On the Generalization of LLM Agents: An Empirical Analysis of Agentic Harness Performance on ARC-AGI

Anonymous ACL submission

Abstract

The ARC-AGI benchmark is known to measure the generalization capabilities to novel, previously unseen tasks and is one of the few challenging benchmarks remaining for LLMs. While models exhibit some level of generalization, their effectiveness lags behind humans when it comes to transferring knowledge to new situations. The agentic harnesses developed in the last few years can alleviate many weaknesses of current models. In this paper, we introduce a harness consisting of a judge, general memory, and a Python coding module. The effect of these three methods on generalization is evaluated on a 50-task subset of the ARC-AGI 1 evaluation dataset. Multiple proprietary and open-weight models are compared to assess the current landscape of LLMs. Reusable memory abstractions and judge-driven iterative refinement emerge as useful strategies, while code-based verification improves performance only for sufficiently capable models. The cost and time dimensions are closely tracked and evaluated for all model-workflow configurations.

1 Introduction

The invention of Large Language Models (LLMs) built upon the Transformer architecture (Vaswani et al., 2017) ushered in a new era of general problem-solving capabilities (Brown et al., 2020). These LLMs are used for a variety of use cases, ranging from industrial applications to scientific endeavors. While earlier models, such as GPT-3, could only perform high school-level math with dubious accuracy, newer models are tested and excel at frontier mathematics and PhD-level scientific questions (Cao et al., 2025). The gold medal in the Math Olympiad claimed by both OpenAI and Google DeepMind – thus, ChatGPT and Gemini – in the same competition shows that the progress is applicable to more than one model provider (Castelvecchi, 2025).

While LLMs might be more general than the narrow forms of artificial intelligence of the past, the extent of this process is hard to measure. Unlike humans, LLMs learn from massive corpora and the statistical connection between the word representations. Human generalization, as formalized by Chollet (2019) in the foundational specification of the ARC-AGI benchmark, is characterized by the capacity to acquire new skills efficiently from very few examples and to apply them flexibly across novel contexts. LLMs, despite their scale, rely on statistical pattern matching over previously encountered distributions and often require task-specific demonstrations to correctly interpret even conceptually familiar problems: the so-called few-shot learning paradigm (Brown et al., 2020).

This constitutes a fundamentally different cognitive process from human reasoning, and the gap is most clearly visible on the Abstract and Reasoning Corpus (ARC-AGI) benchmark. ARC-AGI tasks consist of small, grid-based visual puzzles that require the induction of transformation rules from a handful of input-output examples. The tasks are deliberately designed to be novel, requiring no domain knowledge but substantial abstract reasoning capacity (Chollet, 2019). The benchmark remains the only widely used evaluation suite not yet saturated by frontier models (Ilievski et al., 2025), a property that has persisted through the release of ARC-AGI-2 and ARC-AGI-3 (Chollet et al., 2025). Baseline LLM failures on ARC-AGI can be attributed to concrete architectural limitations: models lack an inherent reflection and self-correction mechanism for effective hypothesis generation, accumulate no persistent knowledge across problem instances, and lack access to deterministic spatial reasoning primitives that humans apply naturally.

Each of these three failure modes corresponds to a well-established body of research in agentic AI that proposes a targeted architectural remedy. The LLM-as-a-Judge paradigm leverages the empiri-

cal finding that language models are substantially more reliable at critiquing and evaluating existing solutions than at generating novel correct ones in a single pass, enabling iterative self-correction without human intervention (Li et al., 2025). Memory-augmented and retrieval-augmented systems address the absence of a persistent knowledge base by allowing agents to store, index, and retrieve learned information across interactions: an approach demonstrated to yield meaningful improvements on abstract reasoning tasks (Ho et al., 2025; Lewis et al., 2021). Coding strategies with a self-correction loop allow the agent to write a deterministic transformation function to solve the task. Together, these three mechanisms form a principled compositional harness that directly targets the identified failure modes in an empirically testable fashion. We introduce the Wanderer ARC-AGI Solver, a research platform implementing all three components as modular, progressively activated solver configurations, and evaluate each on a controlled 50-task subset of the ARC-AGI-1 evaluation dataset across multiple LLMs.

