From Code to Language at Test Time: Evolving Solutions for ARC-AGI v1 and ARC-AGI-2 (2024 → 2025)

Jeremy Berman

October 23, 2025

Abstract

We present an evolutionary test-time compute system that achieved state-of-the-art results on ARC-AGI across two distinct representations. In 2024, our system evolved Python transform functions and reached **53.6%** on ARC-AGI-Pub. In 2025, we replaced code with natural-language instructions evaluated by sub-agents, achieving **79.6%** on ARC-AGI v1 at **$8.42/**task and a new SoTA of **29.4%** on ARC-AGI v2. We analyze why multi-generation search, targeted revisions, and representation choice matter; report failure modes of pooled context; and release implementation details enabling faithful reproduction.

1 Introduction

The ARC-AGI benchmark probes core abstraction and reasoning skills via small grid-transform tasks with few demonstrations. Performance hinges on *searching* for and *verifying* candidate programs or instructions that generalize from examples. In 2024, I topped the ARC-AGI Public leaderboard with an approach that evolves *code*; in 2025 I returned with a system that evolves *language* (natural-language instructions), once again reaching the top of the leaderboards while sharply improving price-performance.

This paper documents the key ideas, engineering, and ablations behind both systems, and explains the central design decision that unlocked further progress: **switching the solution representation from executable Python to English instructions** while keeping the same evolutionary chassis of generation → scoring → selection → revision.

Contributions

- A comparative analysis of two winning systems that share an evolutionary test-time compute backbone but differ in representation (code vs. language).
- A cost-efficient recipe that attains **79.6%** ARC-AGI v1 at $**8.42/**task and **29.4%** on ARC-AGI v2, including practical reliability tooling (retries, logging, model routing).
- Empirical study of *individual* vs. *pooled* revision strategies, and when pooled contexts begin to hurt due to token budget and attention dilution.

ARC Prize 2025 context ARC-AGI-2 replaces ARC-AGI-1 for the 2025 competition and uses a pass@2 metric over 120 tasks per eval split [1]. The official leaderboard emphasizes the *cost vs. accuracy* frontier [2].

[1]See `https://arcprize.org/competitions/2025/` and `https://arcprize.org/blog/announcing-arc-agi-2-and-arc-prize-2025`.

[2]`https://arcprize.org/leaderboard`.

2 Background and Related Work

ARC-AGI. The benchmark family (ARC-AGI-1, ARC-AGI-2) targets reasoning under few examples and penalizes rote memorization. Solutions span discrete program search, neuro-symbolic hybrids, LLM agents with verifiers, and evolutionary search.

Prior systems. Notable publicized results include: Greenblatt et al. (43% ARC-AGI-Pub, 2024), various commercial model previews (e.g., O3) reported at 75.7% on ARC-AGI v1 but at high cost ($\tilde{2}00$/task), and many neurosymbolic/program-synthesis entries. My 2024 system set a new ARC-AGI-Pub high-water mark via evolutionary test-time compute over Python transforms; the 2025 system extends that idea with language instructions.

3 Method I (2024): Evolving Code

Representation. Candidate solutions are Python grid-transform functions.

Loop. *Generate* diverse programs $\rightarrow$ *score* on train examples $\rightarrow$ *select* top-k $\rightarrow$ *revise* via LLM- or heuristic-guided edits. Depth (number of generations) systematically improved pass rates.

Verifier. Execution produces exact grids; scoring is deterministic; tie-break via simpler programs and lower token/compute.

4 Method II (2025): Evolving Language

Representation. Replace code with short, structured *English instructions* that describe the transformation. A lightweight sub-agent converts instructions to candidate outputs and self-checks against demonstrations.

Two-stage revision. (1) *Individual revisions:* each candidate is improved in isolation; (2) *Pooled revisions:* a curated subset of high-scorers is merged into a shared prompt for cross-pollination. We cap pooled parents to prevent context bloat.

Why language? Many ARC-AGI-2 patterns are brittle to express as rigid code but natural in controlled prose (e.g., "*flood-fill the largest green region then outline it in red*"). Language also reduces execution errors and broadens search neighborhoods between generations.

5 Engineering for Scale and Cost

Model routing. A small model-catalog maps tasks to providers/models (instruction synthesis vs. evaluation vs. scoring), enabling price/perf trade-offs.

