
PotARCin: Multi-Dimensional Evaluation of Skill Acquisition in Abstract Reasoning Tasks

Claas Beger
 Santa Fe Institute
 claasbeger@santafe.edu

Ryan Yi
 Santa Fe Institute
 ryi@santafe.edu

Melanie Mitchell
 Santa Fe Institute
 mm@santafe.edu

Abstract

The Abstraction and Reasoning Corpus (ARC) has become a prominent benchmark for evaluating general abstract reasoning and fluid intelligence in AI models. Yet standard ARC evaluation considers only a single capability: producing the correct output grid for a test input. We argue that this narrow format fails to evaluate the diversity of abilities that genuine abstract skill acquisition should enable. We introduce PotARCin, a benchmark that extends ARC by assessing understanding of a task’s underlying abstract rule across five dimensions: Definition, Classification, Constrained Generation, Editing, and Inversion. PotARCin employs programmatic methods to generate new task instances and transform given inputs for a given ARC task, enabling dynamic generative sampling beyond fixed input-output pairs. Across five state-of-the-art models evaluated on the ARC-AGI-1 training set, we observe a 25–52 percentage-point performance gap between standard ARC evaluation and evaluation on PotARCin, and find that multi-dimensional evaluation reorders models that standard accuracy ranks alike. We further investigate effects of generative sampling, difficulty of corruption types, and questions of self-consistency, showing that models frequently contradict their own formalized rule even where they have stated it correctly. We also introduce P-ARC, a held-out hand-crafted test set, on which models achieve 1–8% accuracy across all five dimensions, underscoring the importance of more holistic evaluations of abstract reasoning capabilities.

1 Introduction

The Abstraction and Reasoning Corpus (ARC) Chollet [2019] has become a central benchmark for abstract reasoning and fluid intelligence in modern AI models. Given only a few demonstrations of grid transformations, ARC requires a solver to infer an underlying transformation rule and apply it to a novel test input grid to produce the corresponding output grid. For several years, ARC-AGI-1 [Chollet, 2026] posed a substantial challenge for state-of-the-art models. However, recent progress on AI reasoning models has led to the near-saturation of ARC-AGI-1 evaluation, motivating the release of follow-up benchmarks such as ARC-AGI-2 [Chollet et al., 2026] and ARC-AGI-3 [Foundation, 2026]. In this work, we nevertheless focus primarily on ARC-AGI-1. Rather than proposing harder ARC-style tasks, we ask whether high performance on the original output-generation task reflects the robust competence the benchmark is intended to measure.

We use “skill acquisition” in the sense of Chollet [2019]: intelligence as the efficiency with which a system acquires a new skill from limited task-specific experience. In ARC, the demonstrations *are* that experience and the skill is the transformation rule inferred from them, so the term refers to this within-task inference process, not to longitudinal training or continual learning. PotARCin asks whether the inferred rule has been acquired as a *reusable* skill, usable across several closely related applications.

1.1 Background and Related Work

The Understanding Gap. A growing body of work argues that high benchmark accuracy does not necessarily imply robust understanding of the underlying concept, rule, or skill a benchmark supposedly tests [Ribeiro et al., 2020, Mineault et al., 2026]. This concern is especially important for ARC, where the standard evaluation asks only whether a model produces the correct output grid. Recent work on an ARC-like benchmark called “ConceptARC” [Moskvichev et al., 2023] shows that output accuracy can obscure whether models infer the intended abstractions, or whether they instead rely on unintended shortcuts or identify plausible rules they fail to execute [Beger et al., 2026]. A complementary perspective comes from the notion of “Potemkin understanding” introduced by Mancoridis et al. [2025]. They argue that benchmark success only supports claims about conceptual understanding under the implicit assumption that when models “misunderstand” concepts, they misunderstand in ways similar to humans. When this assumption fails, a model may answer benchmark questions correctly while still failing closely related probes of the same concept—probes that a human who understood the original question would be expected to answer correctly. Their benchmark operationalizes this idea by testing whether models that can define a concept can also use it in closely related classification, constrained generation, and editing tasks [Mancoridis et al., 2025]. We translate this multi-dimensional task framework to the ARC domain, replacing natural-language concepts with task-specific abstract transformation rules.

Generative and Multi-Dimensional Evaluation. These concerns are also aligned with broader critiques of evaluations using aggregate accuracy on static, held-out benchmarks. CheckList, for example, proposes behavioral testing through probes specific to a particular capability or test type rather than an undifferentiated measure of overall performance [Ribeiro et al., 2020]; BIG-bench evaluates models over highly heterogeneous task families [Srivastava et al., 2023]; and GSM-Symbolic uses symbolic templates to generate controlled variants of mathematical reasoning problems, revealing brittleness to small variations in problem instantiation [Mirzadeh et al., 2025]. Other recent benchmarks similarly move toward multi-dimensional evaluation of agent behavior, for example by measuring software-engineering performance through more fine-grained capabilities such as bug fixing, test generation, code-reviewing, and style fixing [Sonwane et al., 2026].

ARC and Program Synthesis. ARC has also been studied through the lens of program synthesis. Some methods attempt to solve ARC tasks by inducing programs that implement the inferred transformation rule [Li et al., 2024, Pourcel et al., 2026], while others use programs to generate procedural variants of ARC-like tasks [Hodel, 2024, Moffitt, 2025].

Together, these lines of work motivate a benchmark that is both generative and multi-dimensional. PotARCI brings this perspective to ARC by using explicit generator and verifier programs to define the space of valid input-output pairs and to sample novel instances beyond the fixed demonstrations. This generative structure allows us to evaluate whether a model has acquired the underlying transformation rule as a reusable abstraction rather than simply solved a single held-out input. We then test this rule-abstraction competence across five dimensions—definition, classification, constrained generation, editing, and inversion—in the style of Mancoridis et al. [2025].

1.2 Motivation and Contributions

Following this perspective, we view each ARC task as defining a small, task-specific skill. This skill is not merely the ability to map a particular input grid to its corresponding output grid, but the ability to infer and use the transformation rule underlying the task. Standard ARC evaluation tests one use of this skill, namely, forward transformation: given a new input grid, the model must produce the corresponding output grid. However, when a human has robustly acquired such a skill, we expect them to be able to use it in other closely related settings, such as formally defining the rule, repairing examples that violate it, or distinguishing valid from invalid input-output pairs. Failures on these related operations therefore reveal gaps in skill competence that output accuracy alone would not reveal. Such gaps are particularly important when considering the deployment of more complex skills in real-world settings, where failures may be less visible and more consequential.

PotARCI aims to measure these gaps, and is named to reflect the connection to [Mancoridis et al., 2025]. Inspired by the Potemkin Understanding framework, this benchmark is designed to expose cases in which a model’s accuracy on ARC tasks gives the appearance of rule

understanding without the corresponding underlying competence. Overall, we make the following **contributions**:

- **PotARCI**n, a benchmark evaluating ARC rule competence across five dimensions, with five procedures for generating plausible corrupted pairs from task-specific generator and verifier programs.
- **P-ARC**, a held-out set of 50 hand-crafted ARC-style tasks with generators, verifiers, and human-generated corruptions, enabling evaluation outside the public ARC-AGI-1 distribution.
- An evaluation of **five frontier models** showing large drops from output-grid accuracy to multi-dimensional competence, including a ranking reversal in which the weakest model by output-grid accuracy is not the weakest by rule competence.
- Two **targeted diagnostics** separating the mechanical cost of conjoining five dimensions from genuine inconsistency: a matched-budget comparison of cross- versus within-dimensional probing, and a self-consistency analysis conditioned on a correct executable Definition.

2 Methodology

Generators and Verifiers. PotARCI

n relies on two task-specific program types: generators and verifiers. A *generator* samples new input-output pairs for a given ARC task, thereby defining a distribution of valid task instances. A *verifier* implements the task transformation by mapping a candidate input grid to its corresponding output grid, providing an automated way to evaluate whether proposed outputs satisfy the underlying transformation rule.

Multiple works have proposed generator programs for ARC-AGI-1 tasks that we can draw upon. RE-ARC [Hodel, 2024] introduces a domain-specific language for constructing functions that can generate diverse new input-output pairs. ARC-GEN [Moffitt, 2025], by contrast, emphasizes what the authors call “mimetic similarity” in the design of its generators, requiring generated examples to closely preserve the constraints and distributional properties of the original demonstrations. This distinction is important for our setting: a generator may be logically consistent with the original demonstrations, yet shift the rule distribution or broaden its scope significantly enough that a solver could not reasonably infer the rule from the demonstrations alone. For PotARCI

n, we therefore require generated examples to be not only valid under a task rule, but also sufficiently mimetic with respect to the original demonstrations.

To illustrate, consider ARC-AGI-1 task a85d4709, whose demonstrations are shown within the dotted outline in Figure 1. While ARC-GEN preserves the task’s demonstrated 3×3 structure with one gray cell per row, in which gray-cell position determines row color, RE-ARC instead generalizes to variable-width grids by dividing rows into three regions and recoloring according to the region containing an outlier cell. This rule is consistent with the demonstrations, but significantly changes the effective task distribution. Without seeing these broader generated examples, a solver would have little reason to infer such a rule from the original ARC task alone. We illustrate this distinction in Figure 1 and will show its significance for enabling the Definition dimension in the Results section.

In addition to generators, PotARCI

n requires task-specific verifiers, programs that, given an input grid, generate the output grid according to the ground-truth rule governing the task. RE-ARC provides verifiers for the ARC-AGI-1 training corpus [Hodel, 2024]. ARC-GEN does not provide verifiers directly, but its associated Kaggle code-golf competition collected short verifier programs for ARC-AGI-1 training tasks [Moffitt et al., 2025]. We use verifier programs from the first-, third-, and fifth-place entries in that competition, together with the RE-ARC verifiers.