The scientific contribution of this paper is threefold: we formalize the relationship between specific architectural augmentations and measurable generalization performance on ARC-AGI; we provide a systematic comparison of four solver configurations across multiple model capability tiers; and we report cost, token consumption, and execution time as practical dimensions of agent harness deployment. To structure the empirical inquiry, we pose the following research question. RQ: Under what boundary conditions (defined by model selection, workflow configuration, and task structural complexity) does the agent framework achieve the highest proportion of correctly solved tasks, and can it achieve consistent generalization to previously unseen ARC-AGI tasks without any human intervention during inference?

In the following section, the related literature will be presented, followed by a separate chapter on the methodology and experimental design selected for this paper. After that, the initial setup of the general knowledge base will be described, followed by the results and evaluation, then a discussion section, and finally, conclusions drawn from the current study.

2 Literature Review

Agentic harnesses have attracted significant attention in both mission-critical AI agent deployments in corporate environments and in scientific research aimed at extending the current capabilities of LLMs, where verification, traceability, and reliability are core components of a successful system. A harness can include multiple components that fill in the gaps of jagged model capabilities (Dell'Acqua et al., 2023): in this paper, the three crucial elements examined are (1) LLM-as-a-Judge, (2) a general memory, and (3) coding the spatial transformation, in order to test the generalizability of these components through experiments on the ARC-AGI 1 evaluation dataset.

The ARC-AGI benchmark has attracted considerable attention as an evaluation suite that resists the statistical memorization strategies underlying the performance gains of frontier LLMs on conventional benchmarks. The ARC-AGI 1 tasks consist of a variable-sized grid with at most 10 colors, each used to fill the grid with patterns. These patterns are created from the changing rules derived from a limited number of input-output pair examples that the system needs to infer these rules from. As such, ARC tasks are designed so that no two are alike and require a high level of general problem-solving to complete multiple tasks successfully.

Bober-Irizar and Banerjee (2024) demonstrate systematically that neural networks struggle with the compositional, rule-based generalization that ARC-AGI demands, even when provided with carefully engineered prompts or fine-tuning regimes. Joffe and Eliasmith (2025) explore vector symbolic algebra architectures as a complementary approach, finding partial improvements on specific task subtypes but no general solution to the induction problem. Ilievski et al. (2025) provide a broader synthesis of the alignment problem between human and machine generalization, concluding that current approaches, such as scaling, instruction tuning, and chain-of-thought prompting, do not close the gap on novel task induction. The core difficulty, as Chollet (2019) articulates, is that ARC tasks are designed to minimize the exploitation of prior knowledge and maximize the need for fluid, core-knowledge reasoning. This makes ARC a structurally distinct challenge relative to the benchmarks on which LLMs have historically excelled, and it motivates the investigation of architectural solutions.

The LLM-as-a-Judge paradigm has emerged as a productive method for improving output quality in language model pipelines without requiring additional human annotation or ground-truth verification at each step. The central empirical finding motivating this approach is that language models evaluate and critique solutions more reliably than they generate correct novel outputs in a single forward pass (Zheng et al., 2023). The primary advantages of this paradigm are scalability, task flexibility, and pipeline compatibility (Li et al., 2025). Self-refinement, meaning iterative model-generated feedback, demonstrably improves solution quality across diverse task types (Madaan et al., 2023). In the context of ARC-AGI, where the correctness of a predicted grid can be verified against ground-truth, the judge can evaluate both the logical coherence of the proposed transformation rule and the structural correctness of the applied prediction. This creates a closed feedback loop that transforms a single-pass prediction into a structured, iterative process of hypothesis refinement.