Reliability. Exponential backoff and constrained concurrency ensure progress under API flakiness. Every run/task emits structured logs (`run_id`, `task_id`, model, prompts, scores) for auditability and reproduction.

Budgeting. Token-level cost accounting and early-stopping heuristics (e.g., stop after two perfect training matches) deliver strong accuracy at $ 8.42/task.

6 Experiments

6.1 Ablations (2024)

Depth helps: deeper evolutionary chains improve accuracy over shallow sampling. We also observe diminishing returns beyond a small number of revisions per generation.

6.2 Individual vs. Pooled Revision (2025)

Pooled prompts accelerate cross-seed idea transfer but can degrade when too many parents inflate context. In practice, a pool size of 2–3 per round worked well with modern context windows.

6.3 Budget Sensitivity

Accuracy saturates quickly with $\sim$40 attempts per task (30 initial, 5 individual revisions, 5 pooled), suggesting most value comes from early exploration plus targeted refinements.

7 Leaderboard and Head-to-Head

The official ARC Prize leaderboard visualizes the cost-accuracy frontier for ARC-AGI-2 [3]. Table 1 summarizes representative publicized results alongside our two systems. (Values for commercial previews are reported figures; ARC-AGI-2 scores are pass@2.)

System	Rep.	Accuracy	Cost / Task	Notes
Greenblatt (2024)	Code	43% v1	–	Prior public SOTA
O3 (preview, 2024)	–	75.7% v1	approx. $200	High cost
Berman (2024)	Code	**53.6% v1**	–	Evolutionary TTC
Berman (2025)	Lang	**79.6% v1**	**$8.42**	New SoTA: 29.4% v2

Table 1: Head-to-head summary across ARC-AGI variants.

8 Figures

Missing figure: figs/v1-architecture.png

Figure 1: V1 (2024): Evolutionary search over Python transforms: generate $\rightarrow$ score $\rightarrow$ select $\rightarrow$ revise.

9 Limitations

Verifier leakage. Instruction followers can overfit to demos; we mitigate with stricter self-checks and holdout-style scoring.

[3] https://arcprize.org/leaderboard

Missing figure: figs/v2-architecture.png

Figure 2: V2 (2025): Language-instruction evolution with individual and pooled revisions. Worst-case ∼40 attempts per task.

Missing figure: figs/depth-helps.png

Figure 3: Depth ablation (2024): deeper generations outperform shallow sampling.

Context brittleness. Pooled prompts suffer from token-limit effects; we cap pool sizes and prefer iterative refinement.

Distribution shift. Some ARC-AGI-2 patterns remain dead zones for current models; future work explores curriculum and tool-augmented verifiers.

10 Reproducibility Checklist

- **Code release**: Public repos for 2024 and 2025 systems (links in footnotes).
- **Seeds**: Fixed RNG seeds for sampling and selection.
- **Prompts**: Full prompts and instruction templates.
- **Attempts**: 30 initial $+5$ individual $+5$ pooled (caps).
- **Models**: Model catalog with provider and price info.
- **Logs**: Run/task IDs, scores, chosen candidates, costs.

Missing figure: figs/pooling-tradeoff.png

Figure 4: When pooling hurts: larger parent pools inflate context and reduce marginal gains.

- **Hardware**: Cloud inference only; no custom accelerators.
- **Costs**: Token-based accounting; average \$8.42/task on v1.

11 ARC Prize 2025 Paper Award Compliance

This paper is linked to an eligible Kaggle code submission and open-source release, per ARC Prize 2025 rules (ARC-AGI-2, 120-task evals, pass@2, open source before private eval) [4]. We include system description, empirical results, ablations, limitations, and reproducibility details as requested for Paper Award consideration.

12 Discussion: From Test-Time Search to General Reasoning

Test-time compute bridges static pattern-matching and systematic reasoning by treating inference as controlled search guided by verifiers. Switching from code to language broadened the reachable neighborhoods during search without sacrificing verifiability. We conjecture that combining compact instruction languages, stronger verifiers, and tool access will continue to push the frontier on ARC-AGI-2 and beyond.

Artifacts Blog write-ups (2024, 2025), full code, prompts, and run logs are linked in the project repository and supplemental material.

Acknowledgments. Thanks to the ARC Prize organizers and open-source contributors who made the evaluation pipeline and datasets accessible.

[4] https://arcprize.org/competitions/2025/ and Kaggle overview: https://www.kaggle.com/competitions/arc-prize-2025/overview