps select among common candidates (Table 1; Figure 3). The ablations identify two useful properties of this feedback: semantic-state discrimination and curriculum-specific credit assignment (Table 4). Using it for both teacher updates and student continuation improves fresh-student and recursive performance (Table 5). Fresh-student evaluation measures the teaching value of the learned generator, while recursive evaluation measures the benefit along the evolving student trajectory. The compute savings persist after charging diagnostics and all candidate branches (Figure 8; Table 2).

Semantic specificity and credit assignment. *Effective representation feedback distinguishes semantic states and assigns each curriculum credit for the learning progress it induces.*

5.4 Conclusion

We show that semantic cohesion progress predicts subsequent learning beyond immediate task performance and provides useful feedback for recursive self-improvement. Building on this finding, RepRSI leverages cohesion progress to guide curriculum generation and student continuation. Across programming and mathematical reasoning, it improves capability acquisition and learning efficiency, with learned curricula also benefiting fresh students. These results highlight the value of internal representation changes for guiding how LLMs learn and improve.

AI USE STATEMENT

Generative AI supported language editing and parts of experimental code. The authors reviewed all AI-assisted content and take full responsibility for the final work.

REPRODUCIBILITY STATEMENT

The main paper and appendix document the method and experimental settings; supplementary materials provide the complete anonymous source code.

REFERENCES

Arash Ahmadian, Chris Cremer, Matthias Gallé, Marzieh Fadaee, Julia Kreutzer, Olivier Pietquin, Ahmet Üstün, and Sara Hooker. Back to basics: Revisiting REINFORCE-style optimization for learning from human feedback in LLMs. In Lun-Wei Ku, Andre Martins, and Vivek Srikumar (eds.), *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 12248–12267, Bangkok, Thailand, August 2024. Association for Computational Linguistics. doi: 10.18653/v1/2024.acl-long.662. URL https://aclanthology.org/2024.acl-long.662/.

Guillaume Alain and Yoshua Bengio. Understanding intermediate layers using linear classifier probes. *ICLR*, 2017. URL https://openreview.net/forum?id=HJ4-rAVtl.

Boaz Barak, Benjamin Edelman, Surbhi Goel, Sham Kakade, Eran Malach, and Cyril Zhang. Hidden progress in deep learning: Sgd learns parities near the computational limit. *Advances in Neural Information Processing Systems*, 35:21750–21764, 2022.

Adrien Baranes and Pierre-Yves Oudeyer. Active learning of inverse models with intrinsically motivated goal exploration in robots. *Robotics and Autonomous Systems*, 61(1):49–73, January 2013. ISSN 0921-8890. doi: 10.1016/j.robot.2012.05.008. URL http://dx.doi.org/10.1016/j.robot.2012.05.008.

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

Andres Campero, Roberta Raileanu, Heinrich Kuttler, Joshua B. Tenenbaum, Tim Rocktäschel, and Edward Grefenstette. Learning with {amig}o: Adversarially motivated intrinsic goals. In *International Conference on Learning Representations*, 2021. URL https://openreview.net/forum?id=ETBc_MIMgoX.

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

Zixiang Chen, Yihe Deng, Huizhuo Yuan, Kaixuan Ji, and Quanquan Gu. Self-play fine-tuning convertsweak language models to strong language models. In *Proceedings of the 41st International Conference on Machine Learning*, ICML'24. JMLR.org, 2024.

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.

Yuntian Deng, Kiran Prasad, Roland Fernandez, Paul Smolensky, Vishrav Chaudhary, and Stuart Shieber. Implicit chain of thought reasoning via knowledge distillation. *arXiv preprint arXiv:2311.01460*, 2023.

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

Nelson Elhage, Tristan Hume, Catherine Olsson, Nicholas Schiefer, Tom Henighan, Shauna Kravec, Zac Hatfield-Dodds, Robert Lasenby, Dawn Drain, Carol Chen, et al. Toy models of superposition. *arXiv preprint arXiv:2209.10652*, 2022.

Yang Fan, Fei Tian, Tao Qin, Xiang-Yang Li, and Tie-Yan Liu. Learning to teach. *arXiv preprint arXiv:1805.03643*, 2018.

Carlos Florensa, David Held, Markus Wulfmeier, Michael Zhang, and Pieter Abbeel. Reverse curriculum generation for reinforcement learning. In *Conference on robot learning*, pp. 482–495. PMLR, 2017.

Carlos Florensa, David Held, Xinyang Geng, and Pieter Abbeel. Automatic goal generation for reinforcement learning agents. In *International conference on machine learning*, pp. 1515–1528. PMLR, 2018.

Zhaolin Gao, Joongwon Kim, Wen Sun, Thorsten Joachims, Sid Wang, Richard Yuanzhe Pang, and Liang Tan. Prompt curriculum learning for efficient llm post-training. In *International Conference on Learning Representations*, volume 2026, pp. 58614–58652, 2026.

Atticus Geiger, Hanson Lu, Thomas Icard, and Christopher Potts. Causal abstractions of neural networks. *Advances in neural information processing systems*, 34:9574–9586, 2021.

Atticus Geiger, Zhengxuan Wu, Christopher Potts, Thomas Icard, and Noah Goodman. Finding alignments between interpretable causal variables and distributed neural representations. In *Causal Learning and Reasoning*, pp. 160–187. PMLR, 2024.

Mor Geva, Roei Schuster, Jonathan Berant, and Omer Levy. Transformer feed-forward layers are key-value memories. In *Proceedings of the 2021 conference on empirical methods in natural language processing*, pp. 5484–5495, 2021.

Alex Graves, Marc G Bellemare, Jacob Menick, Remi Munos, and Koray Kavukcuoglu. Automated curriculum learning for neural networks. In *international conference on machine learning*, pp. 1311–1320. Pmlr, 2017.

Caglar Gulcehre, Tom Le Paine, Srivatsan Srinivasan, Ksenia Konyushkova, Lotte Weerts, Abhishek Sharma, Aditya Siddhant, Alex Ahern, Miaosen Wang, Chenjie Gu, et al. Reinforced self-training (rest) for language modeling. *arXiv preprint arXiv:2308.08998*, 2023.

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, Bing Xue, Bingxuan Wang, Bochao Wu, Bei Feng, Chengda Lu, Chenggang Zhao, Chengqi Deng, Chong Ruan, Damai Dai, Deli Chen, Dongjie Ji, Erhang Li, Fangyun Lin, Fucong Dai, Fuli Luo, Guangbo Hao, Guanting Chen, Guowei Li, H. Zhang, Hanwei Xu, Honghui Ding, Huazuo Gao, Hui Qu, Hui Li, Jianzhong Guo, Jiashi Li, Jingchang Chen, Jingyang Yuan, Jinhao Tu, Junjie Qiu, Junlong Li, J. L. Cai, Jiaqi Ni, Jian Liang, Jin Chen, Kai Dong, Kai Hu, Kaichao You, Kaige Gao, Kang Guan, Kexin Huang, Kuai Yu, Lean Wang, Lecong Zhang, Liang Zhao, Litong Wang, Liyue Zhang, Lei Xu, Leyi Xia, Mingchuan Zhang, Minghua Zhang, Minghui Tang, Mingxu Zhou, Meng Li, Miaojun Wang, Mingming Li, Ning Tian, Panpan Huang, Peng Zhang, Qiancheng Wang, Qinyu Chen, Qiushi Du, Ruiqi Ge, Ruisong Zhang, Ruizhe Pan, Runji Wang, R. J. Chen, R. L. Jin, Ruyi Chen, Shanghao Lu, Shangyan Zhou, Shanhuang Chen, Shengfeng Ye, Shiyu Wang, Shuiping Yu, Shunfeng Zhou, Shuting Pan, S. S. Li, Shuang Zhou, Shaoqing Wu, Tao Yun, Tian Pei, Tianyu Sun, T. Wang, Wangding Zeng, Wen Liu, Wenfeng Liang, Wenjun Gao, Wenqin Yu, Wentao Zhang, W. L. Xiao, Wei An, Xiaodong Liu, Xiaohan Wang, Xiaokang Chen, Xiaotao Nie, Xin Cheng, Xin Liu, Xin Xie, Xingchao Liu, Xinyu Yang, Xinyuan Li, Xuecheng Su, Xuheng Lin, X. Q. Li, Xiangyue Jin, Xiaojin Shen, Xiaosha Chen, Xiaowen Sun, Xiaoxiang Wang, Xinnan Song, Xinyi Zhou, Xianzu Wang, Xinxia Shan, Y. K. Li, Y. Q. Wang, Y. X. Wei, Yang Zhang, Yanhong Xu, Yao Li, Yao Zhao, Yaofeng Sun, Yaohui Wang, Yi Yu, Yichao Zhang, Yifan Shi, Yiliang Xiong, Ying He, Yishi Piao, Yisong Wang, Yixuan Tan, Yiyang Ma, Yiyuan Liu, Yongqiang Guo, Yuan Ou, Yuduan Wang, Yue Gong, Yuheng Zou, Yujia He, Yunfan Xiong, Yuxiang Luo, Yuxiang You, Yuxuan Liu, Yuyang Zhou, Y. X. Zhu, Yanping Huang, Yaohui Li, Yi Zheng, Yuchen Zhu, Yunxian Ma, Ying Tang, Yukun Zha, Yuting Yan, Z. Z. Ren, Zehui Ren, Zhangli Sha, Zhe Fu, Zhean Xu, Zhenda Xie, Zhengyan Zhang, Zhewen Hao, Zhicheng Ma, Zhigang Yan, Zhiyu Wu, Zihui Gu, Zijia Zhu, Zijun Liu, Zilin Li, Ziwei Xie, Ziyang Song, Zizheng Pan, Zhen Huang, Zhipeng Xu, Zhongyu Zhang, and Zhen Zhang. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning.

Nature, 645(8081):633–638, 2025. ISSN 1476-4687. doi: 10.1038/s41586-025-09422-z. URL http://dx.doi.org/10.1038/s41586-025-09422-z.

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.

John Hewitt and Percy Liang. Designing and interpreting probes with control tasks. In *Proceedings of the 2019 conference on empirical methods in natural language processing and the 9th international joint conference on natural language processing (emnlp-ijcnlp)*, pp. 2733–2743, 2019.

John Hewitt and Christopher D. Manning. A structural probe for finding syntax in word representations. In Jill Burstein, Christy Doran, and Thamar Solorio (eds.), *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pp. 4129–4138, Minneapolis, Minnesota, June 2019. Association for Computational Linguistics. doi: 10.18653/v1/N19-1419. URL https://aclanthology.org/N19-1419/.

Arthur E Hoerl and Robert W Kennard. Ridge regression: Biased estimation for nonorthogonal problems. *Technometrics*, 12(1):55–67, 1970.

Chengsong Huang, Wenhao Yu, Xiaoyang Wang, Hongming Zhang, Zongxia Li, Ruosen Li, Jiaxin Huang, Haitao Mi, and Dong Yu. R-zero: Self-evolving reasoning llm from zero data. In *International Conference on Learning Representations*, volume 2026, pp. 130770–130790, 2026.

Minyoung Huh, Brian Cheung, Tongzhou Wang, and Phillip Isola. The platonic representation hypothesis, 2024. URL https://arxiv.org/abs/2405.07987.

Minqi Jiang, Edward Grefenstette, and Tim Rocktäschel. Prioritized level replay. 2021. URL https://arxiv.org/abs/2010.03934.

Minqi Jiang, Michael Dennis, Jack Parker-Holder, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Replay-guided adversarial environment design. 2022. URL https://arxiv.org/abs/2110.02439.

Pascal Klink, Carlo D'Eramo, Jan R Peters, and Joni Pajarinen. Self-paced deep reinforcement learning. *Advances in Neural Information Processing Systems*, 33:9216–9227, 2020.

Harsh Kohli, srinivasan parthasarathy, Huan Sun, and Yuekun Yao. Loop, think, & generalize: Implicit reasoning in recurrent-depth transformers. In *Third Conference on Language Modeling*, 2026. URL https://openreview.net/forum?id=8fz7WRThKL.

Simon Kornblith, Mohammad Norouzi, Honglak Lee, and Geoffrey Hinton. Similarity of neural network representations revisited, 2019. URL https://arxiv.org/abs/1905.00414.

Nikolaus Kriegeskorte, Marieke Mur, and Peter A Bandettini. Representational similarity analysis-connecting the branches of systems neuroscience. *Frontiers in systems neuroscience*, 2:249, 2008.

