Learning Curriculums for Abstract Reasoning on ARC-AGI

Markus Zhang [1]

Abstract

ARC-AGI-1, as a reward environment, has provably correct verifiers for correctness–suitable for DeepSeek-r1-series reasoning models. In this report we attempt to train an LLM policy capable of solving ARC-AGI puzzles via Group Relative Policy Optimization (GRPO) and curriculum learning. We evaluate bootstrapped baselines, including a small model fine-tuned on distilled reasoning traces, transduction- or induction-based outputs, and context length; we conclude that an untuned base model with DSL code generation is best. Then, we construct code sandboxes and DSL linters to shape rewards for both code correctness and style. Experiments show the hand-crafted reward curriculum successfully nudged the policy to output correct, executable DSL code. We observe slow unstable improvement on easy training puzzles, and unsucessful transfer to any evaluation puzzle within a 1-day 1xH100 compute budget.

1. Introduction

The ARC-AGI-1 (Chollet et al., 2024) benchmark has been saturated by compute-inefficient reasoning (o3) or test-time memorization (OmniARC) models (OpenAI, 2024) (Barbadillo, 2024). Recent reasoning models like DeepSeek-R1-Zero have seen tremendous success in STEM reasoning by defining reward functions in tasks where the solution is provably correct or incorrect–for example, math questions by their numerical answers, and coding problems by their test cases. Puzzle attempts in ARC-AGI are also provably correct. In the *transduction* case, where the model learns to explicitly output the guessed grid, the guess is either an exact match or not. In *induction*, where the model learns to output a runnable function representing the guessed pattern, the function either yields the correct output for a given test input or not.

Framing ARC-AGI as a reasoning problem, we investigate whether the smaller reasoning model can improve its baseline performance with Group Relative Policy Optimization (Shao et al., 2024): sampling online trajectories in a group, estimating relative advantage, updating trust-region policy gradients.

ARC-AGI should be a *narrow* reasoning task, in the sense that inputs and outputs grid shapes are well-defined and discrete, which implies that the number of abstract ideas that can be encoded in each puzzle should be relatively small. In contrast, many RL agents solve broad tasks, like independent browsing, where the ideal output is ill-defined. Therefore we hypothesized that a small model could learn to reason over these fixed patterns, but policy improvement was significantly slower and harder than exepcted, as will become clear.

2. Related Work

GRPO The DeepSeek R1 series (DeepSeek, 2025) builds from verification-based learning, starting with R1-Zero from V3 baseline on GRPO with rule-based rewards for math and code tasks. R1 then warm-starts from human-cleaned R1-Zero thought patterns or few-shot learning with extended chain-of-thought reasoning.

DSL (Hodel, 2024) constructed a domain-specific language with syntax, generator, and verifier functions specifically designed for ARC tasks. Their verifier automatically checks test-case outputs without requiring natural language explanations. While their DSL covers all 400 training and 400 public evaluation tasks, it potentially overfits to the training distribution since it was iteratively designed alongside task solvers.

Augmentation Several efforts focus on expanding the limited ARC dataset. BARC (Li et al., 2024) generated 400,000 ARC-Heavy remixed examples using GPT-4o, though only seed tasks were verified. The MIT approach (Akyürek et al., 2024) leverages permutation transforms and in-context learning in a leave-one-out fashion, applying auxiliary losses on context grids. They create new LoRA adapters per test task to maximize performance. MindsAI's unpublished work, known primarily through reverse-engineering efforts, combines synthetic data from reARC with geometric transformations, followed by test-time fine-tuning.

[1]Department of Computer Science. Correspondence to: Markus Zhang <markusz@stanford.edu>.

Voting OmniARC (Barbadillo, 2024) employs multiple surrogate approaches including traditional example-to-output mapping, code induction solving, input distribution learning, verification, and output selection through voting. Their workflow involves fine-tuning smaller models like Qwen-2.5-0.5B with LoRA, augmenting test sets, and applying ensemble voting. The MIT team uses two-round voting across geometric permutations and leave-one-out orderings.

Rejection Sampling (Greenblatt, 2024) used the GPT-4o API only with massively parallel rejection sampling and iterative healing (ask model to fix its own mistakes given code output) to achieve 50% on ARC Public Eval. (Berman, 2024) extended this with guided sampling via evolutionary algorithms that reproduce good reasoning strands that have high correct rates on the puzzle's in-context examples.

We borrow DSL work from (Hodel, 2024) and GRPO from (Shao et al., 2024), while other test-time scaling ideas do not obviously apply due to the different SFT / RL training settings.