Memory and retrieval mechanisms address a second fundamental limitation of base LLMs: the absence of a persistent, updatable knowledge base that can be accumulated through iterative interaction with the environment, enabling experience-based learning throughout the solving process. Lewis et al. (2021) introduced Retrieval-Augmented Generation (RAG) as a framework for connecting LLMs to external knowledge stores, demonstrating that grounding model responses in retrieved documents substantially reduces hallucination and improves factual accuracy. Shuster et al. (2021) extend this finding to conversational settings, showing that retrieval augmentation improves both accuracy and response consistency over multi-turn interactions. In agentic systems, the memory module serves an additional function beyond simple grounding: it enables the progressive accumulation of inferred knowledge, allowing the agent to build a corpus of generalizable principles from its own problem-solving experience. Ho et al. (2025) apply this idea directly to ARC-AGI in the ArcMemo framework, demonstrating that lifelong memory augmentation enables significant improvements on tasks that share structural patterns with previously solved examples. The implementation of cross-task retrieval, where knowledge derived from one ARC task is surfaced during the solution of a structurally related task, operationalizes a form of within-run transfer learning that partially replicates the analogical reasoning capacity humans apply naturally to this benchmark.

Agentic code generation represents a third line of augmentation that addresses a qualitatively distinct limitation of LLMs on spatial reasoning tasks: the unreliability of direct grid prediction as a mechanism for applying inferred transformation rules. Rather than asking the model to predict the output grid token-by-token, this approach directs the LLM to synthesize a deterministic Python function *transform(grid)* that encodes the inferred rule as executable logic, separating the cognitive act of rule discovery from the computational act of rule application. Chen et al. (2023) demonstrate that grounding code generation in execution feedback substantially improves correctness, a principle utilized here through an automated training verification loop that applies the synthesized function to every training example and compares cell-by-cell against the ground-truth output. When verification fails, the agent is prompted to abandon its current hypothesis entirely and induce an alternative rule, rather than just patching syntax, thus enforcing genuine hypothesis revision across repair cycles (Olausson et al., 2024). This architecture converts an open-ended spatial reasoning problem into a program synthesis problem with an exact, automatically computable correctness criterion. In ARC-AGI, every task has a unique ground-truth output against which generated code can be verified without human judgment.

3 Methodology and Experimental Setup

The Wanderer ARC-AGI Solver research program was developed using an AI-assisted incremental development approach, with OpenAI Codex guided by a structured implementation backlog consisting of eight milestones, each with explicit scope definitions and acceptance criteria. This approach required continuous manual supervision at milestone boundaries. It reflects an emerging practice in which LLM-assisted code generation accelerates implementation velocity while the researcher retains full architectural and methodological control over the resulting system. The implementation process also demonstrates that multi-agent research platforms can be developed efficiently with appropriate human-AI collaboration structures. The workflow prompt templates are disclosed in Annex A to support reproducibility.

The experimental design is structured as a progressive ablation study with four solver configurations, denoted M1-M4, each adding exactly one augmentation layer to the previous configuration to isolate the marginal contribution of each architectural component. M1 is the baseline: a single-pass LLM prompt encoding the ARC task's training input-output examples alongside the test input, requesting a predicted output grid and an explanation of the transformation rule. No harness is provided to the model in the M1 workflow.

M2 introduces the solver-judge loop, in which the solver generates a full hypothesis including an explicit rule statement, and the same judge LLM evaluates the hypothesis against the task specification, returning a structured JSON verdict with an accept or reject signal and targeted corrective feedback. This loop produces up to three solver attempts per task under a fixed experimental retry budget. M3 augments M2 with persistent memory. Prior to each solver attempt, the full prebuilt general knowledge base, consisting of 16 reusable procedural entries, is provided to the solver together with summaries of prior attempts on the same task. The construction of this knowledge base is described separately in Section 4. During the comparative evaluation runs, the same memory base was used across the relevant workflows to make the M3 and M4 results comparable. M4 further augments M3 with the coding verification loop, allowing the solver to perform program synthesis before submitting its final solution.