To ensure correctness, we validate candidate verifiers against multiple sources of task instances: the original ARC-AGI-1 train and test examples, the ARC-GEN stable dataset (average of 250 input-output pairs per task), and 50 random samples from the corresponding generators. Verifiers are considered valid only if they produce the expected output grid for all tested examples. Across all available verifier sources, we obtain at least one valid verifier for 394 of the 400 tasks in the ARC-AGI-1 training set; we manually implemented verifiers for the remaining six tasks and validated them using the same process.

Evaluation Dimensions. Using these generator and verifier programs, we define five evaluation dimensions for each ARC task. Four of these dimensions are modified versions of the framework from Mancoridis et al. [2025]; the fifth, inversion, is specific to the ARC domain. Given the demonstrations for an ARC task, a model is evaluated along the following five dimensions:

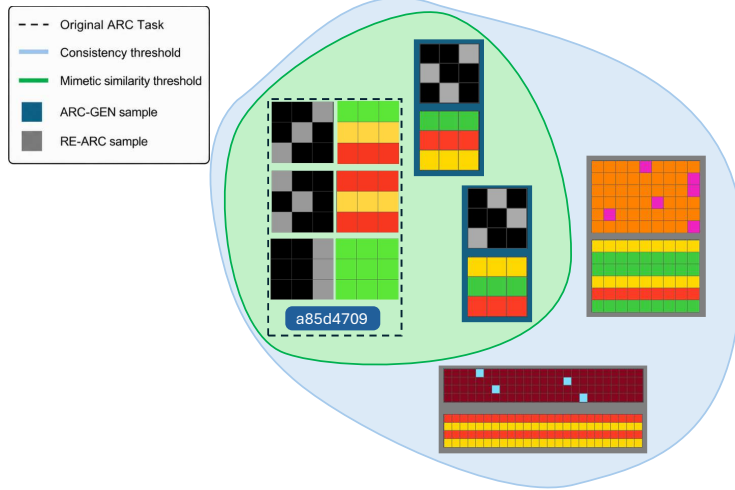


Figure 1: Illustration of consistency and mimetic similarity boundaries for ARC-AGI-1 task a85d4709.

- **Definition:** The model must write a general-purpose Python program that implements the inferred transformation rule, analogous to a verifier. We measure the accuracy of this program on the task demonstrations, the original test input, the ARC-GEN stable dataset, and 50 samples from the underlying generator, where accuracy is defined as producing the correct output grid. Non-executable programs are rare and account for little of this dimension’s error (Appendix D).
- **Classification:** The model is shown five candidate input-output pairs (in addition to the original demonstrations) and must decide whether each pair follows the same transformation rule as the demonstrations. Candidate pairs include both valid pairs sampled from the task generator and invalid pairs produced using the corruption procedures described below. Classification accuracy is strict at the task level: all five candidate judgments must be correct for the dimension to be counted as passed.
- **Constrained Generation:** The model must generate a novel input-output pair that demonstrates the transformation rule it has inferred from the demonstrations. We evaluate the proposed pair by applying the task verifier to the model’s input grid and checking whether the verifier output matches the model’s proposed output grid.
- **Editing:** The model is given a corrupted input-output pair and must repair it while remaining close to the corrupted pair under a normalized edit-distance constraint.¹ The repaired pair is accepted only if it satisfies this constraint and the task verifier maps the edited input grid to the edited output grid.
- **Inversion:** The model is given an output grid and must infer a possible input grid that would produce it under the ground-truth task rule. We evaluate the proposed input by applying the task verifier to it and checking whether the resulting output grid matches the provided output grid.

Taken together, these five dimensions allow us to probe model understanding of ARC tasks from several complementary perspectives without substantially changing the underlying inference framework. In each dimension, the model is given the same task demonstrations and must reason about the same latent transformation rule, but is evaluated on different uses of that rule. Failures on these additional dimensions can reveal shallow, brittle, or unintended forms of rule understanding that standard output-grid accuracy alone may not capture [Beger et al., 2026, Mancoridis et al., 2025, Mineault et al., 2026]. Because all five dimensions are evaluated using explicit generators and verifiers, PotARCin serves to broaden ARC evaluation while still preserving fully automated scoring. One caveat applies throughout: they differ in *verification burden*. Definition is checked against many

¹We use a normalized Hamming distance with padding. We align the corrupted grid and the model-predicted grid on a shared padded bounding box with height $h = \max(h_a, h_b)$ and width $w = \max(w_a, w_b)$, using fill value -1. The distance is the number of differing cells, normalized by $h \cdot w$. This distance is computed separately for the input and output grids; both must be at most 0.70. A similar measure is applied to ensure distance from training examples for constrained generation.

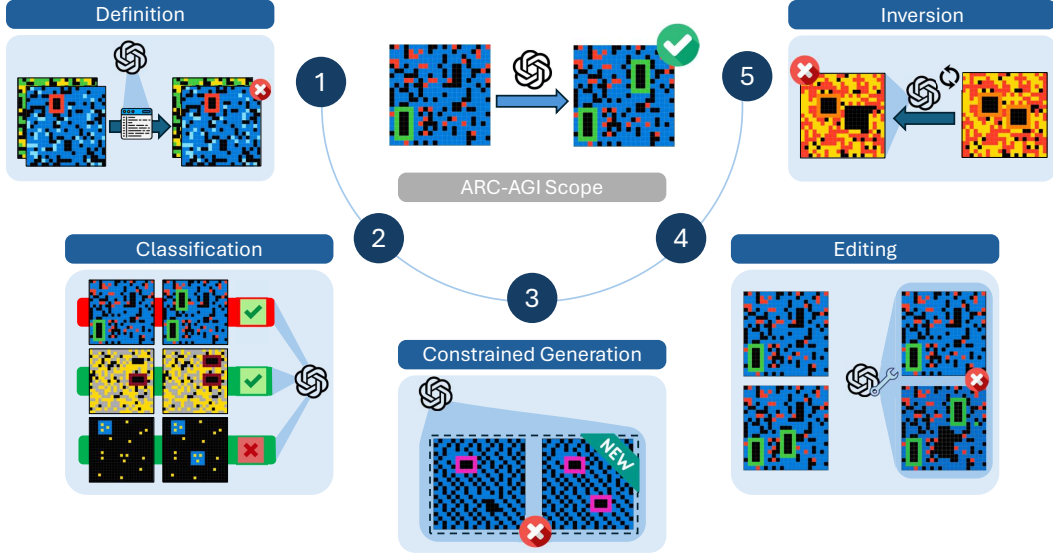


Figure 2: Overview of the PotARCIn benchmark. The shown ARC task is 890034e9, for which GPT-5.4 correctly solved the standard test-grid transformation task but failed all additional dimensions.

sampled instances, while Inversion is checked against only one. We treat the many Definition samples as a single evaluation instance, used to confirm that the program captures the rule broadly rather than overfit to a narrow set of examples, but cross-dimension difficulty comparisons should be read with this asymmetry in mind.

Corruption Types. To evaluate Classification and Editing, PotARCIn requires invalid input-output pairs that are still close enough to the task distribution to be challenging. We therefore construct corrupted pairs using five complementary procedures. Each procedure starts from either a valid task instance or a related retrieved example and produces a candidate invalid pair. We then apply the task verifier to ensure that the corrupted pair does not satisfy the target transformation rule. Examples of all corruption types are provided in Appendix A.

- **Instance Mismatch:** We pair an input grid with an output grid from a different valid instance of the same task. To make the mismatch plausible, we use an edit-distance heuristic to search the ARC-GEN stable dataset and the original training examples for outputs similar to the correct output, then sample from the top three ranked candidates.
- **Retrieval:** We retrieve input-output pairs from other ARC tasks with similar “task embeddings”. To construct these embeddings, we adapt the V-ARC architecture [Hu et al., 2025], which combines a vision transformer backbone with an explicit task embedding vector intended to encode the underlying task rule. More specifically, we adopt the approach of Deliège et al. [2026], who freeze the transformer backbone and tune only the task embedding. We then build a vector database and use cosine similarity to retrieve pairs from related but distinct transformation rules.
- **H-ARC:** We use incorrect, human-generated output grids from the H-ARC dataset [LeGris et al., 2025]. As one of the largest-scale studies on human ARC performance, H-ARC provides a sizable dataset of human-generated answers, including many erroneous solutions. After filtering empty solutions, 374 ARC-AGI-1 training tasks have at least one incorrectly constructed output available, with an average of 8.36 per task. These errors provide naturally occurring invalid outputs that arise from human attempts to solve the same tasks.
- **Corrupted Verifier:** We generate corrupted outputs by mutating verifier programs. For RE-ARC verifiers, which are written as sequences of DSL assignments, we remove a small number of intermediate assignments and reconnect the remaining program. For other verifier implementations, we construct the abstract syntax tree and perform a binary-operation prune, replacing a binary expression with its left subtree. Applying these corrupted verifiers to valid inputs yields outputs produced by a perturbed version of the task rule.

- **Color Flip:** We locally perturb a valid generated pair by sampling a cell from either the input or output grid and reassigning the color of that cell, together with orthogonally connected cells of the same color, to a new color. To avoid trivial background changes, we bias the initial cell selection away from black cells.

At evaluation time, we sample from a fixed mixture over valid pairs and the five corruption types; the exact weights, which differ between datasets to reflect differences in corruption quality, are given in Table 2. Because ARC tasks are generally underspecified, these labels are defined relative to the designated ground-truth rule instantiated by the task generator and verifier; a pair labeled as corrupted may still be compatible with some alternative rule that also explains the demonstrations. For Editing, we sample a single invalid pair from the latter three types.

