Regressive Plasticity Schedule: A Two-Stage Post-Training Schedule for ARC Program Synthesis

Jason Feng

https://x.com/iamjasonfeng

Abstract

This paper presents Regressive Plasticity Schedule (RPS), a two-stage post-training schedule inspired by developmental plasticity. RPS combines two familiar ideas, curriculum learning and learning-rate reduction, in a specific way: the model is first trained on easier data at a higher learning rate, then trained on harder data at a substantially lower learning rate. This differs from ordinary learning-rate decay because the main intervention is not merely reducing the optimizer step size over time within one training stage; instead, RPS couples a discrete stage-level learning-rate drop to a curriculum boundary between easier and harder data. The broader motivation for RPS is to improve general reasoning: program synthesis is an important testbed here, but the more important question is whether staged plasticity can help models preserve foundational reasoning behaviors while adapting to harder domains. I tested RPS on Qwen3-8B using Alibaba Model Studio managed DPO fine-tuning with LoRA. The control condition, Equal Plasticity Schedule (EPS), used the same model, same two-stage data structure, same within-stage cosine scheduler, and same Stage 1 checkpoint, but did not reduce the Stage 2 learning rate. On ARC-AGI-1 public evaluation, RPS improved exact test-output accuracy from 10/419 to 17/419, which provides evidence of improved ARC-style general reasoning because these tasks require inferring and applying latent transformation rules from few examples. On ARC-AGI-2 public evaluation, neither RPS nor EPS solved any test outputs, but RPS substantially improved program-synthesis reliability: 234/240 attempted programs executed without error for RPS, compared with 188/240 for EPS. The result does not show that RPS solves ARC-AGI-2, but it suggests that a curriculum-coupled plasticity reduction can improve ARC-style reasoning behavior and make a model more stable at producing usable reasoning artifacts. If this pattern generalizes beyond ARC, RPS could have large potential as a simple post-training schedule for improving broader reasoning behavior.

1. Introduction

ARC-AGI is designed to test abstraction and generalization from small numbers of examples.[chollet-2019][arc-2026] ARC tasks are difficult for ordinary language-model prompting because success requires inferring a latent transformation rule and applying it exactly to new grids. Recent ARC systems increasingly combine learning, search, program synthesis, test-time adaptation, and synthetic data rather than relying on a single model call.[arc-2025]

This paper explores a small post-training hypothesis: for ARC-style reasoning, it may be useful to reduce model plasticity when moving from easier foundational tasks to harder target tasks. The larger goal is not merely to make models write more reliable programs. It is to test whether a developmental plasticity schedule can improve general reasoning by helping a model first acquire broad reusable patterns, then adapt to harder tasks without overwriting those patterns too aggressively. The intuition is inspired by human development. Humans learn basic perceptual and symbolic skills early, when plasticity is high, and later acquire more specialized skills with lower plasticity. RPS is an attempt to translate that intuition into a simple post-training schedule.

In this experiment, plasticity is operationalized through the learning rate. Stage 1 trains on easier ARC-style examples with a higher learning rate. Stage 2 trains on harder ARC-AGI-2 examples with 10% of the Stage 1 learning rate. This is not proposed as a wholly new primitive. RPS is best understood as curriculum learning plus a stage-level learning-rate reduction. The specific question is whether coupling the learning-rate drop to a curriculum boundary improves ARC behavior compared with an equal-plasticity control.

2. Prior Work

2.1 ARC-AGI and Program Synthesis

ARC-AGI tasks are grid transformation problems where each task provides a small set of training input-output pairs and one or more test inputs.[arc-guide] A system must infer the transformation and produce exact test outputs. Program synthesis is a natural approach: generate a candidate transformation program, execute it on the training examples, reject programs that fail the demonstrations, and apply surviving programs to test inputs.

The 2025 ARC Prize results showed that strong ARC systems can use refinement loops, synthetic data, test-time training, and specialized models.[arc-2025] This paper studies whether RPS can improve the reliability of the kind of program-synthesis behavior that may be useful inside such systems. Program synthesis is treated as a concrete reasoning behavior that can be measured, not as the final scope of RPS.