Experiments are conducted on a 50-task sample drawn from the ARC-AGI-1 public evaluation split (Chollet, 2019) rather than the public training split. This reduces direct training-split contamination, although it does not eliminate the broader risk of exposure associated with public benchmark data. Task correctness is determined by an exact grid match between the solver's predicted output and the ground-truth test output, following the standard ARC Prize evaluation protocol without partial credit. All four workflow configurations are applied to the same task set across all models, enabling comparisons across model capability tiers. The primary evaluation metric is the task solve rate: the proportion of the 50 tasks for which the solver produces a correct output within the allowed retry attempts. Secondary metrics include qualitative judge/check outcomes, aggregate token consumption, monetary cost, and runtime per run, all of which are logged natively by the platform for every provider call.

4 Memory Experiment

The memory was set up through an initial run of the 50 questions. The knowledge base was constructed through an iterative, task-driven abstraction process. From the beginning, the objective was not to collect task-specific solutions but to build a general, reusable procedural knowledge base that could support the solution of as many ARC-AGI tasks as possible (Chollet, 2019). For each task, the model was asked not only to solve the grid transformation, but also to evaluate whether one of the existing knowledge base entries already captured the underlying method. This made the process explicitly metacognitive: every task served both as a benchmark instance and as evidence for whether the current knowledge representation was sufficiently general.

This construction process required a strong reasoning model capable of operating at two levels simultaneously. First, the model had to infer the actual ARC-AGI transformation rule with sufficient reliability. Second, it had to analyze the existing knowledge base and decide whether the current task required: (1) a new knowledge base entry, (2) an extension of an existing entry, or (3) no change because an existing entry already covered the required method. The chosen model for this knowledge base construction phase was ChatGPT 5.5, because it is a top-tier reasoning model and, in our experimental setting, was suitable for both solving ARC-style transformations and compressing the observed solution strategies into reusable abstractions. We also attempted to use weaker LLMs for this role, but they were substantially less reliable: they failed more often at solving the ARC-AGI tasks themselves, and were even less capable of producing stable, general, non-overfitted knowledge base abstractions.

Each knowledge base entry was represented by four fields: title, type, solution, and keywords. The title functions as a compact summary of the method and supports fast inspection and, in larger future knowledge bases, retrieval of potentially relevant entries; the keyword field serves a similar indexing role, with the number of expressions intentionally kept small. The type field describes the recognizable task characteristics that suggest when a given method may be applicable, and the solution field contains the procedural reasoning steps needed to apply the method. When an existing entry had to

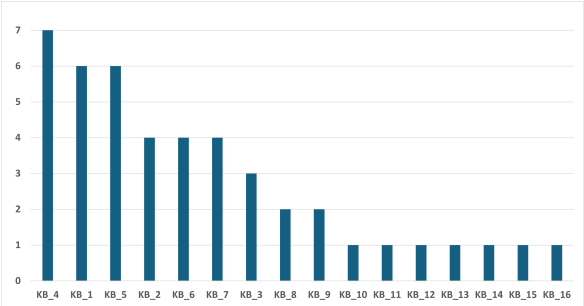

Figure 1: Knowledge base entries by number of extensions (versions).

be expanded, the extension was added to the solution section rather than to the type section. The design reflects an important empirical observation: the diversity of ARC-AGI tasks often arises not from completely new high-level categories, but from variations in how a relatively small number of recurring transformation families are instantiated. In 29 cases, an existing entry only required a short solution-level refinement; in five cases, no modification was needed at all. This supports the view that ARC-AGI task diversity can be partially compressed into reusable procedural schemas.

This first phase of the experiment suggests that a substantial portion of ARC-AGI-1 tasks can be described by reusable procedural abstractions rather than by fully task-specific rules. Across the first 50 evaluation tasks, the interaction produced 16 distinct knowledge base entries, corresponding to 32% of the examined task count. This ratio is important: it does not mean that only 32% of tasks were solvable by general knowledge, but rather that 50 individual tasks could be compressed into 16 reusable methodological patterns. The resulting knowledge base, therefore, acts as a form of abstraction layer over the task set, replacing many isolated solutions with a smaller number of transferable reasoning schemas. The final table contains 16 entries with fields for title, type, solution, keywords, version, and identifier.