Evaluation Protocol. In our evaluation, each of the 400 ARC-AGI-1 training tasks is evaluated over the five dimensions described above, as well as for the original output-grid generation skill. Each skill is evaluated independently, in a new context window. ARC performance is commonly reported using pass@2, where a task is counted as solved if either of two submitted output grids exactly matches the ground truth [Chollet et al., 2025]. We do not adopt pass@2 as our primary metric, both because it does not translate cleanly across all PotARCin dimensions and because our goal is to evaluate rule competence across multiple uses of the same inferred transformation. Instead, we report *full-task accuracy*: the percentage of tasks for which a model passes all five PotARCin dimensions under a given sampling run.

Because full-task accuracy is a coverage metric over five dimensions, we report alongside it the *independence baseline*: the product of the five marginal accuracies, i.e. the joint rate expected if per-dimension pass events were independent. All runs are repeated twice, and we report pooled Wilson 95% confidence intervals over the $n=800$ (ARC-AGI-1) or $n=100$ (P-ARC) task evaluations rather than standard deviations over two runs, which are too few to estimate run-to-run variance and can suggest perfect stability when two runs happen to yield equal counts. Equal counts also hide churn in *which* tasks pass: GPT-5.4 passes 185 and 188 ARC-AGI-1 tasks across its runs, intersecting on only 146 (Jaccard 0.64). Full-task pass-set overlap ranges from 0.49 (Kimi K2.5) to 0.79 (Gemini 3.1 Pro) across models; per-model counts are in Table 7.

We evaluate five frontier models with strong ARC-AGI-1 performance: GPT-5.4, Gemini 3.1 Pro, Claude Opus 4.6, Kimi K2.5, and MiniMax M2.5. Because PotARCin multiplies the number of evaluations per ARC task across dimensions and samples, we generally use low reasoning-effort settings unless otherwise noted; Claude receives a 120K thinking budget, as in prior ARC evaluations. We use the public ARC-AGI-1 training set as a controlled and favorable evaluation setting, since it is accessible, widely studied, and likely easier than held-out ARC-AGI-1 splits [Chollet et al., 2025, LeGris et al., 2025]. Results on ARC-AGI-1 training tasks should therefore be interpreted with caution: o3, GPT-5.4’s predecessor, and Claude Opus 4.6 have been reported to have trained on ARC-AGI-1 training data [Foundation, 2025, Anthropic]. Other frontier models likely have had similar exposure. Even in this favorable setting, PotARCin exposes substantial drops relative to standard output-grid accuracy.

P-ARC Test Set. To estimate performance on unseen tasks, we construct a new held-out ARC-style test set, P-ARC, which will be released upon publication. P-ARC consists of 50 hand-crafted tasks, each with corresponding generator and verifier programs. Because H-ARC errors are unavailable for new tasks, we collect three erroneous human output grids per task as corruptions. Our human protocol is a *feasibility check* embedded in the design process, not a formal study with naive participants: each task was shown to two or three team members other than its author and retained only if at least one produced the correct output, typically within one or two attempts; solve times were not recorded. For every task, a member other than the author inspected at least 50 generator-produced examples and confirmed they realize the intended rule. Taxonomy and further calibration details are in Appendix E. Although precise difficulty is challenging to quantify, we qualitatively estimate P-ARC to lie between ARC-AGI-1 and ARC-AGI-2 in difficulty.

3 Results

We report three levels of performance in Figure 3: standard output-grid correctness, dimension-specific accuracy, and *full-task accuracy*, defined as the percentage of tasks for which a model passes all five PotARCin dimensions in a given sample. Results are shown for both the ARC-AGI-1 training

set and the held-out P-ARC test set. Bars show pooled rates over $n=800$ ARC-AGI-1 or $n=100$ P-ARC task evaluations, with pooled Wilson 95% intervals. Exact values and the independence baseline are reported in Table 3.

ARC-AGI-1. On the standard output-grid task, the proprietary models achieve 84–89% accuracy on the ARC-AGI-1 training set, while MiniMax reaches 72.9% and Kimi 63.4%. Full-task accuracy, requiring success across all five PotARCin dimensions, is only 21–58%, a drop of 25–52 percentage points relative to output-grid accuracy. Definition and Classification are consistently the hardest and Inversion the easiest, subject to the verification-burden asymmetry noted in the Methodology section.

The multi-dimensional evaluation also reveals differences between models that are largely hidden by output-grid accuracy alone. Gemini and Claude achieve similar performance on the standard ARC output-grid-generation, but Gemini outperforms Claude by 20.6 percentage points in full-task accuracy. More strikingly, Kimi K2.5 has the lowest output-grid accuracy of the five models (63.4%) yet nearly doubles MiniMax’s full-task accuracy (38.6% versus 20.6%) while scoring 9.5 points lower on output grids. Kimi also matches Claude’s full-task accuracy (37.6%, with overlapping intervals) despite trailing it by 24.2 points on the standard task. Across individual dimensions, proprietary models often achieve performance closer to their standard output-grid accuracy, but difficulty varies by dimension: Definition and Classification tend to be more challenging, while Editing and Inversion are comparatively easier. We also observe the importance of mimetic similarity in the Definition dimension. When candidate programs are instead evaluated on the broader RE-ARC-generated distribution, overall Definition accuracy drops to roughly 3%, supporting our prior concern that inferred programs do not generalize to distributional shifts that are too wide to easily infer from the original demonstrations alone.

P-ARC. On the held-out P-ARC test set, output-grid accuracy is lower overall and performance gaps between models are larger. Full-task accuracy falls to 1–8% for every model, indicating the difficulty of acquiring robust multi-dimensional competence on unseen tasks. The ordering across dimensions matches ARC-AGI-1: Definition is hardest for all five models, at 6–25%, while Inversion is easiest, at 29–71%. Claude is strongest on five of the six reported measures (with GPT-5.4 leading on Classification), although this performance is accompanied by a marked increase in token consumption despite low reasoning effort (see Appendix F). Overall, PotARCin exposes substantial gaps in skill competence when models are evaluated on newly constructed tasks outside the public ARC-AGI-1 training distribution.

As a stricter control on Constrained Generation, we exclude exact matches to examples in the ARC-GEN stable set. This lowers ARC-AGI-1 Constrained Generation accuracy to 70.0% for GPT-5.4 (−7.2), 80.4% for Gemini 3.1 Pro (−7.6), 73.5% for Claude Opus 4.6 (−5.4), 67.6% for Kimi K2.5 (−4.1), and 61.2% for MiniMax M2.5 (−3.4). These matches therefore account for only a limited share of successes; we report the stricter values as a conservative reference.

3.1 Isolating Cross-Dimensional Failures

Part of the drop is arithmetic, since full-task accuracy conjoins five dimensions. Observed full-task accuracy nevertheless *exceeds* the independence baseline of Table 3 for every model and both datasets, by +7.4 to +18.5 points on ARC-AGI-1, so per-dimension pass events are positively correlated rather than independent, as expected when tasks vary in difficulty: easy tasks tend to be passed across dimensions and hard ones failed across them. Marginal independence is therefore a lower bound on the joint rate. The analyses in this section isolate the component of multi-dimensional failure that repeated probing in a single format would not expose.

The sharper question is whether cross-dimensional probing surfaces failures that repeated probing within one dimension would miss. We test this at a *matched evaluation budget* on the 20 ARC-AGI-1 tasks GPT-5.4 initially passed across all five dimensions, using ten same-seed repeats per task. Evaluating five different dimensions once exposes at least one failure in 25.5% of trials, versus 20.2% when a single typical dimension is sampled five times (all $\binom{10}{5}$ subsets; task-paired t -test $p=0.030$, Wilcoxon $p=0.028$, bootstrap 95% CI on the difference [+1.0, +10.5] pp). Two alternative aggregators over the repeats give 20.0% and 18.8%, both also significant.² Spending a

²Four of the five dimensions individually run in this direction; Constrained Generation is noisier under repetition and runs against it, and excluding it shrinks the comparison to 15.0% vs. 14.4%.

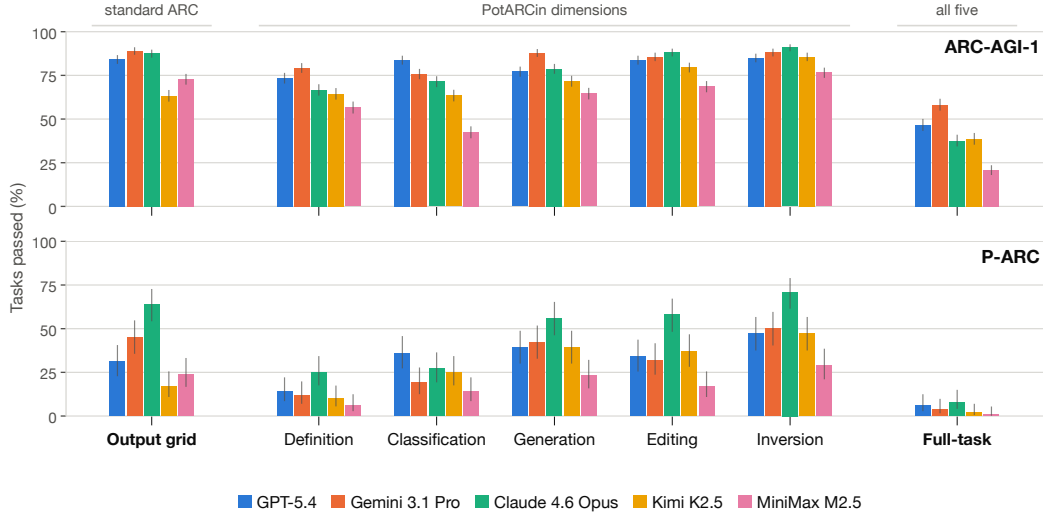


Figure 3: Performance across all seven measures for 400 ARC-AGI-1 training tasks and 50 P-ARC tasks, pooled over two independent runs. Bars are the pooled rate over $n=800$ (ARC-AGI-1) or $n=100$ (P-ARC) task evaluations; whiskers are pooled Wilson 95% intervals. Both panels share a common vertical scale. Exact values for every cell are given in Table 3.

fixed evaluation budget on diagnostic breadth therefore surfaces more failures than spending it on depth, which is the central practical argument for evaluating several uses of the same inferred rule.