M. Kumar, Benjamin Packer, and Daphne Koller. Self-paced learning for latent variable models. In J. Lafferty, C. Williams, J. Shawe-Taylor, R. Zemel, and A. Culotta (eds.), *Advances in Neural Information Processing Systems*, volume 23. Curran Associates, Inc., 2010. URL https://proceedings.neurips.cc/paper_files/paper/2010/file/e57c6b956a6521b28495f2886ca0977a-Paper.pdf.

Jindong Li, Yali Fu, Li Fan, Jiahong Liu, Yao Shu, Chengwei Qin, Menglin Yang, Irwin King, and Rex Ying. Implicit reasoning in large language models: A comprehensive survey. 2025. URL https://arxiv.org/abs/2509.02350.

Xiao Liang, Zhongzhi Li, Yeyun Gong, Yelong Shen, Ying Nian Wu, Zhijiang Guo, and Weizhu Chen. Beyond pass@1: Self-play with variational problem synthesis sustains rlvr. 2025. URL https://arxiv.org/abs/2508.14029.

Hunter Lightman, Vineet Kosaraju, Yura Burda, Harri Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step, 2023. URL https://arxiv.org/abs/2305.20050.

Tianhe Lin, Jian Xie, Siyu Yuan, and Deqing Yang. Implicit reasoning in transformers is reasoning through shortcuts. 2025. URL https://arxiv.org/abs/2503.07604.

Ziming Liu, Ouail Kitouni, Niklas Nolte, Eric J. Michaud, Max Tegmark, and Mike Williams. Towards understanding grokking: An effective theory of representation learning, 2022. URL https://arxiv.org/abs/2205.10343.

Ziming Liu, Eric J. Michaud, and Max Tegmark. Omnigrok: Grokking beyond algorithmic data, 2023. URL https://arxiv.org/abs/2210.01117.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. 2019. URL https://arxiv.org/abs/1711.05101.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback, 2023. URL https://arxiv.org/abs/2303.17651.

Tambet Matiisen, Avital Oliver, Taco Cohen, and John Schulman. Teacher-student curriculum learning. 2017. URL https://arxiv.org/abs/1707.00183.

Haoming Meng, Kexin Huang, Shaohang Wei, Chiyu Ma, Shuo Yang, Xue Wang, Guoyin Wang, Bolin Ding, and Jingren Zhou. Sparse but critical: A token-level analysis of distributional shifts in rlvr fine-tuning of llms. 2026. URL https://arxiv.org/abs/2603.22446.

Kevin Meng, David Bau, Alex Andonian, and Yonatan Belinkov. Locating and editing factual associations in gpt, 2023. URL https://arxiv.org/abs/2202.05262.

Youssef Mroueh. Reinforcement learning with verifiable rewards: Grpo's effective loss, dynamics, and success amplification. 2025. URL https://arxiv.org/abs/2503.06639.

Neel Nanda, Lawrence Chan, Tom Lieberum, Jess Smith, and Jacob Steinhardt. Progress measures for grokking via mechanistic interpretability, 2023. URL https://arxiv.org/abs/2301.05217.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. 2020. URL https://arxiv.org/abs/2003.04960.

Catherine Olsson, Nelson Elhage, Neel Nanda, Nicholas Joseph, Nova DasSarma, Tom Henighan, Ben Mann, Amanda Askell, Yuntao Bai, Anna Chen, Tom Conerly, Dawn Drain, Deep Ganguli, Zac Hatfield-Dodds, Danny Hernandez, Scott Johnston, Andy Jones, Jackson Kernion, Liane Lovitt, Kamal Ndousse, Dario Amodei, Tom Brown, Jack Clark, Jared Kaplan, Sam McCandlish, and Chris Olah. In-context learning and induction heads, 2022. URL https://arxiv.org/abs/2209.11895.

Pierre-Yves Oudeyer, Frdric Kaplan, and Verena V Hafner. Intrinsic motivation systems for autonomous mental development. *IEEE transactions on evolutionary computation*, 11(2):265–286, 2007.

Vardan Papyan, X. Y. Han, and David L. Donoho. Prevalence of neural collapse during the terminal phase of deep learning training. *Proceedings of the National Academy of Sciences*, 117(40): 24652–24663, 2020. ISSN 1091-6490. doi: 10.1073/pnas.2015509117. URL http://dx.doi.org/10.1073/pnas.2015509117.

Jack Parker-Holder, Minqi Jiang, Michael Dennis, Mikayel Samvelyan, Jakob Foerster, Edward Grefenstette, and Tim Rocktäschel. Evolving curricula with regret-based environment design. 2023. URL https://arxiv.org/abs/2203.01302.

Rémy Portelas, Cédric Colas, Katja Hofmann, and Pierre-Yves Oudeyer. Teacher algorithms for curriculum learning of deep rl in continuously parameterized environments. 2019. URL https://arxiv.org/abs/1910.07224.

Alethea Power, Yuri Burda, Harri Edwards, Igor Babuschkin, and Vedant Misra. Grokking: Generalization beyond overfitting on small algorithmic datasets, 2022. URL https://arxiv.org/abs/2201.02177.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Stefano Ermon, Christopher D. Manning, and Chelsea Finn. Direct preference optimization: Your language model is secretly a reward model, 2024. URL https://arxiv.org/abs/2305.18290.

Maithra Raghu, Justin Gilmer, Jason Yosinski, and Jascha Sohl-Dickstein. Svcca: Singular vector canonical correlation analysis for deep learning dynamics and interpretability, 2017. URL https://arxiv.org/abs/1706.05806.

John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms, 2017. URL https://arxiv.org/abs/1707.06347.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. Deepseekmath: Pushing the limits of mathematical reasoning in open language models, 2024. URL https://arxiv.org/abs/2402.03300.

Avi Singh, John D. Co-Reyes, Rishabh Agarwal, Ankesh Anand, Piyush Patil, Xavier Garcia, Peter J. Liu, James Harrison, Jaehoon Lee, Kelvin Xu, Aaron Parisi, Abhishek Kumar, Alex Alemi, Alex Rizkowsky, Azade Nova, Ben Adlam, Bernd Bohnet, Gamaleldin Elsayed, Hanie Sedghi, Igor Mordatch, Isabelle Simpson, Izzeddin Gur, Jasper Snoek, Jeffrey Pennington, Jiri Hron, Kathleen Kenealy, Kevin Swersky, Kshiteej Mahajan, Laura Culp, Lechao Xiao, Maxwell L. Bileschi, Noah Constant, Roman Novak, Rosanne Liu, Tris Warkentin, Yundi Qian, Yamini Bansal, Ethan Dyer, Behnam Neyshabur, Jascha Sohl-Dickstein, and Noah Fiedel. Beyond human data: Scaling self-training for problem-solving with language models, 2024. URL https://arxiv.org/abs/2312.06585.

Sainbayar Sukhbaatar, Zeming Lin, Ilya Kostrikov, Gabriel Synnaeve, Arthur Szlam, and Rob Fergus. Intrinsic motivation and automatic curricula via asymmetric self-play, 2018. URL https://arxiv.org/abs/1703.05407.

Yiyou Sun, Yuhan Cao, Pohao Huang, Haoyue Bai, Hannaneh Hajishirzi, Nouha Dziri, and Dawn Song. Rl grokking recipe: How does rl unlock and transfer new algorithms in llms? 2025. URL https://arxiv.org/abs/2509.21016.

Shobhita Sundaram, John Quan, Ariel Kwiatkowski, Kartik Ahuja, Yann Ollivier, and Julia Kempe. Teaching models to teach themselves: Reasoning at the edge of learnability. 2026. URL https://arxiv.org/abs/2601.18778.

Gemma Team. Gemma 4 technical report, 2026. URL https://arxiv.org/abs/2607.02770.

Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, Chuning Tang, Congcong Wang, Dehao Zhang, Enming Yuan, Enzhe Lu, Fengxiang Tang, Flood Sung, Guangda Wei, Guokun Lai, Haiqing Guo, Han Zhu, Hao Ding, Hao Hu, Hao Yang, Hao Zhang, Haotian Yao, Haotian Zhao, Haoyu Lu, Haoze Li, Haozhen Yu, Hongcheng Gao, Huabin Zheng, Huan Yuan, Jia Chen, Jianhang Guo, Jianlin Su, Jianzhou Wang, Jie Zhao, Jin Zhang, Jingyuan Liu, Junjie Yan, Junyan Wu, Lidong Shi, Ling Ye, Longhui Yu, Mengnan Dong, Neo Zhang, Ningchen Ma, Qiwei Pan, Qucheng Gong, Shaowei Liu, Shengling Ma, Shupeng Wei, Sihan Cao, Siying Huang, Tao Jiang, Weihao Gao, Weimin Xiong, Weiran He, Weixiao Huang, Weixin Xu, Wenhao Wu, Wenyang He, Xianghui Wei, Xianqing Jia, Xingzhe Wu, Xinran Xu, Xinxing Zu, Xinyu Zhou, Xuehai Pan, Y. Charles, Yang Li, Yangyang Hu, Yangyang Liu, Yanru Chen, Yejie Wang, Yibo Liu, Yidao Qin, Yifeng Liu, Ying Yang, Yiping Bao, Yulun Du, Yuxin Wu, Yuzhi Wang, Zaida Zhou, Zhaoji Wang, Zhaowei Li, Zhen Zhu, Zheng Zhang, Zhexu Wang, Zhilin Yang, Zhiqi Huang, Zihao Huang, Ziyao Xu,

Zonghan Yang, and Zongyu Lin. Kimi k1.5: Scaling reinforcement learning with llms. 2025. URL https://arxiv.org/abs/2501.12599.

Jonathan Uesato, Nate Kushman, Ramana Kumar, Francis Song, Noah Siegel, Lisa Wang, Antonia Creswell, Geoffrey Irving, and Irina Higgins. Solving math word problems with process- and outcome-based feedback, 2022. URL https://arxiv.org/abs/2211.14275.

Boshi Wang, Xiang Yue, Yu Su, and Huan Sun. Grokked transformers are implicit reasoners: A mechanistic journey to the edge of generalization. 2024a. URL https://arxiv.org/abs/2405.15071.

Kevin Wang, Alexandre Variengien, Arthur Conmy, Buck Shlegeris, and Jacob Steinhardt. Interpretability in the wild: a circuit for indirect object identification in gpt-2 small, 2022. URL https://arxiv.org/abs/2211.00593.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. Paired open-ended trailblazer (poet): Endlessly generating increasingly complex and diverse learning environments and their solutions, 2019. URL https://arxiv.org/abs/1901.01753.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeff Clune, and Kenneth O. Stanley. Enhanced poet: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. 2020. URL https://arxiv.org/abs/2003.08536.

Tianlu Wang, Ilia Kulikov, Olga Golovneva, Ping Yu, Weizhe Yuan, Jane Dwivedi-Yu, Richard Yuanzhe Pang, Maryam Fazel-Zarandi, Jason Weston, and Xian Li. Self-taught evaluators, 2024b. URL https://arxiv.org/abs/2408.02666.

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, 2023. URL https://arxiv.org/abs/2212.10560.

Xumeng Wen, Zihan Liu, Shun Zheng, Shengyu Ye, Zhirong Wu, Yang Wang, Zhijian Xu, Xiao Liang, Junjie Li, Ziming Miao, Jiang Bian, and Mao Yang. Reinforcement learning with verifiable rewards implicitly incentivizes correct reasoning in base llms. 2025. URL https://arxiv.org/abs/2506.14245.

Jiaran Ye, Zijun Yao, Zhidian Huang, Liangming Pan, Jinxin Liu, Yushi Bai, Amy Xin, Weichuan Liu, Xiaoyin Che, Lei Hou, and Juanzi Li. How do transformers learn implicit reasoning? 2025. URL https://arxiv.org/abs/2505.23653.

Tianyu Yu, Bo Ji, Shouli Wang, Shu Yao, Zefan Wang, Ganqu Cui, Lifan Yuan, Ning Ding, Yuan Yao, Zhiyuan Liu, Maosong Sun, and Tat-Seng Chua. Rlpr: Extrapolating rlvr to general domains without verifiers. 2025. URL https://arxiv.org/abs/2506.18254.

Weizhe Yuan, Richard Yuanzhe Pang, Kyunghyun Cho, Xian Li, Sainbayar Sukhbaatar, Jing Xu, and Jason Weston. Self-rewarding language models. 2025. URL https://arxiv.org/abs/2401.10020.

Albert S. Yue, Lovish Madaan, Ted Moskovitz, DJ Strouse, and Aaditya K. Singh. Harp: A challenging human-annotated math reasoning benchmark. 2024. URL https://arxiv.org/abs/2412.08819.