2.2 Curriculum Learning and Learning-Rate Schedules

Curriculum learning trains models on data ordered by difficulty. Learning-rate schedules reduce or otherwise vary the optimizer step size during training. RPS combines these ideas but ties the plasticity change to a curriculum boundary: the model moves from easier data to harder data while the base learning rate drops sharply.

This matters because ordinary cosine decay within a single stage is not the whole intervention. In this experiment, both RPS and EPS used cosine learning-rate schedules within stages. The difference was the stage-level base learning rate used for Stage 2.

2.3 Preference Optimization

The experiment used DPO-style preference fine-tuning.[dpo] Each training example contained a prompt, a chosen response, and a rejected response. The chosen response contained an ARC reasoning trace, a Python transformation program, and predicted test outputs. The rejected response was generated by a weaker Qwen3-0.6B model under normal solving conditions, not by instructing the model to intentionally be wrong.

3. Method

3.1 Regressive Plasticity Schedule

RPS is a two-stage schedule:

```
Stage 1: easier ARC-style examples, higher learning rate
Stage 2: harder ARC-AGI-2 examples, lower learning rate
```

In this experiment:

```
Stage 1 learning rate: 1e-5
Stage 2 learning rate: 1e-6
```

Stage 2 therefore used 10% of the Stage 1 learning rate.

3.2 Equal Plasticity Schedule Control

EPS is the control condition. It used the same two-stage structure and the same Stage 1 model, but did not reduce the Stage 2 learning rate:

```
Stage 1 learning rate: 1e-5
Stage 2 learning rate: 1e-5
```

Both RPS and EPS used cosine learning-rate schedules within each stage. EPS therefore controls for the fact that each stage already has within-stage learning-rate decay. The main difference between RPS and EPS is the

stage-level reduction in the Stage 2 base learning rate.

3.3 Base Model and Fine-Tuning Setup

The base model was Qwen3-8B.[qwen3] Fine-tuning was performed with Alibaba Model Studio managed DPO fine-tuning using LoRA.[dpo][lora]

Shared settings:

```
Base model: qwen3-8b
Fine-tuning method: DPO
Adapter method: LoRA
Validation split: 1%
Maximum sequence length: 32k
Scheduler: cosine
RPO alpha: 1
Stage 1 dataset size: 400 preference pairs
Stage 2 dataset size: 600 preference pairs
Replay: none
ARC-AGI public evaluation data used for training: none
```

3.4 Dataset

The ARC DPO dataset contained 1,000 preference pairs.

Stage 1 contained 400 pairs from easier ARC-style training tasks. It contained 400 unique tasks. Most examples came from ARC-AGI-1 or ARC-AGI-1/2-overlap tasks, with a small number of ARC-AGI-2 training examples used to fill the quota.

Stage 2 contained 600 pairs from ARC-AGI-2 training tasks only. It contained 250 unique tasks, with some duplicate task IDs allowed to fill the 600-pair quota. No task IDs overlapped between Stage 1 and Stage 2.

Each DPO pair contained:

```
prompt: ARC training examples and test inputs
chosen: reasoning, Python transform program, predicted test outputs
rejected: weaker model response generated under normal solving conditions
```

The chosen responses came from Trelis/arc-agi-2-reasoning-5, a dataset of ARC reasoning traces and Python program-synthesis solutions.[trelis] Rejected responses were generated by Qwen3-0.6B. Both chosen and rejected responses were normalized into the same labeled structure:

```
<think>
...
</think>

Python program:
```

def transform(grid): ...

```
Predicted test outputs:
...
```

The rejected responses were not instructed to be wrong. They were generated by querying the weaker model as if it were solving the task normally. Because Qwen3-0.6B is weak on ARC, these responses were expected to be lower quality than the chosen responses.

4. Evaluation

4.1 ARC-AGI-1 Public Evaluation

ARC-AGI-1 public evaluation was run over 400 tasks.[arc-guide] Each task used three attempts, yielding 1,200 generation attempts. The official score was computed over 419 test outputs. A test output was counted correct if any attempt exactly matched the ground truth output.

4.2 ARC-AGI-2 Public Evaluation

ARC-AGI-2 public evaluation was run over 120 tasks with two attempts per test input grid.[arc-guide] The official score was computed over 167 test outputs.