3.2 Effects of Generative Sampling

A key advantage of PotARCin is that generator programs allow us to sample many distinct input/output pairs for the same underlying ARC task, conditioned on the same demonstrations. This lets us test whether apparent skill competence is robust across multiple instantiations, rather than only across repeated model calls with different stochastic outputs. In the main evaluation, each task is evaluated using a single sampled configuration for each dimension. Here, we ask whether tasks that appear solved under this limited sampling budget remain solved when evaluated more extensively.

Because this experiment is more resource-intensive, we restrict it to GPT-5.4 on a small subset of ARC-AGI-1 training tasks. We begin with tasks for which GPT-5.4 passes all five PotARCin dimensions under the main evaluation. From these, we select 50 random tasks and filter out cases where the generated Definition program implements only trivially simple operations, such as a single rotation, duplication, or basic color remapping, leaving 20 tasks for the increased-sampling study.

For each of these 20 tasks, we sample 10 distinct multi-dimensional evaluation configurations. Classification, Editing, and Inversion naturally support this procedure because each can be resampled by drawing new candidate pairs, corruptions, or target outputs. Definition and Constrained Generation require slight modifications, since repeated calls do not necessarily produce meaningfully different evaluation instances. For Definition, we do not ask the model to write a new program; instead, we increase the number of dynamic generator samples used to test the original Definition program, from 50 to 1,000. For Constrained Generation, we test whether the model can produce multiple distinct valid examples. Whenever the model generates a valid pair, we add it to the demonstrations for the next generation attempt and instruct the model not to copy or minimally alter previous examples.

Results are shown in Figure 4, with statistics for selected sampling budgets reported in Table 6. On 15 of the 20 tasks, GPT-5.4 fails at least one of the 10 sampled configurations, despite having passed all five dimensions in the main evaluation. Averaged across the selected tasks, full-task accuracy under increased sampling is 70%. These results suggest that single-sample full-task accuracy can still overestimate robust rule competence: a model may pass all five dimensions once, yet fail when the same rule is probed through additional valid instantiations. For Definition, increasing the number of dynamic test samples produces little additional change: even with up to 10,000 additional samples, the initial pass/fail outcome never changes. For the other resampled dimensions, failures tend to

emerge quickly, with 11 of the 15 affected tasks exhibiting at least one failure within the first two additional samples. One caveat is that this experiment uses GPT-5.4 with reasoning enabled, for which the OpenAI API does not expose the temperature parameter. Sampling stochasticity is therefore present, but cannot be separated from variation induced by resampling the task instance itself.

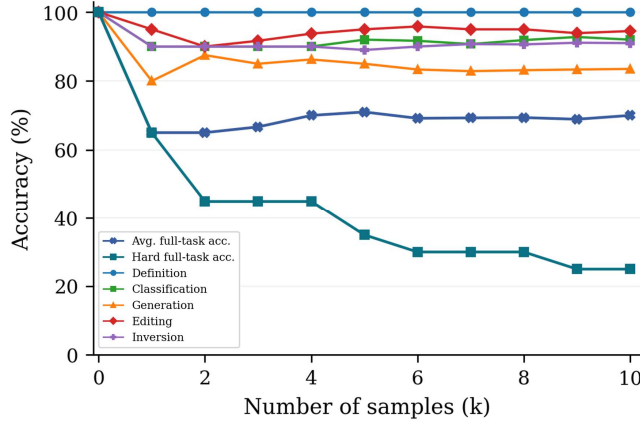


Figure 4: Results of an increased sampling budget across 20 ARC-AGI-1 training tasks that GPT-5.4 initially solved across all five PotARCin dimensions. “Hard full-task accuracy” is the percentage of tasks passing *all* sampled configurations up to that budget.

3.3 Difficulty of Corruption Types

To better understand the Classification dimension, we compute model failure rates separately for each corruption type on both evaluation datasets. Results are shown in Table 1. Each entry reports the fraction of examples of a given type that the model labels incorrectly.

Human-generated errors are, in general, the most difficult corrupted examples for models to classify. This is especially pronounced on P-ARC, where failure rates on human corruptions are substantially higher than for other corruption types. This pattern is intuitive: unlike synthetic perturbations such as color flips or mismatched instances, human errors often arise from attempts to apply a related but incorrect interpretation of the task rule, or from subtle mistakes that are difficult to detect. We also observe substantial differences between models, with GPT-5.4 achieving the lowest failure rate in 8 of the 12 corruption-type/dataset categories.

Across datasets, the relative difficulty of corruption types is broadly similar, but average failure rates are higher on P-ARC. This likely reflects both the greater difficulty of the held-out tasks and differences in corruption quality. In particular, P-ARC human corruptions were collected specifically for our tasks, whereas H-ARC likely includes many human outputs that are clearly incorrect. The retrieval corruption type may also be affected by distribution shift: because the retrieval database is built from ARC-AGI-1 tasks, retrieved examples may be less closely matched to P-ARC tasks than to public ARC-AGI-1 tasks.

Because Classification is scored strictly, it admits two trivial baselines worth stating. The candidate mixture is roughly 71% invalid, so always answering “invalid” scores 19.1% on ARC-AGI-1 and 16.0% on P-ARC, while always answering “valid” scores 0.3% and 2.0%. Every model clears the always-invalid baseline comfortably on ARC-AGI-1. On P-ARC it is closer: Gemini 3.1 Pro (19.0%) is barely above it and MiniMax M2.5 (14.0%) falls below, so P-ARC Classification accuracy should not be read as evidence of rule-consistent discrimination for the weaker models.

3.4 Self-Consistency Conditioned on a Correct Definition

Mancoridis et al. [2025] evaluate self-consistency by comparing model responses across related probes of the same concept and identifying cases where those responses disagree. PotARCin enables an analogous test for ARC. Because the Definition dimension asks models to produce an executable program, we can compare that program against the model’s responses in other dimensions: for Constrained Generation and Inversion, we apply the Definition program to the model-proposed input

Table 1: **Failure rates by corruption type and dataset**, pooled over both runs. Entries are percentages: the fraction of candidates of each type the model labelled incorrectly. For corrupted candidates, failure means accepting an invalid pair; for valid candidates (“Correct”) it means rejecting a valid pair. Bold marks the lowest rate in each column. The last row gives the median number of candidates of that type per model; counts vary by at most 1% across models. Wilson 95% intervals are in Table 8.

Model	ARC-AGI-1						P-ARC					
	Human	Corrupted verifier	Correct	Color flip	Retrieval	Instance mismatch	Human	Corrupted verifier	Correct	Color flip	Retrieval	Instance mismatch
Claude 4.6 Opus	20.74	11.14	1.84	8.76	4.52	4.59	65.03	12.50	4.93	28.57	7.14	12.22
GPT-5.4	8.21	2.30	5.06	6.53	3.15	1.85	47.22	0.00	16.67	0.00	0.00	3.26
Gemini 3.1 Pro	16.37	8.99	5.48	2.82	2.17	6.81	60.42	9.38	22.92	14.29	2.38	7.61
Kimi K2.5	18.54	11.24	5.98	25.78	10.79	7.75	61.11	14.06	12.59	42.86	14.29	11.83
MiniMax M2.5	31.32	26.62	3.66	40.23	40.43	18.12	70.83	23.44	12.50	50.00	30.95	23.91
<i>items per model</i>	562	432	1146	353	417	1086	144	64	144	14	42	92

grid and check whether it produces the corresponding output grid; for Classification, we compare the model’s valid/invalid judgment against whether the Definition program maps the candidate input to the candidate output.

To distinguish genuine cross-dimensional inconsistency from cases in which the model’s Definition is itself wrong, we additionally condition this analysis on a correct executable Definition. Here, a correct Definition is a loadable program that passes the task’s training and test examples. We then compare that program with Classification, Constrained Generation, and Inversion responses, pooling both runs for all five models. A full overview is shown in Table 4.

When both the Definition and the response in the other dimension are correct, pooled agreement is 99.1% on ARC-AGI-1 and 94.0% on P-ARC. In contrast, when the Definition is correct but the other response is incorrect, agreement falls to 21.1% on ARC-AGI-1 and 17.1% on P-ARC. Thus, most mistakes in the other dimensions cannot be explained by the model consistently applying its own correctly formalized rule. The effect is strongest for Classification and Constrained Generation, while Inversion varies more across models; the P-ARC estimates should be interpreted cautiously because the conditional denominators are smaller. Unconditional agreement rates, without conditioning on Definition correctness, are reported in Table 5.

These conditioned inconsistencies provide direct evidence of a Potemkin-like failure mode in ARC rule inference. We frequently observe responses that contradict a rule the same model has correctly formalized in another dimension. Unlike pre-existing concepts studied by Mancoridis et al. [2025], however, the object of understanding here is a task-specific skill: the transformation rule that the solver must infer from a small number of demonstrations.