Yang Yue, Zhiqi Chen, Rui Lu, Andrew Zhao, Zhaokai Wang, Yang Yue, Shiji Song, and Gao Huang. Does reinforcement learning really incentivize reasoning capacity in llms beyond the base model?, 2025. URL https://arxiv.org/abs/2504.13837.

Eric Zelikman, Yuhuai Wu, Jesse Mu, and Noah D. Goodman. Star: Bootstrapping reasoning with reasoning, 2022. URL https://arxiv.org/abs/2203.14465.

Eric Zelikman, Georges Harik, Yijia Shao, Varuna Jayasiri, Nick Haber, and Noah D. Goodman. Quiet-star: Language models can teach themselves to think before speaking, 2024. URL https://arxiv.org/abs/2403.09629.

Dan Zhang, Sining Zhoubian, Ziniu Hu, Yisong Yue, Yuxiao Dong, and Jie Tang. Rest-mcts*: Llm self-training via process reward guided tree search, 2024. URL https://arxiv.org/abs/2406.03816.

Fred Zhang and Neel Nanda. Towards best practices of activation patching in language models: Metrics and methods. In *International Conference on Learning Representations*, volume 2024, pp. 1651–1678, 2024.

Yang Zhang, Amr Mohamed, Hadi Abdine, Guokan Shang, and Michalis Vazirgiannis. Beyond random sampling: Efficient language model pretraining via curriculum learning. 2026. URL https://arxiv.org/abs/2506.11300.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute zero: Reinforced self-play reasoning with zero data, 2025. URL https://arxiv.org/abs/2505.03335.

Yuwei Zhou, Zirui Pan, Xin Wang, Hong Chen, Haoyang Li, Yanwen Huang, Zhixiao Xiong, Fangzhou Xiong, Peiyang Xu, Shengnan Liu, and Wenwu Zhu. CurBench: Curriculum learning benchmark. In Ruslan Salakhutdinov, Zico Kolter, Katherine Heller, Adrian Weller, Nuria Oliver, Jonathan Scarlett, and Felix Berkenkamp (eds.), *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62088–62107. PMLR, 21–27 Jul 2024. URL https://proceedings.mlr.press/v235/zhou24o.html.

Hanqing Zhu, Zhenyu Zhang, Hanxian Huang, DiJia Su, Zechun Liu, Jiawei Zhao, Igor Fedorov, Hamed Pirsiavash, Zhizhou Sha, Jinwon Lee, et al. The path not taken: Rlvr provably learns off the principals. *arXiv preprint arXiv:2511.08567*, 2025.

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

APPENDIX CONTENTS

A Related Work . . . 17
B Diagnostic Semantics and Representation Measurement . . . 18
B.1 Reference Computations and Semantic Labels . . . 18
B.2 Pair Sampling, Readout, and Aggregation . . . 19
B.3 Domain-Specific Diagnostics . . . 19
B.4 Prediction of Subsequent Learning . . . 20
B.5 Cross-Context Exposure Intervention . . . 22
B.6 Statistical Reporting for the Mechanism Figures . . . 23
C Teacher Outputs and Curriculum Construction . . . 24
C.1 Specification Format and Teacher Instructions . . . 24
C.2 Admissible Construction Spaces . . . 24
C.3 Construction Failures and Training Execution . . . 25
D Experimental Details . . . 25
D.1 Experiment Scope and Terminology . . . 25
D.2 Models and Initialization . . . 26
D.3 Benchmark Versions and Data Partitions . . . 26
D.4 Construction of the fail@128 Target Subsets . . . 27
D.5 Fixed Training Configuration . . . 28
D.6 Baseline Implementations . . . 29
D.7 Evaluation and Compute Accounting . . . 30
D.8 Fresh-Student Curriculum Evaluation . . . 31
D.9 Repeated-Student Evaluation of Teacher Consistency . . . 31
D.10 Result Aggregation and Figure Specification . . . 32
E Additional MATH Feedback Comparisons . . . 32

A RELATED WORK

Curriculum learning and learning-progress feedback. Curriculum learning organizes training experience around the learner's capabilities (Bengio et al., 2009; Narvekar et al., 2020). Self-paced learning selects examples using the current model, while self-paced reinforcement learning adapts a task distribution toward a target distribution (Kumar et al., 2010; Klink et al., 2020). Learning-progress signals have long guided exploration: Intelligent Adaptive Curiosity measures prediction improvement, and SAGG-RIAC uses competence progress to sample goals (Oudeyer et al., 2007; Baranes & Oudeyer, 2013). Automated task selection, teacher–student curricula, and ALP-GMM adapt experience using measured student progress (Graves et al., 2017; Matiisen et al., 2017; Portelas et al., 2019). Learning to Teach optimizes a teacher using student feedback (Fan et al., 2018). Our feedback measures the change in semantic representation geometry induced by a complete curriculum.

Goal generation and environment design. Reverse curricula expand initial states outward from goals, while automatic goal generation adapts the goal distribution (Florensa et al., 2017; 2018). Asymmetric self-play and AMIGo train task proposers alongside learners (Sukhbaatar et al., 2018; Campero et al., 2021). POET and Enhanced POET coevolve environments and solutions with transfer between tasks (Wang et al., 2019; 2020). PAIRED uses a protagonist–antagonist performance gap to generate challenging environments (Dennis et al., 2020); prioritized level replay selects encountered environments with high learning potential (Jiang et al., 2021). Replay-guided environment design combines these operations, and ACCEL edits promising environments to extend the curriculum (Jiang et al., 2022; Parker-Holder et al., 2023). These approaches establish adaptive experience generation as a general learning mechanism. Our setting uses executable language-model curricula and compares their induced representation progress from a shared checkpoint.

Self-generated training and recursive teaching. Self-Instruct creates instruction data, while STaR, ReST, and ReST-EM iteratively train on generated outputs selected using task feedback (Wang et al., 2023; Zelikman et al., 2022; Gulcehre et al., 2023; Singh et al., 2024). SPIN uses earlier model outputs in self-play fine-tuning, and Quiet-STaR learns rationales that improve subsequent token prediction (Chen et al., 2024; Zelikman et al., 2024). Absolute Zero validates proposed tasks and solutions through execution; R-Zero coevolves a proposer and solver near the solver's capability boundary (Zhao et al., 2025; Huang et al., 2026). SOAR directly rewards a teacher for student improvement on difficult target problems (Sundaram et al., 2026). We adopt this teacher–student

formulation and study cohesion progress as curriculum feedback, with separate evaluations of the continuing student and curricula transferred to fresh students.

Feedback and optimization for language-model learning. Verifiers select mathematical solutions by predicted correctness; process supervision also evaluates intermediate reasoning steps (Cobbe et al., 2021; Uesato et al., 2022; Lightman et al., 2023). ReST-MCTS* couples process rewards with tree search for self-training (Zhang et al., 2024). Self-Rewarding Language Models train policies with self-generated preferences through direct preference optimization (Yuan et al., 2025; Rafailov et al., 2024); Self-Taught Evaluators train judges from synthetic responses and judgments (Wang et al., 2024b). TTRL derives test-time training rewards from answer consensus, while Self-Refine revises outputs through inference-time self-feedback (Zuo et al., 2026; Madaan et al., 2023). PPO, RLOO, and group-relative optimization provide different policy-update procedures (Schulman et al., 2017; Ahmadian et al., 2024; Shao et al., 2024). DeepSeek-R1 and Kimi k1.5 demonstrate reasoning gains from scaled reinforcement learning (Guo et al., 2025; Team et al., 2025). Our student uses verifiable task rewards and RLOO; cohesion progress supplies the outer teacher's curriculum-level feedback.

Hidden progress and delayed generalization. Grokking studies generalization that develops well after fitting the training data (Power et al., 2022). Representation-learning theories, mechanistic progress measures, and Omnigrok analyze aspects of these learning dynamics (Liu et al., 2022; Nanda et al., 2023; Liu et al., 2023). Hidden progress can also occur before visible accuracy gains in parity learning (Barak et al., 2022). These findings motivate measuring internal change during apparent behavioral plateaus. In language-model reinforcement learning, RL Grokking Recipe examines staged training and algorithm transfer using DELTA (Sun et al., 2025). Large-sample pass@k analyses further motivate distinguishing sampling efficiency from solution coverage (Yue et al., 2025). Our controlled experiments test the predictive value of cohesion progress, and our system evaluation reports both greedy performance and sampled solution coverage.

Representation geometry and semantic information. Representational similarity analysis, SVCCA, and centered kernel alignment characterize relationships between representation spaces (Kriegeskorte et al., 2008; Raghu et al., 2017; Kornblith et al., 2019). Neural collapse describes class-conditioned geometry late in supervised training, and the Platonic Representation Hypothesis studies alignment across learned models (Papyan et al., 2020; Huh et al., 2024). Linear and structural probes assess information accessible from hidden states; control tasks help distinguish encoded information from probe capacity (Alain & Bengio, 2017; Hewitt & Manning, 2019; Hewitt & Liang, 2019). Our score compares within-state and between-state cosine similarity for verifiably equivalent intermediate computations. Fixed semantic labels, paired measurements, and shuffled controls define its interpretation in the present curriculum-learning setting.

Functional analysis of intermediate computation. Feed-forward memory analyses, induction-head studies, and superposition models examine how transformer components represent and reuse information (Geva et al., 2021; Olsson et al., 2022; Elhage et al., 2022). Causal tracing and circuit analysis connect internal activations to specific behaviors (Meng et al., 2023; Wang et al., 2022). Causal abstraction and distributed alignment test correspondence between interpretable variables and neural representations through interventions (Geiger et al., 2021; 2024); activation patching results depend on the intervention and response metric (Zhang & Nanda, 2024). Our closest diagnostic precedent is Ye et al. (2025), who connect implicit multi-hop reasoning to cross-context semantic patching and cosine-based geometry. We use this diagnostic basis to measure curriculum-induced change, predict subsequent learning, and optimize future curriculum generation. MATH and HARP extend evaluation to mathematical reasoning (Hendrycks et al., 2021; Yue et al., 2024).

B Diagnostic Semantics and Representation Measurement

B.1 Reference Computations and Semantic Labels

A diagnostic item follows $s = F_\tau(u)$ and $y = G_\tau(s, v)$ (Equation (1)). Type τ includes the operation signature and fixed parameters, such as a modulus or transition system. Each record stores rendered input x, subcomputation input u, type τ, canonical label ℓ, reference output y, and anchor rule

a. Representation measurement presents only the rendered problem; the behavioral diagnostic in Equation (11) asks explicitly for the intermediate entity. A context is an input realization, and a continuation applies G_τ to its intermediate state.

Canonicalization preserves the information required by every admissible continuation. Entity, rational, polynomial, and finite-state labels use identities, reduced fractions, exact coefficient vectors, and residual behavior equivalence, respectively. These rules fix state equivalence before representation extraction. Each reference specification fixes the operation interface or transition system; its inputs vary to produce different states. Patching tests the correspondence between reference states and hidden representations.

Positive contexts vary the preceding computation while preserving its canonical result. Varying continuations also changes the final task. Negative pairs share a computation type but have different canonical states. Prefix length, rendering template, and continuation frequencies are balanced across positive and negative pairs.

B.2 Pair Sampling, Readout, and Aggregation

Within each type, positive sampling selects a state uniformly and two distinct contexts for it. Negative sampling selects two distinct states uniformly and one context from each. Both give equal weight to states. With type-specific mean similarities C_τ^+ and C_τ^-, the aggregate score is

$$\Phi(\theta;\mathcal{D}) = \sum_{\tau\in\mathcal{T}} w_\tau \big[C_\tau^+(\theta;\mathcal{D}) - C_\tau^-(\theta;\mathcal{D})\big], \quad w_\tau \geq 0, \quad \sum_\tau w_\tau = 1. \tag{10}$$

The expectations in Equations (4)–(5) include these type weights. We assign equal mass to the broad diagnostic families and equal mass to reference specifications within a family. Weights and pairing rules are fixed before teacher optimization.

The diagnostic renderer places the input to F_τ before a fixed boundary and places the continuation after it. The anchor is the final subtoken of the subcomputation input, before the boundary. The reference intermediate state is withheld from the rendered text. We extract the residual stream after a decoder block and normalize using $\epsilon = 10^{-8}$.

For each backbone and diagnostic domain, separate calibration trajectories select the layer. At each candidate layer, a source activation replaces a recipient activation with a different semantic state. We select the layer maximizing the mean probability increase of $G_\tau(s_{\text{source}}, v_{\text{recipient}})$ over the unpatched recipient, averaged across state-balanced examples and calibration checkpoints. Ties select the earlier layer. The layer and anchor remain fixed during recursive training. Calibration data and checkpoints are disjoint from reward measurement and final evaluation, and calibration compute is charged to training.