3. Initial Attempts

3.1. Transduction

Initially, our approach was not code generation but direct transduction. That is, we aimed to train the model to generate the direct $(m \times n)$ output grid rather than the Python DSL function that solves any given input grid. Given the significant performance gap between small models and larger reasoning models like R1, we sought to create a strong fine-tuned baseline.

Our first strategy involved bootstrapping a good policy by fine-tuning on the reasoning traces of larger models–a form of distillation found in s1 (Muennighoff et al., 2025). Verification was performed through rejection sampling, but this turned out to be highly inefficient. Even larger models struggled to solve most tasks, and increasing the number of rejections yielded diminishing returns.

3.2. Reasoning Distillation

We encountered a "blabbering" problem: teacher models like `DeepSeek-R1-Llama-70B` would produce excessively long reasoning traces that often overflowed the 32k context window. These traces frequently contained many incorrect approaches before potentially arriving at a correct conclusion. For example, one model persistently tried to count color numbers and sum them across rows—-a fundamentally flawed approach since operations like adding colors (e.g., green $= 3$, blue $= 2$, but green + blue $\neq 5$) are not meaningful in the context of these puzzles. Even explicit prompting against such invalid operations failed to redirect the model's reasoning.

Over 100 generations, larger models still demonstrated difficulty, and increasing the magnitude of rejection sampling barely improved results. For k distinct thought patterns in a reasoning trace separated by {`Wait, Alternatively, But, ...`}, often the $(k-1)$ patterns were incorrect, with only the last 1 step. A r1 reasoning model would procrastinate, with a correct pattern appearing near the end of the model's context window, as documented in (Qu et al., 2025). This indicates that unnecessarily long reasoning traces risk rewarding incorrect thought patterns since the rejection sampling rule is "final step correct".

As an alterative, we tried generating reasoning traces by prompting with a `(problem, solution)` pair–the key difference being that the solution is given. However, we observed test leakage in generated reasoning, where the model would reveal it already knows the answer. Such a training set can cause hallucination.

For these reasons and high API cost, accepted teacher samples were too sparse for distillation to be feasible, much less efficient. Therefore we attempted direct GRPO with an untuned HuggingFace baseline policy model.

3.3. Induction

Even without fine-tuning, we observed that reasoning models had a tendency to hallucinate incorrect reasoning patterns and verification steps, so we switched to inductive code generation. Instead of manually verifying correctness within in-context examples, we leveraged the model's latent coding ability from pretraining and distillation.

We also considered starting with easy-rated tasks via curriculum learning, allowing teacher models to generate more reliable reasoning traces for simpler problems. However, this raised questions about whether such selective SFT would provide a better prior than direct reinforcement learning.

Inductive code generation provided the additional benefit of being executable on all in-context examples. If a program $\hat{f}$ was correct for the in-context examples $(x_i, y_i)_{i=1}^n$, it is known to be also correct for the test example (x_{n+1}, y_{n+1}) (Hodel, 2024). Moreover, code execution shifts the policy π's action space $\mathcal{A}$ from the set of all $n \times m$ grids to the set of all DSL programs. Since we know the CommonCrawl pretraining data has code aplenty and fewer puzzles, we hypothesized that code generation would be a more natural transfer task for the base LLM, and therefore more sample efficient. In the future, the execution output can be a tool response to allow self-healing and end-to-end training with multi-tool calling.

4. Final Method

4.1. GRPO

Following (DeepSeek, 2025; Shao et al., 2024), we adopt Group Relative Policy Optimization which maximises:

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{q\sim P(Q),\{o_i\}_{i=1}^{G}\sim\pi_{\theta_{old}}(O|q)}$$
$$= [\frac{1}{G}\sum_{i=1}^{G}(\min(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{old}}(o_i \mid q)}A_i, clip(\frac{\pi_\theta(o_i \mid q)}{\pi_{\theta_{old}}(o_i \mid q)}, 1-\varepsilon,$$
$$1+\varepsilon)A_i) - \beta\mathbb{D}_{KL}(\pi_\theta\|\pi_{ref}))]$$

where advantage A_i is estimated critic-free relative to the group's mean normalized rewards:

$$A_i = \frac{r_i - \text{mean}(\{r_1, r_2, \ldots, r_G\})}{\text{std}(\{r_1, r_2, \ldots, r_G\})}$$

instead of typical PPO advantage $A_i = Q^{\pi_{\theta_{old}}}(o_i, q) - V^{\pi_{\theta_{old}}}(q)$.

We train the baseline `DeepSeek-R1-Distill-Qwen-7B` with Unsloth (Unsloth, 2025), a patch of HuggingFace TRL (HuggingFace, 2025) optimized with single-GPU CUDA kernels. To run online GRPO, we alternate between sampling trajectories on vLLM (vLLM, 2025) and updating gradients on TRL. It implies a significant memory overhead with a full cycle of model unload/load per batch, and lowers our MFU to 50%, which is an area for systems optimization left to future work.