4 Conclusion

We introduced PotARCI, a benchmark that extends ARC evaluation beyond output-grid correctness by probing five dimensions of rule competence: Definition, Classification, Constrained Generation, Editing, and Inversion. On ARC-AGI-1, full-task accuracy is 25–52 percentage points below output-grid accuracy, while on P-ARC, it falls to 1–8%. Increased generative sampling further shows that on 15 of 20 tasks GPT-5.4 initially passed across all five dimensions, at least one of ten resampled configurations fails. In addition, our diagnostic analyses reveal systematic differences across corruption types and dimensions, with human-generated errors being the most difficult corruptions for models to classify. Probing all five dimensions once exposes a failure in 25.5% of trials, compared with 20.2% when one dimension is sampled five times. Conditioning self-consistency on a correct executable Definition, pooled agreement is 99.1% on ARC-AGI-1 and 94.0% on P-ARC when the other response is correct, but only 21.1% and 17.1%, respectively, when the other response is incorrect. These contradictions point to a Potemkin-like failure mode in ARC rule inference, where models may correctly formalize a rule in one setting but fail to apply it consistently in another.

Overall, our results suggest that standard ARC accuracy provides an incomplete picture of abstract skill acquisition. Producing the correct output grid for a single test input does not necessarily imply that a model has acquired a reusable, flexible representation of the underlying transformation rule. Multi-dimensional evaluation also reorders models that standard scores rank alike, and the weakest model on output grids is not the weakest on broader rule competence. More generally, PotARCI

supports evaluating not only depth within a fixed task format, but also breadth across different uses of the same inferred rule. Even on ARC-AGI-1 training tasks, which are likely in some models’ training corpus, the observed drops in full-task accuracy and self-consistency suggest that ARC-AGI-1 should not be regarded as fully “solved” from the perspective of abstract skill acquisition. While newer benchmarks such as ARC-AGI-3 [Foundation, 2026] explore richer interactive settings, our results show that even simple static transformation rules remain challenging when models are asked to use them flexibly. By combining generator- and verifier-based evaluation with multiple rule-use dimensions, PotARCin offers a more demanding and more diagnostic framework for measuring abstract rule competence in ARC-style tasks.

Limitations and Future Work. PotARCin depends on the quality of its generator and verifier programs. Although these programs enable scalable automated evaluation, they must remain aligned with each task’s intended transformation rule. For P-ARC, we write and test these programs; for ARC-AGI-1, we reuse existing verifier sources and observe that some candidate verifiers capture nearby variants rather than the intended rule. Even with extensive testing, exact rule alignment is difficult to guarantee. A related, more fundamental issue concerns Classification: ARC tasks are underspecified and may admit multiple rules consistent with the demonstrations [Beger et al., 2026]. A pair labeled invalid under the designated verifier may therefore remain explicable under an alternative consistent rule. Classification should be interpreted as testing adherence to the benchmark’s intended rule, not as proving that no consistent alternative exists. Another limitation is model-compute comparability. Although all models are configured for low reasoning effort, Claude Opus 4.6 was observed to consume substantially more tokens than other models on P-ARC, which may affect comparisons; see also Appendix F. In addition, P-ARC’s estimated difficulty rests on an internal feasibility check rather than a formal human study; formal solve-rate and solve-time measurements are left to future work. Finally, several aspects of rule competence remain outside our scope. We restrict generated examples to the mimetic similarity boundary of the demonstrations, leaving principled rule shifts and conceptual slippage [Hofstadter and Mitchell, 2019] for future work. We also observe that models often exploit the Constrained Generation and Inversion dimensions by producing the simplest valid examples possible. Future work could quantify the diversity and expressiveness of model-generated examples, especially for fully autonomous construction of ARC-style tasks.

Acknowledgments and Disclosure of Funding

Sandia National Laboratories is a multimission laboratory managed and operated by National Technology and Engineering Solutions of Sandia, LLC, a wholly owned subsidiary of Honeywell International, Inc., for the U.S. Department of Energy’s National Nuclear Security Administration under contract DE-NA-0003525. C. Beger and R. Yi were supported in part through the BANYAN Institute, funded by Sandia National Laboratories’ Laboratory Directed Research and Development program. The authors would like to thank Marina Mancoridis for visiting the Santa Fe Institute and presenting her work on Potemkin Understanding, and Brenden Lake and Tom Griffiths for constructive discussions regarding the PotARCin experiments. We also thank Anna Patelli for supporting the development of the P-ARC test set and Navya Sahay for assistance with model evaluation.

References

- Anthropic. System card: Claude opus 4.6 february 2026 anthropic.com. URL <https://www-cdn.anthropic.com/Odd865075ad3132672ee0ab40b05a53f14cf5288.pdf>.
- Claas Beger, Ryan Yi, Shuhao Fu, Kaleda Denton, Arseny Moskvichev, Sarah W. Tsai, Sivasankaran Rajamanickam, and Melanie Mitchell. Do ai models perform human-like abstract reasoning across modalities?, 2026. URL <https://arxiv.org/abs/2510.02125>.
- Francois Chollet, Mike Knoop, Gregory Kamradt, and Bryan Landers. Arc prize 2024: Technical report, 2025. URL <https://arxiv.org/abs/2412.04604>.
- Francois Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. Arc-agi-2: A new challenge for frontier ai reasoning systems, 2026. URL <https://arxiv.org/abs/2505.11831>.
- François Chollet. On the measure of intelligence, 2019. URL <https://arxiv.org/abs/1911.01547>.
- François Chollet. The Abstraction and Reasoning Corpus (ARC), 2026. <https://github.com/fchollet/ARC>, last accessed, February 2026.
- Adrien Delière, Claas Beger, Marc Van Droogenbroeck, and Melanie Mitchell. Implicit rule induction with test-time task embeddings in arc-like tasks, 2026. URL <https://arxiv.org/abs/2609.21181>.
- ARC Prize Foundation. OpenAI o3 Breakthrough High Score on ARC-AGI-Pub | ARC Prize — arcprize.org. <https://arcprize.org/blog/oai-o3-pub-breakthrough>, 2025. [Accessed 06-05-2026].
- ARC Prize Foundation. Arc-agi-3: A new challenge for frontier agentic intelligence, 2026. URL <https://arxiv.org/abs/2603.24621>.
- Michael Hodel. Addressing the abstraction and reasoning corpus via procedural example generation, 2024. URL <https://arxiv.org/abs/2404.07353>.
- Douglas R Hofstadter and Melanie Mitchell. Conceptual slippage and analogy-making: a report on the copycat project. In *10th Annual Conference Cognitive Science Society Pod*, pages 601–607. Psychology Press, 2019.
- Keya Hu, Ali Cy, Linlu Qiu, Xiaoman Delores Ding, Runqian Wang, Yeyin Eva Zhu, Jacob Andreas, and Kaiming He. Arc is a vision problem!, 2025. URL <https://arxiv.org/abs/2511.14761>.
- Solim LeGris, Wai Keen Vong, Brenden M. Lake, and Todd M. Gureckis. A comprehensive behavioral dataset for the abstraction and reasoning corpus. *Scientific Data*, 12(1), August 2025. ISSN 2052-4463. doi: 10.1038/s41597-025-05687-1. URL <http://dx.doi.org/10.1038/s41597-025-05687-1>.
- Wen-Ding Li, Keya Hu, Carter Larsen, Yuqing Wu, Simon Alford, Caleb Woo, Spencer M Dunn, Hao Tang, Michelangelo Naim, Dat Nguyen, et al. Combining induction and transduction for abstract reasoning. *arXiv preprint arXiv:2411.02272*, 2024.
- Marina Mancoridis, Bec Weeks, Keyon Vafa, and Sendhil Mullainathan. Potemkin understanding in large language models. In Aarti Singh, Maryam Fazel, Daniel Hsu, Simon Lacoste-Julien, Felix Berkenkamp, Tegan Maharaj, Kiri Wagstaff, and Jerry Zhu, editors, *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pages 42857–42881. PMLR, 13–19 Jul 2025. URL <https://proceedings.mlr.press/v267/mancoridis25a.html>.
- Patrick J. Mineault, Thomas L. Griffiths, and Sean Escola. Cognitive dark matter: Measuring what ai misses, 2026. URL <https://arxiv.org/abs/2603.03414>.
- Iman Mirzadeh, Keivan Alizadeh, Hooman Shahrokhi, Oncel Tuzel, Samy Bengio, and Mehrdad Farajtabar. Gsm-symbolic: Understanding the limitations of mathematical reasoning in large language models, 2025. URL <https://arxiv.org/abs/2410.05229>.

- Michael D. Moffitt. Arc-gen: A mimetic procedural benchmark generator for the abstraction and reasoning corpus, 2025. URL <https://arxiv.org/abs/2511.00162>.
- Michael D. Moffitt, Divy Thakkar, Ryan Burnell, Orhan Firat, Walter Reade, Sohler Dane, and Addison Howard. Neurips 2025 - google code golf championship. <https://kaggle.com/competitions/google-code-golf-2025>, 2025. Kaggle.
- Arseny Moskvichev, Victor Vikram Odouard, and Melanie Mitchell. The ConceptARC benchmark: Evaluating understanding and generalization in the ARC domain. *Transactions on Machine Learning Research*, 2023.
- Julien Pourcel, Cédric Colas, and Pierre-Yves Oudeyer. Self-improving language models for evolutionary program synthesis: A case study on arc-agi, 2026. URL <https://arxiv.org/abs/2507.14172>.
- Marco Tulio Ribeiro, Tongshuang Wu, Carlos Guestrin, and Sameer Singh. Beyond accuracy: Behavioral testing of NLP models with CheckList. In Dan Jurafsky, Joyce Chai, Natalie Schluter, and Joel Tetreault, editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 4902–4912, Online, July 2020. Association for Computational Linguistics. doi: 10.18653/v1/2020.acl-main.442. URL <https://aclanthology.org/2020.acl-main.442/>.
- Atharv Sonwane, Eng-Shen Tu, Wei-Chung Lu, Claas Beger, Carter Larsen, Debjit Dhar, Simon Alford, Rachel Chen, Ronit Pattanayak, Tuan Anh Dang, Guohao Chen, Gloria Geng, Kevin Ellis, and Saikat Dutta. Omnicode: A benchmark for evaluating software engineering agents, 2026. URL <https://arxiv.org/abs/2602.02262>.
- Aarohi Srivastava, Abhinav Rastogi, Abhishek Rao, Abu Awal Md Shoeb, Abubakar Abid, Adam Fisch, Adam R. Brown, Adam Santoro, Aditya Gupta, Adrià Garriga-Alonso, et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models, 2023. URL <https://arxiv.org/abs/2206.04615>.