Each system-training domain uses a reward-diagnostic pool of 4,096 examples and a separate monitoring pool of 1,024 examples. Each round samples a balanced batch of 256 reward examples, reused for the starting student and all candidate endpoints. Dropout is disabled for these measurements. The training reward uses the paired before–after difference on the round's reward batch.

B.3 Domain-Specific Diagnostics

Two-hop relations. The intermediate state is the entity $r_1(e)$. Identity of this entity defines the canonical label. Positive pairs use different (e, r_1) prefixes reaching the same entity, and negative pairs reach different entities. The continuation applies a second relation. The reference graph fixes answers, and complete query tuples are separated across training, calibration, reward, and evaluation.

Manufactoria. The diagnostic uses HAS and REGEX semantics from DELTA (Sun et al., 2025). Each specification is compiled into a deterministic finite-state acceptor using the verifier's alphabet and transition rules. A prefix u reaches state $s = \delta^*(s_0, u)$, and the continuation tests whether suffix v leads to an accepting state. Canonical labels contain the specification and minimized-acceptor state, so equal labels preserve acceptance under every suffix.

Inputs ask whether the prefix followed by the suffix is accepted, with the prefix before the anchor. Positive groups contain distinct prefixes reaching the same state. Suffixes include accepting and

rejecting continuations when both exist; absorbing states follow the verifier's fixed outcome. Negative states admit at least one distinguishing suffix. These execution questions diagnose pattern-processing representations in the program-synthesis student.

For each of the two diagnostic families, the reward pool contains 128 reference specifications, four reachable states per specification, and four distinct prefix contexts per state. Specifications have at least four reachable states, each admitting the required number of prefixes. Reference states are selected by balanced enumeration of valid prefixes. A round samples eight specifications per family and all 16 examples for each specification, giving 256 examples. The monitoring pool uses 32 additional specifications per family with the same state and context counts.

Mathematical reasoning. The mathematical diagnostic uses four exact computation families: rational arithmetic, univariate polynomial manipulation, modular arithmetic, and linear systems. Each reference problem contains a designated subexpression or system transformation whose output is used in a continuation. The diagnostic renderer marks the input boundary without printing that output. MATH and HARP evaluation problems retain their original statements; these generated diagnostics form a separate measurement set.

For rational arithmetic, the label is the reduced fraction (p, q) with $q > 0$. Contexts use different exact expressions with the same value; continuations apply admissible rational operations. For polynomial computation, the label is the coefficient vector of the expanded polynomial under the fixed variable z. Expansion, factorization, and reassociation provide distinct contexts; continuations evaluate or combine the resulting polynomial. For modular arithmetic, diagnostic moduli are primes from 5 to 31, providing at least four distinct states. The label is (m, r) with $0 \leq r < m$, and subsequent operations act in the same residue ring. For linear systems, the label is the reduced row-echelon augmented matrix under a fixed variable order. Contexts apply invertible row operations to equivalent systems, and continuations query a variable or a linear expression on the solution set. Questions requesting a unique value are generated with sufficient constraints to determine it.

The reward pool contains 64 reference specifications per family, four semantic states per specification, and four distinct contexts per state. This gives 4,096 examples. Each round samples four specifications from each family, giving 256 examples. The monitoring pool uses 16 separate specifications per family. Specification identities, input expressions, and rendered questions are separated across construction, calibration, reward, and monitoring pools. Within a fixed type, the transformations preserve exact semantics; variable names, operation placement, and surface renderings vary across contexts.

B.4 Prediction of Subsequent Learning

The controlled-task block of Table 1 predicts target accuracy after a common target-training continuation from curriculum-endpoint features. The three conditions use behavioral features alone, with shuffled $\Delta\Phi$, or with correctly paired $\Delta\Phi$.

Data generation and training budgets. Each of 200 independent runs supplies its step-400 checkpoint, selected before observing subsequent outcomes. A fixed generation policy samples ten groups of four curricula per run, with no group filtering. Each curriculum contains 100 questions and trains for 150 optimizer steps in each of five trials. Candidates within a group share initial parameters and optimizer state, with paired training seeds.

Every trial endpoint receives 400 target-training steps. At this transition, model parameters are retained and the optimizer is reset. Paired seeds fix the target distribution and question order across candidates. Both stages use RLOO with AdamW and learning rate 2×10^{-6} under Appendix D.5's student settings. Each step uses eight prompts and four completions per prompt (32 trajectories), with gradient accumulation of 1.

Target evaluation. Current and post-continuation accuracy use the fixed 1,000-query two-hop test set $\mathcal{Q}_{\text{test}}^{\text{2hop}}$. The student generates a final entity without an explicit reasoning trace, using `do_sample=False`, `num_beams=1`, and `max_new_tokens=32`. Both measurements use the same parsing and correctness rules. Complete test queries are disjoint from training (Section 2.1).

Predictor and fitting. We use ridge regression (Hoerl & Kennard, 1970) with an intercept to predict post-continuation target accuracy in percent. The behavioral feature vector contains three endpoint measurements: current target accuracy, curriculum-training accuracy, and intermediate-state prediction accuracy, defined below. The representation condition appends the scalar $\Delta\Phi = \Phi_{\text{after}} - \Phi_{\text{before}}$; the shuffled condition appends its permuted counterpart. All features are measured before target-training continuation.

Features are standardized using the means and standard deviations of the fitting partition. We select the ridge penalty from $\{10^{-3}, 10^{-2}, 10^{-1}, 1, 10, 100, 1000\}$ by four-fold cross-validation grouped by run, minimizing validation mean absolute error. Standardization is fitted separately within each fold using only that fold's fitting observations. The selected configuration is then fitted on the complete training partition, with its standardization parameters applied to the held-out test observations. Predictions are evaluated without clipping. All three conditions share the data split, predictor class, penalty grid, and model-selection procedure.

Behavioral measurements. Current target accuracy and curriculum-training accuracy measure final-answer correctness on their respective query sets at the curriculum endpoint. Intermediate-state prediction accuracy asks explicitly for $s_i = r_{1,i}(e_i)$ using single-hop input $(e_i, r_{1,i})$. On fixed diagnostic set $\mathcal{D}_{\text{local}}$ with fixed decoding, we compute

$$A_{\text{local}}(\theta) = \frac{100}{|\mathcal{D}_{\text{local}}|} \sum_{i \in \mathcal{D}_{\text{local}}} \mathbf{1}[\text{can}(\hat{s}_i) = \text{can}(s_i)], \tag{11}$$

where $\hat{s}_i$ is the generated entity and can applies the task's parsing and canonicalization rules. The reference entities also supply the semantic labels for cohesion measurement.

Repeated trials and observation unit. One observation represents a candidate curriculum averaged over five trials. Each trial supplies the three behavioral features and $\Delta\Phi$ before target continuation, plus accuracy after continuation. Each feature and the target are averaged separately across those trials.

Sample counts and partitioning. The 200 runs contain 2,000 groups, 8,000 curriculum observations, and 40,000 training trials. Within a run, all ten groups share the step-400 checkpoint and differ in sampled curricula.

Random seed 2027 partitions complete runs into training (160 runs, 1,600 groups, 6,400 observations) and test (40 runs, 400 groups, 1,600 observations). Cross-validation uses only the training partition. All groups from a run stay together, and curriculum content is disjoint between training and test.

Additional controlled-task branches. An independent set of experiments uses 17 randomly selected branches. Each branch summarizes five training trials. Cohesion progress, current target accuracy, and post-continuation target accuracy are averaged separately over those same five trials. Branches 8 and 9 share their starting checkpoint and curriculum-training and target-continuation budgets.

Shuffled representation control. We permute the four mean $\Delta\Phi$ values within each group, independently in training and test, while holding behavioral features and targets fixed. This preserves cohesion-progress values and changes their assignment to curricula. With seed 2027, we repeat shuffling and fitting 20 times using the same model-selection procedure. Table 1 reports mean test MAE and ranking accuracy across repetitions.

Prediction error. Let y_j be the observed post-continuation target accuracy in percent and $\hat{y}_j$ its prediction for held-out branch j. For N_{test} held-out observations, mean absolute error is

$$\text{MAE} = \frac{1}{N_{\text{test}}} \sum_{j=1}^{N_{\text{test}}} |\hat{y}_j - y_j|, \tag{12}$$

reported in percentage points.

Ranking accuracy. We compare pairs of candidate curricula within each held-out curriculum group. Let g_i denote the group of observation i and define

$$\mathcal{P} = \{(i, j) : i < j,\ g_i = g_j,\ y_i \neq y_j\}. \tag{13}$$

The observed accuracies y_i and y_j are the five-trial averages. Pairs with equal observed accuracies are excluded. A correctly predicted ordering receives score 1, an incorrect ordering receives score 0, and equal predictions receive score $1/2$. Accordingly,

$$\mathrm{RankAcc} = \frac{100}{|\mathcal{P}|} \sum_{(i,j)\in\mathcal{P}} \left\{ \mathbf{1}[(\hat{y}_i - \hat{y}_j)(y_i - y_j) > 0] + \frac{1}{2}\mathbf{1}[\hat{y}_i = \hat{y}_j] \right\}. \tag{14}$$

Tie checks use strict equality on the unrounded values. Scores are pooled over all eligible pairs across the test groups. The 400 test groups provide $400\binom{4}{2} = 2{,}400$ candidate pairs; excluding 208 pairs with equal observed outcomes leaves $|\mathcal{P}| = 2{,}192$. All three predictor conditions use this same comparison set. For the shuffled control, ranking accuracy is computed separately for each repetition and then averaged across the 20 repetitions.

MATH prediction and curriculum selection. The MATH block of Table 1 predicts relative learning gains within curriculum groups, holding out complete runs. The predictor uses current target, curriculum-training, and intermediate-state accuracy, together with changes in reference-answer and intermediate-state log likelihood on the cohesion diagnostic items. The representation condition adds $\Delta\Phi$; the label-permutation control computes cohesion progress after permuting semantic labels.

Figure 3 examines this signal in three ways. The left panel reports its residual association with subsequent learning after adjustment for behavioral features using predictors fitted on separate runs. The center compares continuation gains when selecting from common candidate pools using cohesion progress, permuted-label cohesion, target accuracy, or answer and state likelihood. The right panel evaluates the selected curricula on fresh students.

B.5 Cross-Context Exposure Intervention

Curriculum construction and training. Full, Reduced, and Restored start independently from a shared checkpoint. Each receives the same number of examples and 500 optimizer steps with five paired seeds, followed by 400 target-training steps. The construction below generates the conditions in Figure 4. Target accuracy is measured after continuation; patching success follows Equation (18).

Joint exposure and preserved quantities. Let $N^{(c)}_{s,i,j}$ count presentations of training query $(p_{s,i}, v_j)$ in condition c, where prefix $p_{s,i} = (e, r_1)$ reaches state $s = r_1(e)$ and continuation $v_j = r_2$ produces answer $v_j(s)$. Repeated presentations are counted separately. The intervention changes these joint counts while preserving prefix, continuation, and state marginals across conditions. Atomic coverage is the set of single-hop facts appearing in the training queries. Restoring connection coverage means that removed combinations become available again; their multiplicities can remain different from Full.

Experimental construction. We use six intermediate states, eight distinct training prefixes per state, and eight continuations defined on every state. We select these from the training partition in canonical identifier order, excluding complete queries reserved for diagnostics or evaluation. All 64 combinations per state are valid under the fixed relation table. The answer to each query is supplied by that table. The renderer and answer format remain fixed across conditions.

For each state, we partition the eight prefixes into two sets of four and pair $p_{s,a}$ with $p_{s,a+4}$ for $a \in \{1, \ldots, 4\}$. Likewise, we pair v_b with v_{b+4} for $b \in \{1, \ldots, 4\}$. Before making these partitions, we permute each state's prefix list and the shared continuation list using a NumPy `Generator(PCG64(2027))`, processing states in canonical order and the continuation list last. This fixes the assignments before training. Each tuple (s, a, b) defines one 2×2 block, with the two prefixes as rows and the two continuations as columns. Its presentation counts are

$$B^{\mathrm{Full}} = \begin{pmatrix} 2 & 2 \\ 2 & 2 \end{pmatrix}, \qquad B^{\mathrm{Reduced}} = \begin{pmatrix} 4 & 0 \\ 0 & 4 \end{pmatrix}, \qquad B^{\mathrm{Restored}} = \begin{pmatrix} 3 & 1 \\ 1 & 3 \end{pmatrix}. \tag{15}$$

Each block contains eight presentations, with four per prefix and four per continuation. Reduced shifts both off-diagonal counts to the diagonal. Restored returns one count to each off-diagonal cell, recovering all connections at half of Full's off-diagonal exposure.