4.3 Program-Synthesis Reliability

The models were evaluated not only by final exact output accuracy but also by whether they produced executable Python programs. An attempt was counted as an executed-program output when the generated Python program ran without error and produced valid grid outputs that were used for scoring. Parsed JSON fallbacks were not counted as program-synthesis success.

This metric is not a substitute for task accuracy. It also is not a full measure of general reasoning. It measures whether the model stayed in the intended program-synthesis mode and produced usable executable artifacts, which I use as one observable proxy for structured reasoning behavior.

5. Results

5.1 ARC-AGI-1 Exact Test-Output Accuracy

RPS improved ARC-AGI-1 public evaluation score relative to EPS.

Model	Correct test outputs	Total test outputs	Score
RPS	17	419	4.06%
EPS	10	419	2.39%

At the task level, RPS solved 14/400 tasks, compared with 10/400 for EPS.

5.2 ARC-AGI-1 Program-Synthesis Reliability

RPS also improved program-synthesis reliability on ARC-AGI-1.

Metric	RPS	EPS
Attempts	1,200	1,200
Program executions without error	1,145	870
Attempts scored from executed programs	1,086	812
Invalid attempts	26	250
Tasks with all attempts scored from executed programs	306/400	137/400
Tasks with at least one executed-program attempt	398/400	382/400
Tasks with zero executed-program attempts	2/400	18/400

5.3 ARC-AGI-2 Exact Test-Output Accuracy

Neither model solved any ARC-AGI-2 public-evaluation test outputs.

Model	Correct test outputs	Total test outputs	Score
RPS	0	167	0.00%
EPS	0	167	0.00%

This is the central limitation of the current result. RPS did not solve ARC-AGI-2 in this standalone setup.

5.4 ARC-AGI-2 Program-Synthesis Reliability

Despite the zero exact score on ARC-AGI-2, RPS substantially improved program-synthesis reliability.

Metric	RPS	EPS
Attempts	240	240
Program executions without error	234	188
Attempts scored from executed programs	214	176
Parsed-JSON fallback attempts	18	19
Invalid attempts	8	45
Tasks with both attempts scored from executed programs	100/120	60/120
Tasks with at least one executed-program attempt	114/120	116/120
Tasks with zero executed-program attempts	6/120	4/120

Paired attempt-level comparison:

Comparison	Count
Both RPS and EPS produced executed-program outputs	160/240
RPS only produced an executed-program output	54/240
EPS only produced an executed-program output	16/240
Neither produced an executed-program output	10/240

The paired difference in ARC-AGI-2 executed-program attempts was significant under an exact McNemar test, with p = 5.85e-6. This statistical test should be interpreted cautiously because attempts are not fully independent, but it supports the view that the observed reliability difference is unlikely to be random noise.

5.5 Token and Thinking-Budget Checks

Thinking-budget saturation was rare and does not appear to explain the RPS/EPS difference.

ARC-AGI-1:

```
RPS thinking-budget hits: 0/1200
EPS thinking-budget hits: 15/1200
```

ARC-AGI-2:

```
RPS thinking-budget hits: 1/240
EPS thinking-budget hits: 6/240
```

For ARC-AGI-1, excluding test outputs with thinking-budget-hit attempts produced:

Model	Correct test outputs	Kept test outputs	Score
RPS	17	405	4.20%
EPS	10	405	2.47%

Thus the ARC-AGI-1 result remains favorable to RPS after excluding thinking-budget-hit cases.

6. Discussion

6.1 What RPS Appears To Improve

RPS produced two encouraging signals. First, it improved ARC-AGI-1 exact test-output accuracy relative to EPS. Because ARC-AGI-1 tasks require inferring abstract grid transformations from small numbers of examples, this is evidence of improved ARC-style general reasoning, not merely improved formatting or program syntax.

Second, on both ARC-AGI-1 and ARC-AGI-2, RPS improved program-synthesis reliability. RPS made the model more likely to produce a runnable Python transformation program and less likely to produce invalid responses.

This matters because ARC systems often need to generate, test, repair, and rank candidate programs. A model that reliably produces syntactically and semantically executable transformation programs may be more useful inside a larger ARC system than a model that produces free-form reasoning or invalid outputs.