A Corruption Type Examples

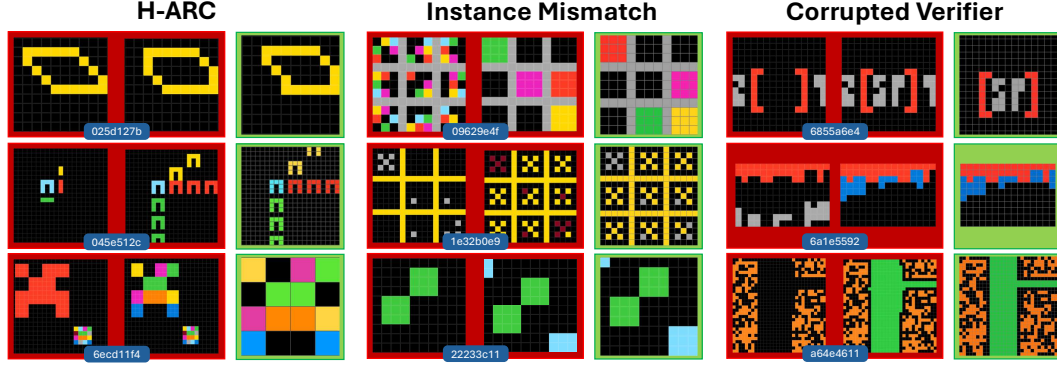


Figure 5: Examples of corruption types generated on the ARC-AGI-1 training set. Correct output grids are shown to the right of each corrupted pair, outlined in green. **Left:** H-ARC corruptions are errors made by human participants in the H-ARC dataset. **Middle:** Instance mismatch corruptions replace the correct output grid with a similar output from the generator dataset, selected using an edit-distance metric. **Right:** Corrupted verifiers produce outputs from perturbed verifier programs, either by deleting AST branches or reassigning intermediate variables.

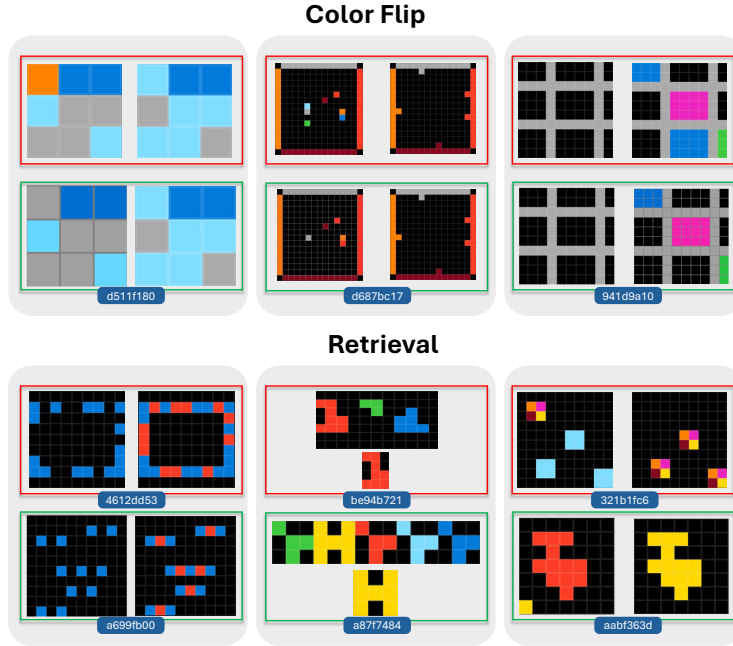


Figure 6: Examples of color flip and retrieval corruptions. **Top:** Color flip corruptions change the color of a sampled cell or small connected group in either the input or output grid. **Bottom:** Retrieval corruptions use embedding-based lookup to retrieve tasks with similar underlying rules, from which an input-output pair is sampled.

B Sampling Mixture for Classification and Editing

Table 2: Sampling weights over valid pairs and corruption types used to construct Classification candidates. The weights differ between datasets to reflect differences in corruption quality: Retrieval is less suitable for the smaller, distribution-shifted P-ARC set (lower embedding-similarity scores), and Color Flip is less challenging because P-ARC contains fewer color-transformation rules, whereas manually validated P-ARC verifiers and human errors are comparatively reliable. The overall split between valid and invalid candidates is held fixed.

Candidate type	ARC-AGI-1	P-ARC
Valid (dynamic correct)	0.250	0.250
Human error (H-ARC / P-ARC)	0.235	0.265
Instance mismatch	0.215	0.215
Corrupted verifier	0.130	0.160
Retrieval	0.103	0.073
Color flip	0.067	0.037

C Additional Experimental Results

Table 3: Model performance on ARC-AGI-1 Train and P-ARC, pooled over two independent runs ($n=800$ and $n=100$ task evaluations). Entries are percentages; brackets are pooled Wilson 95% intervals. **Indep.** is the product of the five dimension accuracies, i.e. the full-task rate expected if per-dimension pass events were independent. Bold marks the best full-task mean within each dataset block. These are the exact values underlying Figure 3.

Model	Output grid	Definition	Classification	Generation	Editing	Inversion	Full-task	Indep.
ARC-AGI-1 TRAIN								
GPT-5.4 Low	84.2 [81.6, 86.6]	73.6 [70.5, 76.6]	83.8 [81.0, 86.1]	77.2 [74.2, 80.0]	84.0 [81.3, 86.4]	85.1 [82.5, 87.4]	46.6 [43.2, 50.1]	34.0
Gemini 3.1 Pro Low	89.1 [86.8, 91.1]	79.2 [76.3, 81.9]	75.9 [72.8, 78.7]	88.0 [85.6, 90.1]	85.8 [83.2, 88.0]	88.2 [85.8, 90.3]	58.2 [54.8, 61.6]	40.0
Claude 4.6 Opus Low (120K)	87.6 [85.2, 89.7]	66.8 [63.4, 69.9]	71.5 [68.3, 74.5]	78.9 [75.9, 81.6]	88.2 [85.8, 90.3]	91.0 [88.8, 92.8]	37.6 [34.3, 41.0]	30.2
Kimi K2.5	63.4 [60.0, 66.6]	64.5 [61.1, 67.7]	63.5 [60.1, 66.8]	71.8 [68.5, 74.8]	79.6 [76.7, 82.3]	85.8 [83.2, 88.0]	38.6 [35.3, 42.0]	20.1
MiniMax M2.5	72.9 [69.7, 75.8]	56.6 [53.2, 60.0]	42.4 [39.0, 45.8]	64.6 [61.2, 67.9]	68.6 [65.3, 71.7]	76.6 [73.6, 79.4]	20.6 [18.0, 23.6]	8.1
P-ARC								
GPT-5.4 Low	31.0 [22.8, 40.6]	14.0 [8.5, 22.1]	36.0 [27.3, 45.8]	39.0 [30.0, 48.8]	34.0 [25.5, 43.7]	47.0 [37.5, 56.7]	6.0 [2.8, 12.5]	0.3
Gemini 3.1 Pro Low	45.0 [35.6, 54.8]	12.0 [7.0, 19.8]	19.0 [12.5, 27.8]	42.0 [32.8, 51.8]	32.0 [23.7, 41.7]	50.0 [40.4, 59.6]	4.0 [1.6, 9.8]	0.2
Claude 4.6 Opus Low (120K)	64.0 [54.2, 72.7]	25.0 [17.5, 34.3]	27.0 [19.3, 36.4]	56.0 [46.2, 65.3]	58.0 [48.2, 67.2]	71.0 [61.5, 79.0]	8.0 [4.1, 15.0]	1.6
Kimi K2.5	17.0 [10.9, 25.5]	10.0 [5.5, 17.4]	25.0 [17.5, 34.3]	39.0 [30.0, 48.8]	37.0 [28.2, 46.8]	47.0 [37.5, 56.7]	2.0 [0.6, 7.0]	0.2
MiniMax M2.5	24.0 [16.7, 33.2]	6.0 [2.8, 12.5]	14.0 [8.5, 22.1]	23.0 [15.8, 32.2]	17.0 [10.9, 25.5]	29.0 [21.0, 38.5]	1.0 [0.2, 5.4]	0.0

Table 4: **Self-consistency conditioned on a correct Definition.** Percentage of cases in which the model’s Definition program agrees with its own answer in the given dimension, split by whether that answer is correct (✓) or incorrect (×). Denominators in parentheses. Pooled over both runs.