Finite schedule and exact budget. There are $6 \times 4 \times 4 = 96$ blocks. We enumerate their (s, a, b) tuples lexicographically and permute this list once using a separate `Generator(PCG64(2027))`. We repeat that permutation five times and append its first 20 blocks, giving exactly 500 optimizer steps. One step uses the eight presentations from its scheduled block as the global prompt batch, with four sampled completions per prompt and gradient accumulation of one. The repeated queries retain their fixed rendering. For each paired training seed in $\{0, 1, 2, 3, 4\}$, we apply the same within-batch permutation to the eight row-major presentation slots in all conditions. The block schedule and cell multiplicities are fixed across seeds. All conditions use the same student optimizer and decoding settings described in the experimental configuration.

Consequently, each condition receives 4,000 prompt presentations and 16,000 sampled answer trajectories during the 500-step intervention. Full and Restored each cover 384 distinct two-hop queries; Reduced covers 192. Every prefix and every state–continuation atomic fact appears in all three conditions.

Let $m_{s,a,b}$ be the number of times a block appears in the schedule: it equals six for the first 20 blocks of the permutation and five for the remaining 76. The corresponding 2×2 submatrix of the complete joint count matrix is exactly $m_{s,a,b} B^{(c)}$. This implies

$$\sum_j N^{(c)}_{s,i,j} = \sum_j N^{(\mathrm{Full})}_{s,i,j}, \qquad \sum_i N^{(c)}_{s,i,j} = \sum_i N^{(\mathrm{Full})}_{s,i,j}, \tag{16}$$

for every state, prefix, continuation, and condition. These identities also preserve overall prefix, continuation, and state frequencies in the realized schedule. The comparison therefore tests joint exposure under fixed marginals and budget; Restored recovers Full's connection coverage with different multiplicities.

State-resolved cohesion. For the controlled two-hop computation, condition the positive and negative pair distributions in Equations (4) and (5) on the first member having label s:

$$C^{\pm}_s = \mathbb{E}_{(i,j)\sim\mathcal{P}^{\pm}}[z_\theta(q_i)^\top z_\theta(q_j) \mid \ell_i = s], \qquad \Phi_s = C^{+}_s - C^{-}_s. \tag{17}$$

The state-balanced pairing uses the same state weights w_s in the positive and negative expectations. Hence the global score is $\Phi = \sum_s w_s \Phi_s$, with $\sum_s w_s = 1$.

Absolute patching success. Let $\mathcal{I}_s$ be the valid source–recipient test pairs whose source represents intermediate state s. For pair i, let v_i be the recipient's continuation and $\hat{a}^{\mathrm{patch}}_i$ its greedily decoded answer after activation replacement. Patching success is

$$\mathrm{PatchSuccess}_s = \frac{100}{|\mathcal{I}_s|} \sum_{i \in \mathcal{I}_s} \mathbf{1}\Big[\mathrm{can}(\hat{a}^{\mathrm{patch}}_i) = \mathrm{can}(G_\tau(s, v_i))\Big]. \tag{18}$$

Here can is the task's answer parsing and entity canonicalization rule. Equation (18) expresses absolute counterfactual accuracy after patching as a percentage. Figure 4 displays $\mathrm{PatchSuccess}_s\,/100$ on a proportion scale. The 11.2 percentage-point effect in Section 2.2 is the patched-minus-unpatched counterfactual-answer probability change. Absolute greedy patching success and this probability change are separate metrics.

B.6 Statistical Reporting for the Mechanism Figures

Figures 2 and 4 report target accuracy over five training runs. At each evaluation checkpoint t, we first compute accuracy over the evaluation examples separately for each run. Let $a_r(t)$ be the accuracy of run r. The mean and sample standard deviation (SD) across runs are

$$\bar{a}(t) = \frac{1}{5}\sum_{r=1}^{5} a_r(t), \qquad s_a(t) = \sqrt{\frac{1}{4}\sum_{r=1}^{5}\big(a_r(t) - \bar{a}(t)\big)^2}.$$

The curves show $\bar{a}(t)$, and the shaded bands span $\bar{a}(t) \pm s_a(t)$. For Figure 4, this aggregation is performed separately for Full, Reduced, and Restored, with the five training seeds paired across conditions.

The lower-panel contours in Figure 2 enclose 50% and 90% of the projected representation mass for one semantic state; they describe the spatial distribution of representations.

C Teacher Outputs and Curriculum Construction

C.1 Specification Format and Teacher Instructions

The teacher returns a JSON object with 32 ordered items. Each contains `family` (constructor), `parameters` (values), `composition` (operation dependencies), and `presentation` (semantics-preserving rendering). The prompt specifies allowed keys and types. Array order determines training order.

The fixed teacher instruction is:

> Generate an ordered curriculum of 32 training problems using the supplied constructors. Return one JSON object with an items array. For every item, provide its family, parameters, composition, and presentation fields using the listed schemas. Choose the problem content, parameter values, composition, and order to improve the student's learning. The task interface constructs each problem and verifies student answers. Your curriculum receives feedback from the student's progress after training on the complete curriculum.

The prompt appends the domain schema and one valid example per constructor, fixed across rounds. Semantic labels, diagnostic instances, and reference hidden vectors remain internal to the evaluator. Curriculum rewards update the teacher policy.

C.2 Admissible Construction Spaces

Manufactoria curricula. The allowed families are START, APPEND, EXACT, REGEX, COMPR, and HAS. Parameters follow the corresponding DELTA generator's accepted schema. String patterns, numeric thresholds, and regular-expression parameters are checked by that generator. Composition is represented by the ordered curriculum; each item invokes one family constructor. The builder renders the synthesis problem and supplies its executable verifier. Student training uses binary full-pass reward. The diagnostic's acceptor traces and labels are generated independently of the proposed program-synthesis examples.

Mathematical curricula. The four mathematical constructors share the computation families used by the diagnostic. A composition is a typed, acyclic expression graph with one to four operation nodes. The builder checks operand and output types, executes the graph with exact arithmetic, and renders its requested output as a problem. The teacher chooses graph structure and values within the following domains: integer constants from -20 to 20; positive rational denominators from 1 to 20; polynomial inputs of degree at most four with integer coefficients from -9 to 9; prime moduli from 2 to 31; and linear systems with two or three variables and integer coefficients from -9 to 9. Division requires a nonzero divisor. Polynomial operations respect a degree cap of eight. Linear-system constructors provide full-rank systems when a unique solution is requested. The presentation field selects a direct symbolic or a fixed verbal rendering of the same computation.

The reference answer is computed from the executable specification. The student receives the rendered problem and produces its answer. Mathematical verification compares exact numbers, expressions, or ordered solution tuples under the constructor's output type. MATH and HARP target sets separately evaluate the reasoning abilities acquired through these generated curricula.

Operation schemas. Each graph node contains an identifier, an operation name, and an ordered argument list. An argument refers to a named parameter or an earlier node; the last node supplies the requested output. Rational operations are addition, subtraction, multiplication, and division. Polynomial operations are addition, subtraction, multiplication, differentiation, and evaluation at a rational argument. Modular operations are addition, subtraction, multiplication, and nonnegative

integer exponentiation, with exponents at most eight. Linear-system operations are row exchange, addition of a row multiple, nonzero row scaling, solution, and evaluation of a linear form. The builder propagates exact types and checks the stated size bounds after every operation.

A complete construction example. The following object illustrates one item in the teacher's 32-item list. Rational parameters are encoded as numerator–denominator pairs.

```
{
  "family": "rational",
  "parameters": {"a": [1,2], "b": [1,3], "c": [6,1]},
  "composition": [
    {"id": "n0", "op": "add", "args": ["a", "b"]},
    {"id": "n1", "op": "mul", "args": ["n0", "c"]}
  ],
  "presentation": "symbolic"
}
```

The builder renders $(1/2 + 1/3) \times 6$ and verifies the exact answer 5. A separate diagnostic can use the first sum as F_τ, yielding state $5/6$, and multiplication as the continuation. The context $7/6 - 1/3$ yields the same state; $1/2 + 1/4$ yields negative state $3/4$. The diagnostic places the expression before the anchor and continuation after it, storing $5/6$ as the label. Training and diagnostics thus use the same reference semantics on separate examples.

C.3 Construction Failures and Training Execution

Grammar-constrained decoding enforces the JSON schema and field values. The builder checks constraints, graph types, and execution; a valid candidate contains exactly 32 constructible items. Invalid candidates receive no student trial and a teacher-update reward of -5 after normalization. They remain in the update group but cannot be selected for continuation. If all are invalid, the teacher update is skipped and the student retained (Appendix D.5). All comparison conditions use these rules, and proposal and checking costs are counted.

For each valid candidate, the ordered items are traversed twice in eight minibatches of eight prompts. Four student completions are sampled per prompt. The full item list, verifier outcomes, and candidate reward remain associated with the generated specification. The teacher log-probability covers its complete sampled token sequence under the constrained generation policy.

D Experimental Details

D.1 Experiment Scope and Terminology

Runs, rounds, and repeated trials. A system run is a complete execution from the initial teacher and student under one root seed. An outer round generates curricula, trains candidate branches, measures rewards, updates the teacher, and selects a student to continue. A curriculum group contains candidates compared from the same starting student. A training replicate is one candidate trial under one seed; a student step is one optimizer update. A checkpoint records model parameters, and continuation also restores optimizer state.

The prediction analysis uses 200 runs, ten groups per run, and five trials per candidate (Appendix B.4). System comparisons use five runs; our method uses 100 outer rounds and two trials per candidate (Appendix D.5). Repeated-student evaluation follows Appendix D.9.

Curricula, students, and evaluation sets. A curriculum is an ordered list of specifications. A constructor renders each specification and supplies a verifier for answer correctness (Appendix C). The continuing student is the selected branch carried across rounds. A fresh student starts from the common initial checkpoint to evaluate a fixed teacher's curricula; its resulting performance measures *curriculum effectiveness*.

A task family is a class of tasks within a benchmark. The *full test set* is its designated held-out partition (Appendix D.3); fail@128 retains problems unsolved during 128 initial screening attempts.

Pass@k measures sampled solution coverage, and greedy accuracy uses one response formed by choosing the most probable token at each step. Manufactoria full-pass requires a program to pass every test case for its problem.

Diagnostic data and feedback. Calibration data select the readout layer, reward-diagnostic data measure cohesion progress, and a separate monitoring pool tracks representations (Appendix B.2). Reference computations supply the semantic labels for C^+, C^-, and Φ. *Cohesion progress* is $\Delta\Phi$; *representation feedback* is its use as a curriculum-level reward or selection criterion. *Outcome feedback* uses target-accuracy gain ΔS. Appendix D.7 defines total training compute and cost-to-target.

D.2 Models and Initialization

Teacher and student maintain separate parameters and optimizer states and start from the same instruction-tuned checkpoint within a run. The main checkpoint is `google/gemma-4-E4B-it`, with 4.5B effective parameters, 8B parameters including embeddings, and 42 decoder layers. The within-family checkpoint is `google/gemma-4-E2B-it`, with 2.3B effective parameters, 5.1B including embeddings, and 35 decoder layers. The cross-family checkpoint is `Qwen/Qwen3-4B-Instruct-2507`, with 4.0B parameters and 36 decoder layers. The selected model revisions are:

- Gemma 4 E4B-IT: `ee0ef6023621cff504d758262d4e04895a5af4a2`.
- Gemma 4 E2B-IT: `3e22461f65e89153144f8adb70e3b8c2cc9845a7`.
- Qwen3-4B-Instruct-2507: `cdbee75f17c01a7cc42f958dc650907174af0554`.

Weights, tokenizer or processor, and chat templates use the same revision. The official model cards are available at `https://huggingface.co/google/gemma-4-E4B-it`, `https://huggingface.co/google/gemma-4-E2B-it`, and `https://huggingface.co/Qwen/Qwen3-4B-Instruct-2507`.

All inputs are text. Gemma chat templates set `enable_thinking=False`; the selected Qwen checkpoint supports non-thinking generation. Language-model parameters are optimized in the 16-bit brain floating-point format (BF16), with multimodal encoders frozen. Each benchmark is trained in a separate run initialized from the named checkpoint. The controlled study in Section 2 retains its initialization after atomic preparation.