However, initial results showed that the model struggled primarily with index chasing—handling complex displacements within a grid. The model often had the correct intuition but failed to implement it correctly in code. This flaw in reward function design led to situations where the model was not rewarded for correct reasoning if the code contained an index error. Consequently, the advantage was low, and the model received zero reward despite demonstrating partial correctness.

To mitigate this, we adopted a domain-specific language (DSL) tailored to ARC. This DSL featured a lexer, parser, and static analysis tools, enabling a gradient-correctness reward function to improve compute efficiency. By framing puzzles as pure functional operations with high-level object selection and filling capabilities, we narrowed the gap between correct reasoning and correct implementation.

This adaptation brought our reward function closer to the ideal reward formulation, aligning execution correctness with reasoning correctness. This approach proved promising, laying the foundation for future improvements through curriculum learning, end-to-end multi-tool calling, and test-time rejection sampling.

5. Method

5.1. GRPO

In GRPO, the advantage is relative to a group, meaning that absolute reward values do not matter. We applied a square root function to rewards, ensuring diminishing returns past a certain point. This helps prevent reward drift while keeping the focus on improvement.

5.2. Environment Design: DSL and Execution Sandbox

To ensure execution ran smoothly, we implemented an isolated execution sandbox. This was necessary to prevent reward hacking, where the model could exploit unintended behaviors by accessing networks or files. The sandbox also prevented a single failed execution from crashing the entire training process by running each task in a separate subprocess.

Ensuring reliable code interpretation was a challenge. Initially, we added a tracer to track variable states and intermediate assignments, similar to a debugger. However, models frequently generated code that broke tracer assumptions, such as omitting return statements or creating infinite loops. Due to this instability, we ultimately decided to only inspect full function inputs and outputs.

We believe a properly implemented tracer could be useful in the future, especially for debugging long DSL-generated code where standard input-output diffs fail to locate root causes of errors.

5.3. Lexers, Linters, and Parsers

The model showed initial reluctance to use our DSL, defaulting to deeply nested Python indexing notation or redundantly re-importing function definitions. To counteract this, we implemented a style-based reward function: - Positive rewards for using DSL functions correctly. - A sparsity penalty for generated code exceeding 100 lines. - Penalties for more than four function definitions to discourage redundant definitions.

We used Abstract Syntax Trees (AST) and Concrete Syntax Trees (CST) for reliable static analysis of generated code. Manual linting rules were applied to detect excessive function definitions and enforce stylistic consistency.

5.4. Reward Function

Our reward function is split into two main components:

1. **Style Reward** - Primarily composed of penalties to discourage poor coding practices rather than rewards for good style.
2. **Execution Reward** - Measured by the percentage of

correctly generated grid pixels. This correctness score follows a polynomial curve, where gains for improving the last few correct grids are significantly steeper than early improvements.

This design counters the increasing marginal effort required to reason correctly about additional correct cells in the grid, ensuring that late-stage improvements are still incentivized.

6. Conclusion

While we observed success in solving most easy tasks within the training dataset, we did not observe significant transfer to the evaluation set. Nevertheless, we believe this direction holds promise, especially with further enhancements in curriculum learning and refined reward delta functions. Future work will focus on addressing the remaining issues with reasoning trace fidelity and leveraging programmatic verification to further improve model reliability.

7. Introduction

This project examines if a distilled r1 model can match r1 on ARC-AGI-1 performance (or substantially improve base performance) by deliberately constructing a small, handcrafted dataset of reasoning tasks similar to ARC-AGI-1 (Chollet et al., 2024; DeepSeek, 2025). If a small model saturates performance on this single domain, it raises concerns regarding the validity of such isolated evaluations in testing AGI, implying small models well-optimized for domain-specific reasoning tasks may be misleading, and frontier models should be tested across an ensemble of reasoning tasks from very different domains.

8. Data

Our dataset begins with 400 public ARC-AGI (question, answer) tasks focused on visual pattern recognition in nxm grid boxes. We will augment this seed set by using a larger model (o3 or r1) to generate an additional 10–100$\times$ diverse QA pairs through permutations and repeated sampling, as described by the curriculum generation of phi-series papers. Each generated pair, along with its reasoning trace, will be verified by three human labellers; any disagreement will result in the pair's rejection, ensuring a correct reward.