Model	Classification		Constrained generation		Inversion	
	answer ✓	answer ×	answer ✓	answer ×	answer ✓	answer ×
ARC-AGI-1 TRAIN						
GPT-5.4	99.3 (3095)	14.4 (90)	97.0 (526)	31.9 (113)	98.2 (559)	65.0 (80)
Gemini 3.1 Pro	99.3 (3211)	3.2 (154)	99.4 (622)	48.0 (50)	99.2 (608)	73.0 (63)
Claude 4.6 Opus	99.2 (2759)	4.1 (196)	98.4 (491)	28.0 (100)	98.5 (545)	21.7 (46)
Kimi K2.5	99.6 (2503)	8.9 (202)	98.4 (446)	43.8 (96)	99.4 (506)	35.1 (37)
MiniMax M2.5	99.3 (2101)	3.1 (419)	96.7 (391)	38.9 (113)	97.4 (431)	47.9 (73)
Pooled	99.1 (answer ✓)		vs.	21.1 (answer ×)		
P-ARC						
GPT-5.4	96.2 (105)	40.0 (5)	100.0 (13)	11.1 (9)	85.7 (14)	25.0 (8)
Gemini 3.1 Pro	93.9 (82)	11.1 (18)	100.0 (15)	20.0 (5)	83.3 (12)	25.0 (8)
Claude 4.6 Opus	93.9 (230)	3.1 (65)	94.3 (35)	28.0 (25)	93.3 (45)	26.7 (15)
Kimi K2.5	93.7 (63)	23.5 (17)	100.0 (8)	37.5 (8)	84.6 (13)	0.0 (3)
MiniMax M2.5	94.3 (35)	20.0 (15)	83.3 (6)	50.0 (4)	100.0 (5)	20.0 (5)
Pooled	94.0 (answer ✓)		vs.	17.1 (answer ×)		

C.1 Unconditional Self-Consistency

Table 5 reports agreement between the Definition program and other dimensions without conditioning on Definition correctness. These rates are harder to interpret than the conditioned analysis in subsection 3.4: when the Definition is itself wrong, disagreement with it is uninformative about rule use. Classification agreement is substantially higher than for the two exact-generation dimensions on ARC-AGI-1, plausibly because Classification is a binary decision over externally provided candidates with many negatives, so agreement with the Definition-implied label is easier to achieve than exact grid reconstruction. On P-ARC agreement is lower and more balanced, consistent with less reliable Definition programs.

Table 5: **Unconditional self-consistency by dataset and model.** Panel (a) reports the percentage of cases in which the Definition program agrees with the model response in the corresponding dimension. Panel (b) restricts to model error cases.

(a) All cases						
Model	ARC-AGI1			P-ARC		
	Classification	Constrained generation	Inversion	Classification	Constrained generation	Inversion
Claude 4.6 Opus	88.23	74.56	78.61	66.67	55.00	62.00
GPT-5.4	93.62	77.01	83.17	68.37	37.76	31.63
Gemini 3.1 Pro	90.48	87.17	87.15	63.43	47.47	30.30
Kimi K2.5	83.50	66.67	71.05	55.80	22.22	19.19
MiniMax M2.5	72.94	61.35	66.28	48.27	22.67	10.81
Aggregate	85.83	73.43	77.32	61.10	37.79	31.91

(b) Model mistakes only						
Model	ARC-AGI1			P-ARC		
	Classification	Constrained generation	Inversion	Classification	Constrained generation	Inversion
Claude 4.6 Opus	5.73	25.61	24.64	7.14	25.00	24.14
GPT-5.4	27.98	29.28	52.10	26.88	23.33	23.53
Gemini 3.1 Pro	11.58	46.81	61.96	22.22	24.56	24.49
Kimi K2.5	12.50	26.15	16.22	12.32	11.67	7.69
MiniMax M2.5	5.81	23.51	28.25	10.08	13.79	1.85
Aggregate	9.98	28.00	35.92	15.14	19.35	15.32

C.2 Increased Sampling Budget

Table 6: GPT-5.4 sampling budget trends over 20 ARC-AGI-1 tasks, each evaluated under 10 sampled configurations across all five dimensions (200 complete five-dimension evaluations). Entries are percentages, reported as cumulative mean $\pm$ sample standard deviation across sampled configurations.

Dimension / Measure	Sample 1	Sample 2	Sample 5	Sample 10
Avg. full-task acc.	65.0 $\pm$ 24.7	65.0 $\pm$ 20.2	71.0 $\pm$ 13.2	70.0 $\pm$ 11.0
Hard full-task acc.	65.0	45.0	35.0	25.0
Definition	100.0 $\pm$ 0.0	100.0 $\pm$ 0.0	100.0 $\pm$ 0.0	100.0 $\pm$ 0.0
Classification	90.0 $\pm$ 7.1	90.0 $\pm$ 5.8	92.0 $\pm$ 5.2	92.0 $\pm$ 6.1
Generation	80.0 $\pm$ 14.1	87.5 $\pm$ 10.4	85.0 $\pm$ 8.8	83.5 $\pm$ 7.4
Editing	95.0 $\pm$ 3.5	90.0 $\pm$ 7.6	95.0 $\pm$ 5.8	94.5 $\pm$ 5.9
Inversion	90.0 $\pm$ 7.1	90.0 $\pm$ 5.8	89.0 $\pm$ 4.9	91.0 $\pm$ 4.0

D Definition Dimension Error Analysis

Because the Definition dimension asks for executable code, some failures could in principle reflect programming or interface problems rather than incorrect rule inference. They largely do not. Table 9 reports the share of collected programs that fail to load at all: between 0% and 2% for every model except MiniMax M2.5, whose 3.6% (ARC-AGI-1) and 25% (P-ARC) rates stem mainly from mangled code or natural-language text mixed into the program, apparently a side effect of substantially longer thinking chains.

We classify each Definition failure as *static* (the program does not load), *runtime* (it loads, but every failure across the four scored splits is a raised exception), or *rule* (it loads and returns at least one incorrect output grid). A program that returns a wrong grid on any scored split counts as a rule failure even if it also raises an exception elsewhere. Pooled over both runs and all five models, this gives a static/runtime/rule split of 4%/12%/84% on ARC-AGI-1 and 6%/9%/85% on P-ARC; excluding MiniMax M2.5, which dominates the static counts, 3%/11%/87% and 1%/6%/93%. Runtime

Table 7: **Run-to-run stability of full-task pass sets.** For each model, the number of ARC-AGI-1 tasks passing all five dimensions in each run, their intersection and union, the Jaccard overlap, and the percentage of tasks on which the two runs agree (both pass or both fail). P-ARC is omitted: with 0–4 passing tasks per run, the overlap statistic is not meaningful.

Model	Run 1	Run 2	Shared	Jaccard	Task agreement
GPT-5.4 Low	185	188	146	0.64	79.8%
Gemini 3.1 Pro Low	239	227	205	0.79	86.0%
Claude 4.6 Opus Low (120K)	160	141	107	0.55	78.2%
Kimi K2.5	162	147	102	0.49	73.8%
MiniMax M2.5	76	89	57	0.53	87.2%

Table 8: Failure rates by corruption type with pooled Wilson 95% confidence intervals, over both runs. “*items per model*” is the median candidate count for that type; counts vary by at most 1% across models. Several P-ARC cells rest on few candidates (14 for color flip, 42 for retrieval) and the corresponding intervals are correspondingly wide.

Model	Human	Corrupted verifier	Correct	Color flip	Retrieval	Instance mismatch
ARC-AGI-1						
Claude 4.6 Opus	20.74 [17.6, 24.3]	11.14 [8.5, 14.5]	1.84 [1.2, 2.8]	8.76 [6.2, 12.2]	4.52 [2.9, 7.0]	4.59 [3.5, 6.0]
GPT-5.4	8.21 [6.2, 10.8]	2.30 [1.3, 4.2]	5.06 [3.9, 6.5]	6.53 [4.4, 9.6]	3.15 [1.8, 5.3]	1.85 [1.2, 2.8]
Gemini 3.1 Pro	16.37 [13.5, 19.7]	8.99 [6.6, 12.0]	5.48 [4.3, 6.9]	2.82 [1.5, 5.1]	2.17 [1.1, 4.1]	6.81 [5.5, 8.5]
Kimi K2.5	18.54 [15.5, 22.0]	11.24 [8.6, 14.6]	5.98 [4.7, 7.5]	25.78 [21.5, 30.6]	10.79 [8.2, 14.1]	7.75 [6.3, 9.5]
MiniMax M2.5	31.32 [27.6, 35.3]	26.62 [22.7, 31.0]	3.66 [2.7, 4.9]	40.23 [35.2, 45.4]	40.43 [35.8, 45.2]	18.12 [15.9, 20.5]
<i>items per model</i>	562	432	1146	353	417	1086
P-ARC						
Claude 4.6 Opus	65.03 [56.9, 72.4]	12.50 [6.5, 22.8]	4.93 [2.4, 9.8]	28.57 [11.7, 54.6]	7.14 [2.5, 19.0]	12.22 [7.0, 20.6]
GPT-5.4	47.22 [39.2, 55.3]	0.00 [0.0, 5.7]	16.67 [11.5, 23.6]	0.00 [0.0, 21.5]	0.00 [0.0, 8.4]	3.26 [1.1, 9.2]
Gemini 3.1 Pro	60.42 [52.3, 68.0]	9.38 [4.4, 19.0]	22.92 [16.8, 30.4]	14.29 [4.0, 39.9]	2.38 [0.4, 12.3]	7.61 [3.7, 14.9]
Kimi K2.5	61.11 [53.0, 68.7]	14.06 [7.6, 24.6]	12.59 [8.1, 19.0]	42.86 [21.4, 67.4]	14.29 [6.7, 27.8]	11.83 [6.7, 20.0]
MiniMax M2.5	70.83 [62.9, 77.6]	23.44 [14.7, 35.1]	12.50 [8.1, 18.9]	50.00 [26.8, 73.2]	30.95 [19.1, 46.0]	23.91 [16.4, 33.6]
<i>items per model</i>	144	64	144	14	42	92

failures are concentrated in the two open-weight models: they account for 22.5% of Kimi K2.5’s and 16.1% of MiniMax’s ARC-AGI-1 Definition failures, against 1.9% for GPT-5.4 and 2.4% for Gemini 3.1 Pro. We do not separate overfitting to the demonstrations from incorrect rule inference, as the former is a special case of the latter. Overall, most Definition failures come from formalizing the wrong rule, not from producing non-executable code.