D.3 Benchmark Versions and Data Partitions

DELTA–Manufactoria. We use the released Manufactoria tasks of Sun et al. (2025). The source repository is `https://github.com/sunblaze-ucb/rl-grok-recipe`. The evaluated families are START, APPEND, EXACT, REGEX, COMPR, and HAS. The released training and test assignments are retained, and HAS supplies target-training and target-reward queries. The held-out HAS partition contains 103 problems. Each held-out instance includes a specification and a test suite. A solution is accepted only when the generated program passes the entire suite under the task's executor and resource limits.

MATH. We use the original 2021 MATH release (Hendrycks et al., 2021), containing 7,500 training and 5,000 test problems. Dataset identity follows the official repository, `https://github.com/hendrycks/math`, and its released file lists. The seven subject labels are Algebra, Counting & Probability, Geometry, Intermediate Algebra, Number Theory, Prealgebra, and Precalculus; native difficulty labels range from Level 1 to Level 5. The heatmap's MATH column evaluates all 5,000 official test problems.

We sort the training instances by their released relative file paths and shuffle with Python's `random.Random(2026)`. The first 6,750 problems form the target-training pool, and the remaining 750 form the development pool. The official test partition remains intact. Prompt text comes from the released problem field; reference answers are extracted from the released solutions.

HARP. We use HARP's default short-answer release (Yue et al., 2024). The source file is `HARP.jsonl.zip`, available from `https://github.com/aadityasingh/HARP`, at

revision `dac2734ff6443bcaf3bbdcb10f13cf21ae9729c2`. The archive contains 4,780 problems with automatically checkable answers. Its SHA-256 is `40bcbda27876ada4d5de3c26b3f6e4a67d932874f645d4ae7c1fc0aed0e09392`. The source identifier is the tuple `(year, contest, number)`. We retain the released subject and six-level difficulty annotations. The student receives the short-answer problem field, and the verifier uses the answer field. Human-written solutions remain evaluation metadata.

HARP's default release is an evaluation collection. For this study, we define an identity-level training, development, and test partition. We sort problems by numerical year, contest string, and numerical problem number, then shuffle with `random.Random(2026)`. The first 3,824 identities form the training/development pool and the remaining 956 form the test set. In the shuffled training/development pool, the first 3,442 identities form the target-training pool and the remaining 382 form the development pool. These assignments are shared across all backbones and methods. The heatmap's HARP column evaluates all 956 problems in the full test set. The full 4,780-problem collection is the source pool for this partition.

Role of each partition. Data assignment precedes model-dependent difficulty screening. Target-training problems support Direct target RL and target-grounded feedback; development problems support implementation checks with the fixed protocol. Final results use test problems. Training runs for MATH and HARP are separate. Generated curricula, diagnostic reward pools, and monitoring examples use their own constructor seeds and example identities. The diagnostic labels are obtained from executable reference computations as specified in Appendix B.3. The mathematical benchmark's official answers serve the task evaluator.

D.4 Construction of the fail@128 Target Subsets

Definition and screening model. For a benchmark partition $\mathcal{P}$ and initial checkpoint θ_0, we independently sample 128 responses to every problem $x \in \mathcal{P}$. Let $V(x, y) \in \{0, 1\}$ denote the fixed task verifier. We define

$$\mathcal{H}_{\theta_0}(\mathcal{P}) = \left\{ x \in \mathcal{P} : \sum_{j=1}^{128} V(x, y_j) = 0, \quad y_j \sim \pi^S_{\theta_0}(\cdot \mid x) \right\}. \tag{19}$$

This operational definition follows the initial-failure selection principle in SOAR (Sundaram et al., 2026). The screening checkpoint is the instruction-tuned initial student specified in Appendix D.2. Each backbone therefore has its own fixed hard-subset manifest. Within a backbone, every method and all five training seeds use the same manifest.

Sampling configuration. Screening uses temperature 1.0, top-$p = 1.0$, disabled top-k filtering, and a maximum of 4,096 generated tokens per response. The input limit is 16,384 tokens. Chat formatting and thinking mode match final evaluation. For mathematics, the common system instruction is *Solve the problem and put the final answer in a single boxed expression.* The user message contains the original problem statement. All 128 samples are drawn, including after an earlier sample succeeds. Sampling streams are determined from screening seed 2026, the backbone identifier, problem identifier, and sample index using SHA-256-derived integer seeds. The manifest stores each problem identifier, its 128 verifier outcomes, the checkpoint revision, and decoding settings.

Training and held-out subsets. Equation (19) is applied separately to the target-training, development, and test pools. Direct target RL samples from $\mathcal{H}_{\theta_0}(\mathcal{P}_{\text{train}})$, and target-grounded feedback draws its reward questions from that same subset. The teacher's constructor space remains the one defined in Appendix C. The mathematical test subsets are $\mathcal{H}_{\theta_0}(\mathcal{P}_{\text{test}})$. Their membership is fixed before curriculum training.

The full-test heatmap and hard-subset curves evaluate the same final checkpoint for each run. The latter use fixed subsets of the complete test problems, with a separate sampling budget and a relative performance metric. Table 6 reports the hard-subset counts obtained from screening each initial backbone. A zero-sized subset is reported as empty, and its score is undefined. Invalid or missing model answers count as unsuccessful attempts; input-limit and verifier exceptions are logged separately, and examples with verifier exceptions are checked against the released references before screening is finalized. The full test sets retain their original problem identities.

Backbone	Benchmark	Hard train	Hard dev.	Hard test
Gemma 4 E4B-IT	MATH	1,215	135	900
Gemma 4 E4B-IT	HARP	1,446	160	402
Gemma 4 E2B-IT	MATH	1,890	210	1,400
Gemma 4 E2B-IT	HARP	1,824	202	507
Qwen3-4B-Instruct-2507	MATH	1,485	165	1,100
Qwen3-4B-Instruct-2507	HARP	1,583	176	440

Table 6: Sizes of the model-dependent fail@128 subsets obtained by screening each initial backbone. All methods share the same subsets within each backbone.

Independent evaluation. The full-test heatmap draws 32 new responses per problem. The hard-subset curves draw a separate set of 128 new responses per problem to support evaluation through $k = 128$. Both use sampling streams disjoint from screening. The initial-checkpoint heatmap row is also evaluated with fresh samples. Fresh samples can solve a problem that failed all screening attempts. Fail@128 identifies observed initial failure under the specified sampling budget. Its membership remains fixed as the student and its success probabilities change during training.

Verification. MATH uses reference answers extracted from the official solution field; HARP uses the released short-answer field. We use HARP's released answer checking code (`src/eval/latex_answer_check.py`), pinned to the dataset repository revision above, for both mathematical benchmarks, with adapters supplying each benchmark's reference. The final boxed answer is extracted, with the last boxed answer used when more than one is present. Missing final answers and unsuccessful parses are incorrect. Numeric and symbolic equivalence follow the pinned checker. The same extraction and checking procedure is used during screening, target-reward evaluation, and final evaluation. Generated curriculum examples retain their constructor-specific exact verifier. For Manufactoria, each candidate is checked independently against all test cases assigned to its problem; a successful candidate must satisfy the entire suite.

D.5 Fixed Training Configuration

Branch training and teacher objective. For valid candidate k and replicate a, the student endpoint is

$$\theta'_{t,k,a} = \mathrm{Train}(\theta_t; \mathcal{C}_{t,k}, B, \xi_a), \qquad a = 1, \ldots, R, \tag{20}$$

where B counts optimizer steps and ξ_a specifies training randomness. The teacher maximizes

$$J_t(\phi) = \mathbb{E}_{\mathcal{C} \sim \pi_\phi^T(\cdot|p)} \left[r^{\mathrm{rep}}(\mathcal{C}; \theta_t) \right], \tag{21}$$

where p fixes generation instructions and the constructor schema (Appendix C.1). The reward is defined in Equation (8); the expectation also averages student-training randomness and diagnostic sampling. The representation-guided system runs in Section 4 use 100 outer rounds, $K = 4$ candidate curricula per round, $N = 32$ ordered items per curriculum, and $R = 2$ student-training replicates per candidate. Each replicate receives $B = 8$ optimizer steps with eight prompts and four completions per prompt. The first replicate of the selected candidate is continued, including its optimizer state. Ties select the lowest candidate index. Each round uses 256 reward-diagnostic examples; outcome-based feedback uses 256 target-training reward questions.

Teacher and student use RLOO (Ahmadian et al., 2024) with AdamW (Loshchilov & Hutter, 2019). Teacher comparison groups contain candidate curricula; student groups contain answers to the same prompt. Learning rates are 10^{-6} and 2×10^{-6}, respectively. Both use $(\beta_1, \beta_2) = (0.9, 0.95)$, $\epsilon = 10^{-8}$, zero weight decay, gradient-norm clipping at 1.0, and a constant learning rate. Each rollout batch receives one student update, and each round receives one teacher update. Both use a KL coefficient of 0.01 relative to their initial policies. Constrained teacher and reference probabilities use the same token masks.

Measured rewards and optimizer inputs. The representation and positive-cohesion conditions first compute the raw changes $\Delta\Phi$ and ΔC^+. The outcome condition computes ΔS from accuracies expressed as fractions in $[0, 1]$. Each curriculum-level reward averages its repeated student trials before normalization. Let $\mathcal{V}_t$ be the valid candidates in outer round t, with $m_t = |\mathcal{V}_t|$. For $m_t \geq 1$,

define

$$\mu_t = \frac{1}{m_t} \sum_{k \in \mathcal{V}_t} r_{t,k}, \qquad \sigma_t = \sqrt{\frac{1}{m_t} \sum_{k \in \mathcal{V}_t} (r_{t,k} - \mu_t)^2}. \tag{22}$$

The normalization uses the current round's valid candidates and the population standard deviation:

$$\mathcal{N}_t(r) = \frac{r - \mu_t}{\max(\sigma_t, \epsilon)}, \qquad \epsilon = 10^{-8}. \tag{23}$$

All teacher-feedback conditions use this rule. The optimizer-input reward is

$$\widetilde{r}_{t,k} = \begin{cases} \mathcal{N}_t(r_{t,k}), & \mathcal{C}_{t,k} \text{ is valid,} \\ -5, & \mathcal{C}_{t,k} \text{ is invalid.} \end{cases} \tag{24}$$

Invalid candidates are excluded from the normalization statistics and receive their penalty after normalization. With one valid candidate, or identical rewards among valid candidates, all valid normalized rewards are zero. Invalid candidates still participate in the leave-one-out baseline. If no candidate is valid, we skip the teacher update and retain the current student. Student-branch selection uses the original rewards among valid candidates. We apply no additional teacher-advantage normalization and no reward or advantage clipping; the gradient-norm clipping specified above applies to optimizer gradients.

The RLOO advantage computed from the optimizer-input rewards is

$$\widehat{A}_{t,k} = \widetilde{r}_{t,k} - \frac{1}{K-1} \sum_{j \neq k} \widetilde{r}_{t,j}. \tag{25}$$

The reward-dependent gradient estimate is

$$\widehat{g}_t = \frac{1}{K} \sum_{k=1}^{K} \mathrm{sg}(\widehat{A}_{t,k}) \nabla_\phi \log \pi_\phi^T(\mathcal{C}_{t,k} \mid p)\big|_{\phi=\phi_t}, \tag{26}$$

where sg denotes stop-gradient. The KL term is applied separately with the coefficient above. Student RLOO advantages are computed within the four responses sampled for each prompt. Candidate replicates are executed from the same saved parameter and optimizer state; execution can be sequential while preserving the paired seeds and common starting point.

Training samples use temperature 1.0, top-$p = 1.0$, and disabled top-k filtering. Input prompts are limited to 2,048 tokens and generated outputs to 4,096 tokens. The teacher output limit applies to the complete curriculum specification. Greedy evaluations use the same student output limit. Final sampled evaluation uses temperature 1.0, with 32 independent completions per problem for the full-test heatmap and 128 for the mathematical hard-subset curves. Greedy evaluation generates one response per problem without sampling. Benchmark evaluation uses an input limit of 16,384 tokens and the same 4,096-token output limit, with identical handling across methods. Statements exceeding that input limit are counted as unsolved and their count is reported with the evaluation.

D.6 Baseline Implementations

Direct target RL. The student trains directly on the target training split using RLOO (Ahmadian et al., 2024) and the common verifier reward. This direct-training control serves the same role as the Hard-Only comparison in SOAR (Sundaram et al., 2026). It uses the matched total budget for student learning. Manufactoria targets HAS; mathematical runs use the corresponding MATH-fail@128 or HARP-fail@128 target-training subset.

Matched recursive search. This internal control measures the contribution of teacher-policy updates under the same candidate-training and continuation procedure. The teacher stays at its initial checkpoint and samples four curricula per round. Candidate trials and cohesion-based selection follow our procedure.