The larger potential of RPS is broader than program synthesis. The ARC-AGI-1 exact-score improvement is the clearest current evidence for reasoning improvement, while program synthesis provides an additional objective way to measure whether a model can produce structured, checkable reasoning artifacts. The reason RPS is interesting is that the same high-to-low plasticity pattern may help in other reasoning domains where a model must first learn reusable foundations and then adapt to harder tasks without excessive representational drift. From this perspective, the ARC-AGI-1 accuracy gain and the program-synthesis reliability gain are early signals about a more general post-training idea.

6.2 Why EPS Is The Key Control

One possible objection is that RPS is simply learning-rate decay. EPS helps address this. Both RPS and EPS used two stages. Both used cosine schedules within each stage. Both used the same Stage 1 checkpoint and the same Stage 2 dataset. The main difference was that EPS kept the Stage 2 learning rate equal to Stage 1, while RPS reduced the Stage 2 learning rate to 10% of Stage 1.

Therefore, the observed difference is not explained by the mere presence of within-stage cosine learning-rate decay. The result is more consistent with the hypothesis that the curriculum boundary and the stage-level plasticity reduction matter together.

6.3 Limitations

This experiment has several important limitations.

First, RPS did not solve any ARC-AGI-2 public-evaluation test outputs in the standalone setup. The ARC-AGI-2 result is therefore a reliability result, not an accuracy result.

Second, the experiment used one base model and one dataset construction. The result should be tested across model sizes, preference datasets, SFT/RL variants, multiple random seeds, and non-ARC reasoning benchmarks. This is especially important because the long-term motivation for RPS is general reasoning, not only ARC program synthesis.

Third, the rejected responses were generated by a weak model and were not independently verified as incorrect. The assumption was that the chosen program-synthesis traces were higher quality than Qwen3-0.6B generated responses. This is plausible but imperfect.

Fourth, some Stage 2 task IDs were duplicated to fill the 600-pair quota. Future experiments should test larger and cleaner datasets with less duplication.

Fifth, the current results are public-evaluation results and should not be treated as private leaderboard performance.

7. Conclusion

RPS is a simple curriculum-coupled plasticity schedule: first train on easier data with higher plasticity, then train on harder data with lower plasticity. Its ultimate motivation is improving general reasoning, with ARC serving as a concrete and measurable testbed. In a Qwen3-8B DPO experiment on ARC-style program synthesis, RPS improved ARC-AGI-1 exact test-output score relative to an equal-plasticity control and substantially improved program-synthesis reliability on both ARC-AGI-1 and ARC-AGI-2 public evaluations.

The result is preliminary. It does not show that RPS solves ARC-AGI-2, and it does not yet prove broad cross-domain general-reasoning improvement. However, the ARC-AGI-1 exact-score gain is evidence of improved ARC-style general reasoning. Together with the program-synthesis reliability gains, it suggests that stage-level plasticity reduction may help stabilize structured reasoning behavior when moving from easier tasks to harder tasks. The next step is to test RPS inside a larger ARC solver that uses candidate generation, execution, filtering, and ranking, and then test whether the same schedule improves broader reasoning benchmarks beyond ARC.

Acknowledgments

I came up with the RPS idea myself and ran the experiments independently. I used Codex to help with the training workflow, dataset preparation, evaluation notebooks, analysis, and writing of this paper.

References

[chollet-2019]: Chollet, F. (2019). On the Measure of Intelligence. arXiv:1911.01547.

[arc-2026]: ARC Prize Foundation. ARC Prize 2026.

[arc-guide]: ARC Prize Foundation. ARC Prize Guide.

[arc-2025]: ARC Prize Foundation. ARC Prize 2025 Results and Analysis.

[qwen3]: Qwen Team. (2025). Qwen3 Technical Report. arXiv:2505.09388.

[dpo]: Rafailov, R., Sharma, A., Mitchell, E., Manning, C. D., Ermon, S., and Finn, C. (2023). Direct Preference Optimization: Your Language Model is Secretly a Reward Model. arXiv:2305.18290.

[lora]: Hu, E. J. et al. (2021). LoRA: Low-Rank Adaptation of Large Language Models. arXiv:2106.09685.

[trelis]: Trelis Research. arc-agi-2-reasoning-5.