Table 9: Share of collected Definition programs that fail to load.

Model	Suite	Programs	Failed to load	%
GPT-5.4	ARC-AGI-1	800	4	0.5
Gemini 3.1 Pro	ARC-AGI-1	800	4	0.5
Claude 4.6 Opus	ARC-AGI-1	800	10	1.2
Kimi K2.5	ARC-AGI-1	800	8	1.0
MiniMax M2.5	ARC-AGI-1	800	29	3.6
GPT-5.4	P-ARC	100	2	2.0
Gemini 3.1 Pro	P-ARC	100	1	1.0
Claude 4.6 Opus	P-ARC	100	0	0.0
Kimi K2.5	P-ARC	100	0	0.0
MiniMax M2.5	P-ARC	100	25	25.0

E P-ARC Construction and Calibration

Taxonomy. Grouping the 50 tasks by their primary underlying operation gives: physics and dynamics, such as gravity, motion, bouncing and flow (9 tasks); rigid geometric transforms such as rotation, reflection, scaling and wrapping (8); connection and pathfinding, including shape connection, routing and minimum spanning trees (7); symmetry, completion, occlusion recovery and frame repair (8); object composition and assembly (5); constraint satisfaction and logic, such as sudoku completion,

map coloring and XOR of halves (6); and color or attribute mapping, such as color shifts and directional shadowing (5). Two tasks have no clean category assignment. Per-task rule descriptions and category assignments are released with the dataset.

Feasibility check. Our human protocol is a feasibility check embedded in the design process, not a formal human study with naive participants. Each candidate task was shown to team members other than its author—at least one, most often two or three—and retained only if at least one produced the correct output. Multiple attempts were permitted, but tasks were typically solved within one or two guesses. Solve times were not recorded, so we report no solve-rate or solve-time statistics; a formal human study is left to future work, and our positioning of P-ARC between ARC-AGI-1 and ARC-AGI-2 in difficulty remains a qualitative estimate. Erroneous attempts collected during this process serve as the human corruption source, with three incorrect output grids retained per task.

Generator and verifier validation. For every task, a team member other than the task’s author manually inspected at least 50 generator-produced examples and confirmed that they realize the intended rule; tasks were admitted only once this held. This validates generator and verifier behavior on their sampled outputs, which is the property our evaluation depends on, and complements the generator/verifier alignment limitation discussed in the main text.

Release. P-ARC is released under a permissive open license. Each task ships in the standard ARC JSON schema together with its generator and verifier programs, 50 stable generated examples, and the three human-error grids used as corruptions.

F Thinking Token Consumption by Claude Opus 4.6

In our experiments, Claude Opus 4.6 shows a clear increase in thinking-token consumption when moving from ARC-AGI-1 to P-ARC. After accounting for pricing differences, its token consumption on P-ARC is roughly twice that of Gemini or GPT, whereas consumption is comparable across models on ARC-AGI-1. Prompt-to-response latency increases even more sharply, yielding approximately threefold longer evaluation time, with extreme outliers requiring up to an hour for a single dimension. This initially constrained the evaluation budget, but we subsequently completed a second inference pass so that the final reported results use two runs.

We also evaluate Claude without a thinking budget in one full additional run on the ARC-AGI-1 training set. Relative to the pooled two-run configuration reported in Table 3, full-task accuracy drops from 37.6% to 21.8%. Dimension-level performance decreases across the board: Definition 66.8 $\rightarrow$ 48.0 (−18.8 points), Classification 71.5 $\rightarrow$ 62.0 (−9.5), Constrained Generation 78.9 $\rightarrow$ 58.0 (−20.9), Editing 88.2 $\rightarrow$ 65.5 (−22.7), and Inversion 91.0 $\rightarrow$ 75.8 (−15.2). The largest drop is in Editing and the smallest in Classification, indicating that enabling thinking is an important driver of Claude’s performance on PotARCin.

G Compute Budget and Hyperparameters

For all API requests, we set the temperature to 1.0 whenever this option was available. For most proprietary models evaluated, this matches the default setting when reasoning or thinking mode is enabled. For GPT-5.4 with reasoning enabled, including the generative-sampling study, the API does not expose a user-configurable temperature parameter. The embeddings used for the retrieval corruption type were computed on a MacBook Pro with an Apple M4 Pro chip. Overall, our experiments do not rely on compute-intensive procedures or hardware-specific optimizations that would materially affect reproducibility. The random seed used for all reported runs is 77 and is recorded in the run logs.

H Dimension Prompts

H.1 Definition Prompt

Find the common rule that maps an input grid to an output grid, given the examples below. Generate a python script that takes a grid as input and transforms it according to this rule. Anticipate that this program will be applied to several grids, not only the shown test grid.

Your script must define exactly one callable entrypoint:

```
def solve(grid):
```

where `grid` is passed as a list of lists of integers (`list[list[int]]`) and the function returns the transformed grid in the same format.

Example 1

Input:

```
<TRAIN_INPUT_1_GRID>
```

Output:

```
<TRAIN_OUTPUT_1_GRID>
```

Example 2

Input:

```
<TRAIN_INPUT_2_GRID>
```

Output:

```
<TRAIN_OUTPUT_2_GRID>
```

... one block per training example

Below is a test input grid.

Input:

```
<TEST_INPUT_GRID>
```

Your final answer should just be the python script itself, no other text or markdown.

H.2 Classification Distribution Prompt

Find the common rule that maps an input grid to an output grid given the examples below. You will see five additional input/output pairs (test items 0–4). For each test item, decide whether that pair could have been produced by the same underlying rule and data distribution as the training examples (1 = same distribution / consistent, 0 = not).

Training example 1

Input:

```
<TRAIN_INPUT_1_GRID>
```

Output:

```
<TRAIN_OUTPUT_1_GRID>
```

Training example 2

Input:

```
<TRAIN_INPUT_2_GRID>
```

Output:

```
<TRAIN_OUTPUT_2_GRID>
```

... one block per training example

Test pairs (evaluate each in order, index 0 ... 4):

Test 0

Input:

<TEST0_INPUT_GRID>

Output:

<TEST0_OUTPUT_GRID>

Test 1

Input:

<TEST1_INPUT_GRID>

Output:

<TEST1_OUTPUT_GRID>

... through Test 4

Your final answer must be only a single Python list of exactly five integers, each 0 or 1, in order for test 0 through test 4, for example [1, 0, 1, 1, 0]. Output nothing else: no other text, no markdown code fences, and no explanation.

H.3 Constrained Generation Prompt

Find the common rule that maps an input grid to an output grid given the examples below.

Example 1

Input:

<TRAIN_INPUT_1_GRID>

Output:

<TRAIN_OUTPUT_1_GRID>

Example 2

Input:

<TRAIN_INPUT_2_GRID>

Output:

<TRAIN_OUTPUT_2_GRID>

... one block per training example

Generate one NEW input/output pair that follows the same transformation rule.

- Do NOT copy any demonstration pair exactly.
- The generated input and output must be different (non-trivial transformation).
- Your pair must be consistent with the same underlying rule.

Your final answer must be ONLY a single Python dict with exactly these keys:

{"input": [...], ..., "output": [...], ...}

Use integer grid cells 0–9. Output nothing else.

H.4 Editing Prompt

Find the common rule that maps an input grid to an output grid given the examples below.

Example 1

Input:

<TRAIN_INPUT_1_GRID>

Output:

<TRAIN_OUTPUT_1_GRID>

Example 2

Input:

<TRAIN_INPUT_2_GRID>

Output:

<TRAIN_OUTPUT_2_GRID>

... one block per training example

The following candidate input/output pair has been corrupted in some way. Your task is to return the corrected input/output pair that is consistent with the rule inferred from the training examples.

Candidate pair

Input:

<CANDIDATE_INPUT_GRID>

Output:

<CANDIDATE_OUTPUT_GRID>

Your final answer must be ONLY a single Python dict with exactly these keys:

```
{"input": [...], ..., "output": [...], ...}
```

Use integer grid cells 0–9. Output nothing else.

H.5 Inversion Prompt

Find the common rule that maps an input grid to an output grid given the examples below.

Example 1

Input:

<TRAIN_INPUT_1_GRID>

Output:

<TRAIN_OUTPUT_1_GRID>

Example 2

Input:

<TRAIN_INPUT_2_GRID>

Output:

<TRAIN_OUTPUT_2_GRID>

... one block per training example

Now infer an input grid that would produce the following output by applying the same rule.

Desired output:

<DESIRED_OUTPUT_GRID>

Your final answer must be ONLY a single Python list-of-lists grid:

[[...], ...]

Use integer grid cells 0–9. Output nothing else.

H.6 Output Grid Correctness Prompt

Find the common rule that maps an input grid to an output grid, given the examples below.

Example 1

Input:

<TRAIN_INPUT_1_GRID>

Output:

<TRAIN_OUTPUT_1_GRID>

Example 2

Input:

<TRAIN_INPUT_2_GRID>

Output:

<TRAIN_OUTPUT_2_GRID>

... one block per training example

Below is a test input grid. Predict the corresponding output grid by applying the rule you found.

Test Input:

<TEST_INPUT_GRID>

Your final answer must be ONLY a single Python list-of-lists grid for that output:

[[...], ...]

Use integer grid cells 0–9. Output nothing else.