Uncertainty-driven RSI. This control evaluates the intermediate-difficulty objective studied in R-Zero (Huang et al., 2026) under our verifiable curriculum interface. R-Zero uses $1 - 2|\widehat{p} - 1/2|$ with majority-vote pseudo-labels; our implementation uses normalized Bernoulli variance with verifier-based correctness estimates. The teacher is updated using uncertainty in the current student's correctness on its proposed examples. For each example, four sampled student answers estimate a verifier-success frequency $\widehat{p}_i$. The curriculum score is $N^{-1} \sum_i 4\widehat{p}_i(1 - \widehat{p}_i)$. The teacher update and candidate selection use this score. Candidate students receive the same training procedure as in our method, and uncertainty-estimation rollouts contribute to total cost.

Target-grounded Teacher. We adapt SOAR's student-improvement reward (Sundaram et al., 2026) to the common candidate-training and continuation procedure. The feedback is

$$r^{\text{out}}_{t,k} = \frac{1}{R} \sum_{a=1}^{R} \left[S(\theta'_{t,k,a}; Q_t) - S(\theta_t; Q_t) \right], \tag{27}$$

where Q_t contains 256 target-training questions and S is greedy verifier accuracy in $[0, 1]$. Percentage scores multiply it by 100. We denote this trial-averaged accuracy change by ΔS and use it for teacher updates and branch selection. Mathematical questions come from the fixed fail@128 training subset, sampled with replacement when it contains fewer than 256 problems. Candidates share each round's reward batch. Architecture, constructors, budgets, and the designated replicate match our method.

Prompted Teacher. This frozen-generator control is analogous to the Base Challenger baseline in R-Zero (Huang et al., 2026). The initial teacher generates one curriculum at a time without policy updates. The student trains sequentially on every valid curriculum with the common optimizer until the matched total budget is exhausted.

Reference conditions. The initial checkpoint is evaluated before training. The human-designed Manufactoria curriculum allocates one third of its student-training budget to an equal mixture of START, APPEND, and EXACT, one third to REGEX, and one third to HAS, following the staged task sequence studied by Sun et al. (2025). The mathematical reference is our instantiation of a curriculum that progresses from simpler to more complex problems (Bengio et al., 2009). It allocates successive thirds to one-operation, two-operation, and three-to-four-operation problems, balancing constructor families within each stage. These schedules use the same generators and student optimizer as the automated curricula.

D.7 EVALUATION AND COMPUTE ACCOUNTING

For sampled evaluation, each problem receives n independent candidate solutions: $n = 32$ for the full-test heatmap and $n = 128$ for the mathematical hard-subset curves. A Manufactoria candidate is one complete program and succeeds when it passes every test case for its problem. Mathematical correctness uses the final-answer checker in Appendix D.4. For m problems, let c_i be the number of correct candidates on problem i. We use the pass@k estimator of Chen et al. (2021):

$$\widehat{\text{pass@}k} = \frac{1}{m} \sum_{i=1}^{m} \left[1 - \frac{\binom{n-c_i}{k}}{\binom{n}{k}} \right], \qquad 1 \le k \le n, \tag{28}$$

with a zero numerator when $n - c_i < k$. At $k = n$, this equals $m^{-1} \sum_i \mathbf{1}[c_i > 0]$. Values are multiplied by 100 for percentages. For the hard-subset curves, the same 128 evaluation responses supply all eight points at $k \in \{1, 4, 8, 16, 24, 32, 64, 128\}$. Their vertical axis is

$$\Delta \,\text{pass@}\,k(\mathcal{M}) = 100 \left[\widehat{\text{pass@}k}(\mathcal{M}) - \widehat{\text{pass@}k}(\text{Direct target RL}) \right], \tag{29}$$

in percentage points. The direct-target baseline is zero at every k.

Curriculum-effectiveness and recursive-learning curves use one greedy response per problem on full test sets: full-pass rate for Manufactoria and final-answer accuracy for mathematics. Invalid or missing answers are incorrect. Methods share prompts, verifiers, token limits, and decoding settings. Evaluation provides no execution feedback or repair; its token counts and compute are recorded separately.

System results use five root seeds, 0–4. Student-trial seeds derive from the root seed, round, and replicate index and are paired across candidates. Means and sample standard deviations are computed across complete runs. Method differences are paired by backbone, target split, and root seed before aggregation. Prediction and teacher-consistency analyses follow their repetition schemes in Appendices B.4 and D.9.

Shared budget and charged operations. Total training compute sums device time over all participating NVIDIA B200 GPUs. Within each backbone, benchmark, and seed, the common budget is the total training cost of our 100-round run. Methods share hardware, numerical precision, and this budget. Charged operations include teacher generation and updates, every candidate student branch, representation and outcome diagnostics, and calibration. Target-training and development screening are shared preparation costs charged equally to methods using the same benchmark protocol. Test-set screening and test-only evaluation are excluded from training compute.

Normalized compute and comparison endpoints. The lower row of Figure 8 and Table 2 use the same recursive-learning trajectories. They report cumulative total training compute as a percentage of the common budget, with 100% denoting the full-budget endpoint for both methods. The upper row evaluates fresh students under a separate student-training budget (Appendix D.8).

The matched-performance target is the Target-grounded Teacher's final mean greedy score: 46.6% on Manufactoria–HAS, 33.2% on MATH, and 17.4% on HARP. For each method, cost-to-target is read retrospectively from the first plotted checkpoint whose mean reaches or exceeds that target, using the same cumulative cost accounting as the full-budget comparison. This is the first crossing of the plotted mean curve. The values are read at the plotted checkpoints without interpolation.

Our target-reaching checkpoints occur at 65%, 75%, and 65% of the common budget; the baseline's occur at 100%, 95%, and 100%. Dividing our cost-to-target by the baseline's full-budget cost gives ratios of 0.65, 0.75, and 0.65. Comparing costs at each method's first target-reaching checkpoint gives reductions of $1 - 65/100 = 35\%$, $1 - 75/95 \approx 21.1\%$, and $1 - 65/100 = 35\%$, respectively.

D.8 FRESH-STUDENT CURRICULUM EVALUATION

For Figure 8's upper row, each final teacher generates 32 curricula of 32 examples (1,024 total). A fresh student trains from the common initial checkpoint for 800 optimizer steps, using the recursive trials' batching and sampling settings. Curricula are concatenated in order and cycled to the budget, with evaluation every 40 steps.

Direct target RL uses an equal-sized target-training sample, with replacement when needed; mathematical examples come from the fixed fail@128 training subset. Human-designed curricula retain their fixed family or composition-depth schedules. Each teacher seed pairs with one fresh-student seed, shared across conditions. Variation is measured across the five complete pairs. Full held-out Manufactoria–HAS, MATH, and HARP test sets use one greedy response per problem, scored by full-pass rate or mathematical accuracy.

D.9 REPEATED-STUDENT EVALUATION OF TEACHER CONSISTENCY

For Figure 9, multiple fresh students train on each teacher's fixed sampled curriculum under different student seeds. Within a teacher, the checkpoint, examples and order, initial student, optimizer, and budget remain fixed. Conditions share the student-seed list.

Feedback conditions. Uncertainty-driven RSI and the Target-grounded Teacher follow Appendix D.6. Representation-guided teaching uses Equation (8) for both teacher updates and student-branch selection. The shuffled-representation teacher uses the same candidate trials, but randomly permutes the measured $\Delta\Phi$ rewards among candidates within a round for the teacher update. Student-branch selection retains the correctly associated $\Delta\Phi$ rewards. This is the teacher-reward assignment control described in Section 5.2.

Within-teacher and between-teacher variation. Let $A_{j,b}(u)$ denote held-out student performance after u optimizer steps, for teacher j and fresh-student training seed b. With n_S student seeds per

teacher, the teacher-specific mean trajectory is

$$\bar{A}_j(u) = \frac{1}{n_S} \sum_{b=1}^{n_S} A_{j,b}(u). \tag{30}$$

Within-teacher variation is the sample standard deviation across student seeds. Between-teacher variation compares $\bar{A}_j(u)$ across teachers and includes both policy and curriculum-sampling differences. This repeated evaluation separates student-training variation within a teacher from variation across the complete pairs in Appendix D.8.

Figure-specific configuration. The benchmark is Manufactoria–HAS. Evaluation uses the complete held-out partition of 103 problems also used for this task in Figure 8. The score is greedy full-pass rate: a generated program succeeds only when it passes all test cases of its problem. Greedy decoding and the 4,096-token output limit follow Appendix D.5.

Each feedback condition has five independently trained teachers with seeds $\{0, 1, 2, 3, 4\}$. We use the final teacher checkpoint at the corresponding benchmark's system-training budget, following Appendix D.7; the representation-guided reference run uses 100 outer rounds. Each teacher generates 32 curricula of 32 examples, for 1,024 ordered examples, using curriculum-sampling seed 2027. The sampled examples and their order remain fixed across repeated student trials. Reward shuffling uses seed 2027 in the shuffled-feedback condition.

For each teacher, five fresh students start from the common initial student checkpoint and use training seeds $\{0, 1, 2, 3, 4\}$. Each trains for 800 optimizer steps, with evaluation every 40 steps. The batching and curriculum execution procedure follow Appendix D.8. This produces 25 fresh-student trials per feedback condition and 100 across the four conditions. Each curve corresponds to one teacher seed and plots Equation (30) with $n_S = 5$. Its shaded band is $\bar{A}_j(u) \pm s_j(u)$, where

$$s_j(u) = \sqrt{\frac{1}{n_S - 1} \sum_{b=1}^{n_S} \big(A_{j,b}(u) - \bar{A}_j(u)\big)^2}. \tag{31}$$

The band therefore measures student-training variation for a fixed teacher and curriculum.

D.10 Result Aggregation and Figure Specification

Figure 6 uses one common 0–100 color scale. Its eight columns are START, APPEND, EXACT, REGEX, COMPR, HAS, MATH, and HARP. The last two columns average over all problems in their respective full test sets. Each cell averages problem-level pass@32 within a run and then averages the five run scores.

Figure 7 uses the fixed mathematical hard-test manifests and a separate 128-sample evaluation of the final checkpoints. The horizontal axis is $k \in \{1, 4, 8, 16, 24, 32, 64, 128\}$, and the vertical axis is Δpass@k relative to Direct target RL, in percentage points. We calculate paired method differences within each run before taking their mean and sample standard deviation. Differences between two plotted curves give their pairwise pass@k difference at the same k.

In Figure 8, the upper row compares fresh students and the lower row compares recursively trained students. Both rows evaluate the full test sets using full-pass rate for Manufactoria and greedy accuracy for MATH and HARP.

E Additional MATH Feedback Comparisons

E.1 Comparison and Reporting Protocol

The comparison evaluates cohesion and likelihood progress as feedback signals in the recursive training loop. The three conditions are Target-grounded Teacher, Likelihood-progress Teacher, and REPRSI, with root seeds 0–4. The training configuration follows Appendix D.5: four candidate curricula per round, eight student optimizer steps per trial, and two trials per candidate ($K = 4$, $B = 8$, $R = 2$). Student training uses the final-answer verifier reward specified there.

Reference states and prompts. Likelihood-progress Teacher uses reference-state likelihood on the cohesion diagnostic examples. The reference output is the correct intermediate state used to define the semantic labels for cohesion, expressed in a fixed canonical text form. We reuse the state-likelihood prompt and chat formatting from Figure 3:

```
{diagnostic_context}
What is the intermediate state after {specified_operation}?
Return only the state.
```

The diagnostic context and specified operation form the input x_i; the reference state is the scored completion. The prompt and reference serialization remain fixed across candidates and before and after each student trial.

Length-normalized likelihood. For diagnostic example i, let $(s_{i,1}, \ldots, s_{i,L_i})$ be the reference-state content tokens included in scoring. We compute their log probabilities by teacher forcing, average within each example, and then average equally across diagnostic examples:

$$\ell(\theta; D_t) = \frac{1}{|D_t|} \sum_{i=1}^{|D_t|} \frac{1}{L_i} \sum_{j=1}^{L_i} \log p_\theta(s_{i,j} \mid x_i, s_{i,<j}). \tag{32}$$

Only reference-state content tokens contribute to the score; prompt tokens, padding, and special end-of-sequence tokens are excluded. This normalization gives each diagnostic example equal weight.