9. Method

To post-train a r1-distilled 7B model of Llama architecture, we will apply GRPO—a variant of PPO—with a rule-based reward that is positive when the model's answer matches the human/o3 reference, and zero otherwise (Schulman et al., 2017; Shao et al., 2024). If this reward signal is too sparse for policy model convergence, alternatively we can try a more continuous reward of percentage boxes correct in the visual grid of the ARC-AGI-1 task. We will also attempt s1's budget forcing method of test-time scaling: if the model's internal "thinking" sequence terminates before reaching a desired token count n, the premature "`</think>`" token is replaced with "Wait," and if it exceeds n, the sequence is truncated and ended with "`</think>` (Muennighoff et al., 2025)." This may amplify the DSL effect observed in R1-Zero: where language mixing and uninterpretable thinking token could be the policy optimizing its thinking for a narrow domain task.

10. Literature Review

The phi-series and recent s1 papers will inform our dataset curation, while R1, R1-Zero, and v3 are the best open-source reasoning models, particularly the rule-based reward of R1-Zero (Microsoft, 2024). DeepSeek-Math and Open-R1 offer more details to replicate the reasoning process, ARC-AGI's leaderboard papers will inform task-specific methods. GRPO and PPO learns our policy.

11. Evaluation

We will attempt to request the ARC foundation to evaluate on ARC-AGI-1 Semi-Private, or Public otherwise. We expect our smaller distilled model to compare with R1's metrics only on ARC-AGI-1, and to be likely much worse on all other benchmarks. Qualitatively, metrics should scale with generated data size and thinking token count. However, the possible failure modes are: the Semi-Private score is much lower due to overfitting; if so, our hypothesis that a small reasoning model can generalize to one well-defined task is false. It is also possible that the reward signal is too sparse to converge, or improvement is too slow at small model sizes.

References

Akyürek, E., Damani, M., Qiu, L., Guo, H., Kim, Y., and Andreas, J. The Surprising Effectiveness of Test-Time Training for Abstract Reasoning, November 2024.

Barbadillo, G. arc24, 2024. URL `https://ironbar.github.io/arc24/`.

Berman, J. How I came in first on ARC-AGI-Pub using Sonnet 3.5 with Evolutionary Test-time Compute, dec 2024. URL `https://jeremyberman.substack.com/p/how-i-got-a-re`

Chollet, F., Knoop, M., Kamradt, G., and Landers, B. Arc prize 2024: Technical report. *arXiv preprint arXiv:2412.04604*, 2024.

DeepSeek, D. Deepseek-r1: Incentivizing reasoning capa-

bility in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025.

Greenblatt, R. Getting 50% (SoTA) on ARC-AGI with GPT-4o, jun 2024. URL https://redwoodresearch.substack.com/p/getting-50-sota-on-arc-agi-with-gpt.

Hodel, M. Addressing the Abstraction and Reasoning Corpus via Procedural Example Generation, apr 2024. URL http://arxiv.org/abs/2404.07353.

HuggingFace. TRL: Transformer Reinforcement Learning, 2025. URL https://github.com/huggingface/trl. original-date: 2020-03-27T10:54:55Z.

Li, W.-D., Hu, K., Larsen, C., Wu, Y., Alford, S., Woo, C., Dunn, S. M., Tang, H., Naim, M., Nguyen, D., Zheng, W.-L., Tavares, Z., Pu, Y., and Ellis, K. Combining Induction and Transduction for Abstract Reasoning, 2024. URL http://arxiv.org/abs/2411.02272.

Microsoft, M. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024.

Muennighoff, N., Yang, Z., Shi, W., Li, X. L., Fei-Fei, L., Hajishirzi, H., Zettlemoyer, L., Liang, P., Candès, E., and Hashimoto, T. s1: Simple test-time scaling. *arXiv preprint arXiv:2501.19393*, 2025.

OpenAI. OpenAI o3 Breakthrough High Score on ARC-AGI-Pub, 2024. URL https://arcprize.org/blog/oai-o3-pub-breakthrough.

Qu, Y., Yang, M. Y. R., Setlur, A., Tunstall, L., Beeching, E. E., Salakhutdinov, R., and Kumar, A. Optimizing Test-Time Compute via Meta Reinforcement Fine-Tuning, mar 2025. URL http://arxiv.org/abs/2503.07572.

Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

Shao, Z., Wang, P., Zhu, Q., Xu, R., Song, J., Bi, X., Zhang, H., Zhang, M., Li, Y. K., Wu, Y., and Guo, D. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024. URL https://arxiv.org/abs/2402.03300.

Unsloth. unslothai/unsloth, 2025. URL https://github.com/unslothai/unsloth. original-date: 2023-11-29T16:50:09Z.

vLLM. vllm-project/vllm, 2025. URL https://github.com/vllm-project/vllm. original-date: 2023-02-09T11:23:20Z.