Closed-loop feedback. Likelihood progress averages the change over the R student trials:

$$r^{\mathrm{LL}}_{t,k} = \frac{1}{R} \sum_{a=1}^{R} \left[\ell(\theta'_{t,k,a}; D_t) - \ell(\theta_t; D_t) \right]. \tag{33}$$

The same D_t is used before and after each trial and across candidates within a round. This reward supplies both the teacher update and the selection of the continuing student branch, following the reward standardization and continuation rules in Appendix D.5. Reference states are used for diagnostic scoring; the student objective remains the final-answer verifier reward.

Two budget conventions. Trajectories cover outer rounds 0–120. We summarize checkpoints at 100 rounds and at the last recorded checkpoint within a common cap of 92.0178613 GPU-hours, displayed as 92.02. The latter cap is shared by all 15 runs; completed rounds and actual consumed compute consequently vary across runs. This additional comparison has its own absolute cap, distinct from the per-seed full-budget convention in Appendix D.7. Its scores and those in Table 2 refer to these different budget conventions.

Metrics and uncertainty. We report full-test pass@32 and greedy accuracy, using the metric definitions in Appendix D.7. Method summaries report the mean and sample standard deviation across the five runs. Paired gains subtract Likelihood-progress Teacher from RepRSI within each root seed. Their 95% intervals are $\bar{d} \pm t_{0.975,4} s_d/\sqrt{5}$, where s_d is the sample standard deviation of the five run-level differences. The complete run associated with each root seed is the unit of replication. Compute-indexed curves use the last recorded checkpoint within each budget, restricted to the interval covered by all runs; scores are neither interpolated nor extrapolated.

E.2 Closed-Loop Performance

RepRSI has higher mean performance under both budget conventions (Table 7). Under the common compute cap, its paired advantage is 3.51 points in pass@32 (95% interval $[2.29, 4.74]$) and 2.88 points in greedy accuracy ($[0.68, 5.07]$). Both differences are positive for all five seeds (Figure 10). At 100 rounds, the corresponding advantages are 1.85 and 2.16 points. Pass@32 improves in all five pairs and greedy accuracy in four; the remaining greedy difference is -0.20 points. Thus, the positive mean differences occur across multiple runs under both comparisons.

Between the budget-limited checkpoints and round 100, RepRSI's mean pass@32 decreases from 85.50% to 83.65%, while greedy accuracy rises from 37.09% to 38.36%. Initial greedy accuracy differs across methods within every seed (maximum within-seed range: 1.26 points), limiting attribution of endpoint differences to feedback alone.

Table 7: **Full MATH feedback comparison.** Values are mean ± SD across five runs; actual completed rounds and GPU-hours are shown. The budget-limited block uses the last checkpoint at or below 92.02 GPU-hours. Paired gains are RepRSI minus Likelihood-progress Teacher, with 95% t intervals (pp).

Method	Rounds	GPU-hours	pass@32 (%)	Greedy (%)
100 outer rounds				
Target-grounded	100	94.34 ± 3.78	81.76 ± 2.25	33.84 ± 0.85
Likelihood-progress	100	112.99 ± 4.79	81.80 ± 2.18	36.19 ± 1.83
RepRSI	100	99.47 ± 4.17	$\mathbf{83.65 \pm 2.73}$	$\mathbf{38.36 \pm 1.01}$
Paired gain (pp)	—	—	+1.85 [0.45, 3.24]	+2.16 [0.31, 4.01]
Common budget: 92.02 GPU-hours				
Target-grounded	96.8 ± 4.1	91.32 ± 0.10	81.80 ± 1.61	33.82 ± 1.33
Likelihood-progress	81.2 ± 3.6	91.79 ± 0.31	81.99 ± 2.05	34.22 ± 1.78
RepRSI	92.0 ± 3.9	91.47 ± 0.33	$\mathbf{85.50 \pm 1.81}$	$\mathbf{37.09 \pm 1.71}$
Paired gain (pp)	—	—	+3.51 [2.29, 4.74]	+2.88 [0.68, 5.07]

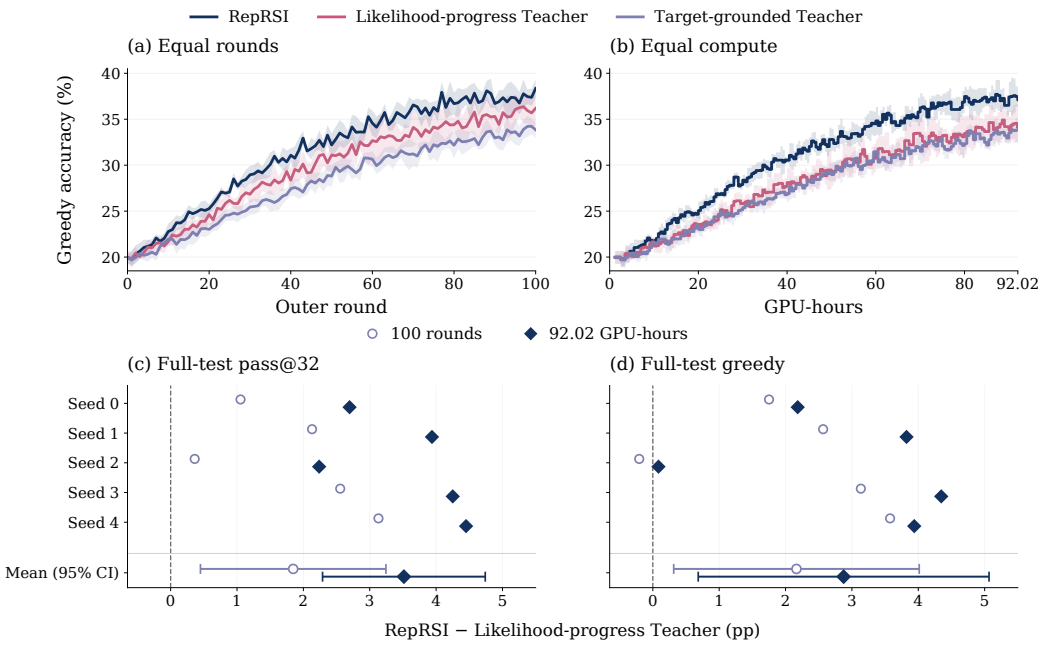

Figure 10: **MATH performance under two budget conventions.** (a) Greedy accuracy versus outer rounds. (b) Greedy accuracy versus cumulative compute budget, using the last recorded checkpoint within each budget. (c–d) Paired RepRSI minus Likelihood-progress Teacher differences at 100 rounds and at the common 92.02 GPU-hour cap. Curves show mean ± SD across five runs; paired means have 95% t intervals.

E.3 Compute Cost and Completed Rounds

Within the common cap, completed rounds average 92.0 ± 3.9 for RepRSI, 81.2 ± 3.6 for Likelihood-progress Teacher, and 96.8 ± 4.1 for Target-grounded Teacher. Actual compute averages 91.47, 91.79, and 91.32 GPU-hours, respectively (Table 8); unused budget ranges from 0.006 to 0.915 GPU-hours across the 15 runs.

At 100 rounds, RepRSI uses 99.47 GPU-hours on average and Likelihood-progress Teacher uses 112.99, a 12.0% reduction computed from the ratio of mean costs. Most of this difference lies in feedback computation: 13.03 versus 26.61 GPU-hours. Mean teacher costs are 13.13 versus 13.14 GPU-hours, and all-student-branch costs are 72.08 versus 72.01 GPU-hours. Shared preparation contributes 1.23 GPU-hours to each run. Together, the two comparisons describe recorded performance at 100 rounds and within a common compute cap, under which the methods complete different numbers of rounds.

Total compute comprises teacher generation and updates, all student branches, feedback computation, and shared preparation. Test-only evaluation follows the separate accounting convention in Appendix D.7.

Table 8: **Per-run compute accounting.** Costs are GPU-hours at the last checkpoint within 92.02 GPU-hours. Teacher includes generation and updates; student includes all candidate branches; feedback includes diagnostic scoring. Prep. denotes shared preparation, and unused denotes remaining budget.

Seed	Rounds	Teacher	Student	Feedback	Prep.	Total	Unused
Target-grounded Teacher							
0	101	12.24	69.99	8.01	1.23	91.48	0.54
1	93	12.65	69.75	7.71	1.23	91.33	0.68
2	98	11.98	70.55	7.48	1.23	91.24	0.77
3	92	13.01	68.95	8.04	1.23	91.22	0.79
4	100	13.20	69.35	7.52	1.23	91.31	0.71
Likelihood-progress Teacher							
0	84	11.00	58.54	21.00	1.23	91.77	0.25
1	78	10.89	57.65	21.49	1.23	91.26	0.76
2	82	10.48	58.58	21.71	1.23	91.99	0.02
3	77	10.39	58.31	22.07	1.23	92.01	0.01
4	85	10.52	58.75	21.40	1.23	91.91	0.11
RepRSI							
0	95	11.83	66.78	11.32	1.23	91.16	0.86
1	89	12.37	66.91	11.35	1.23	91.87	0.15
2	93	12.53	65.73	12.14	1.23	91.64	0.38
3	87	11.74	65.68	12.45	1.23	91.10	0.91
4	96	11.78	66.01	12.58	1.23	91.61	0.41

E.4 Reward-Pool and Monitoring-Pool Diagnostics

The diagnostics track RepRSI on the reward and monitoring pools at rounds 0, 10, 50, and 90. For each run and pool, changes are measured from that pool's own initial value. The cohesion definition gives $\Phi = C^+ - C^-$ and therefore $\Delta\Phi = \Delta C^+ - \Delta C^-$. Figure 11 shows these changes, and Table 9 reports their means and standard deviations.

Cohesion increases between successive recorded checkpoints in both pools for all five seeds. At round 90, mean $\Delta\Phi$ is 0.166 ± 0.016 on the reward pool and 0.141 ± 0.026 on the monitoring pool. In both pools, the mean increase combines higher C^+ and lower C^-; individual component trajectories can fluctuate. The positive monitoring-pool changes indicate that the reported cohesion increase extends to the separate monitoring examples. The smaller monitoring-pool change also motivates reporting both pools.

This analysis tracks the continuing student's representations across outer rounds. Candidate-ranking agreement and top-1 selection repeatability remain unmeasured because the analysis does not include

repeated evaluation of fixed candidate endpoints on independent diagnostic batches. The two-pool trajectories describe aggregate diagnostic changes. Candidate-level remeasurements are needed to assess selection repeatability; the extent of diagnostic adaptation remains unresolved.

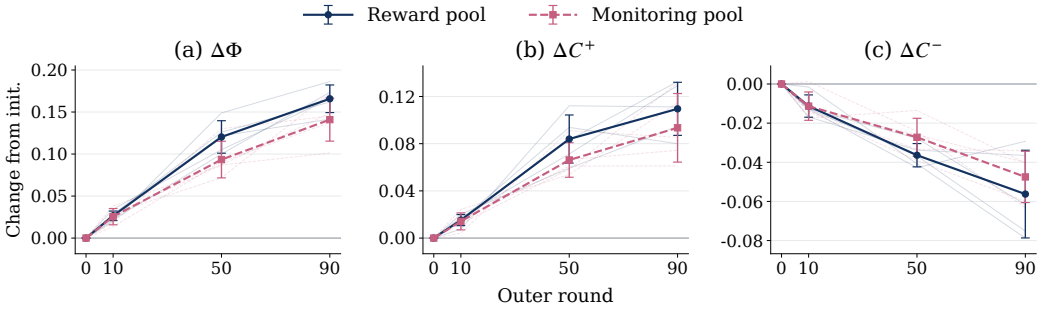

Figure 11: **Cohesion changes on reward and monitoring pools.** Changes in Φ, C^+, and C^- are relative to initialization within each pool. Thick lines show means, error bars show sample SD across five runs, and thin lines show individual runs. Connecting lines join the four recorded checkpoints.

Table 9: **Reward-pool and monitoring-pool diagnostics.** Changes are relative to initialization in each pool, reported as mean ± SD across five runs. Each run satisfies $\Delta\Phi = \Delta C^+ - \Delta C^-$ up to rounding.

Round	Pool	$\Delta\Phi$	ΔC^+	ΔC^-
10	Reward	0.027 ± 0.006	0.015 ± 0.005	-0.011 ± 0.006
10	Monitoring	0.025 ± 0.010	0.014 ± 0.007	-0.011 ± 0.007
50	Reward	0.120 ± 0.019	0.084 ± 0.020	-0.036 ± 0.006
50	Monitoring	0.093 ± 0.022	0.066 ± 0.015	-0.027 ± 0.010
90	Reward	0.166 ± 0.016	0.110 ± 0.022	-0.056 ± 0.022
90	Monitoring	0.141 ± 0.026	0.093 ± 0.029	-0.047 ± 0.013