The distribution of version counts further supports this interpretation. Several entries were reused and refined across multiple tasks. The most frequently updated entry, **KB_4**, "Recoloring an object based on a mask or marker pattern," reached version 7. This indicates that mask-based control, key-pattern interpretation, and marker-based recoloring are not isolated phenomena but recurring mechanisms across different ARC-AGI tasks. Similarly, **KB_1**, related to horizontal and vertical connections, and **KB_5**, related to internal regions of frames, contours, and solid objects, reached version 6. These three high-version entries suggest that a considerable part of the sampled task space is organized around a limited number of structural principles: spatial connection, mask-based control, and interpretation of object interiors.

The version distribution is also informative because it is not uniform. Some entries remained at version 1, while others accumulated several refinements. This produces a long-tailed structure: a small number of general patterns explain many task variants, while a larger number of more specialized patterns explain fewer cases. Such a distribution is consistent with the hypothesis that ARC-AGI tasks are diverse at the surface level but partially repetitive at the level of cognitive operations. The same general operation may appear under different visual conditions: connecting points may become full-row or full-column projections; a mask may guide color mapping or mark certain blocks for modification; and a contour may require filling, extracting, or recoloring internal holes. The important observation is that these variations did not always require new knowledge base entries. Often, a short extension to the "solution" field was sufficient.

In five cases, the existing knowledge base required no modification at all: the already formulated entry was sufficient for the new task. In the remaining 45 cases, the knowledge base generally did not need to be replaced but only extended with a few additional sentences in the solution section. This suggests that the knowledge representation is not merely a list of memorized task solutions. Instead, it behaves like a growing set of operational templates. Each new task can either instantiate an existing template, refine it, or, less frequently, motivate the creation of a new template. From the perspective of weaker LLMs, this is precisely the desirable property of an external knowledge base: it should not provide exact answers to future tasks, but it should steer the model toward useful abstractions and reduce the search space of possible transformations.

5 Results

The experiments use four standardized solver configurations. M1 is the baseline single-pass solver. M2 adds the solver-judge loop. M3 adds retrieval from the task-level and general memory system. M4 adds the deterministic Python code synthesis

Model	M1	M2	M3	M4
4o-mini	3/50 (6%)	3/50 (6%)	5/50 (10%)	0/50 (0%)
5.4-mini	4/50 (8%)	7/50 (14%)	5/50 (10%)	8/50 (16%)
Qwen3-480B	6/50 (12%)	9/50 (18%)	10/50 (20%)	3/50 (6%)

Table 1: Exact-match solve rate on the 50-task ARC-AGI-1 evaluation subset. M1 is the baseline, M2 adds the judge loop, M3 adds memory, and M4 adds code verification.

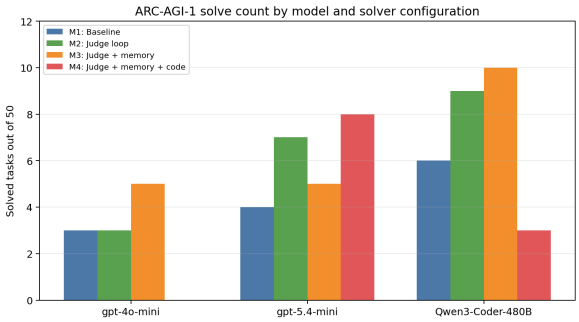

Figure 2: Solved tasks by model and solver configuration.

and verification loop on top of M3.

The result pattern is mixed but informative. The highest absolute score is 10 solved tasks out of 50, obtained by Qwen3-Coder-480B with M3. GPT-5.4-mini reaches its best result with M4, solving 8 tasks, while GPT-4o-mini performs best with M3, solving 5 tasks. These numbers confirm that ARC-AGI-1 (Chollet, 2019) remains difficult even under agentic augmentation, but they also show that the harness changes the behavior of the models in measurable ways. Improvement over the M1 baseline was observed for all models examined. However, the gains are not uniform across model-workflow combinations.

The judge loop improves the two stronger models but not the weakest one. GPT-4o-mini remains at 3 solved tasks in M1 and M2, while GPT-5.4-mini improves from 4 to 7 solved tasks and Qwen3-Coder-480B improves from 6 to 9. This suggests that judge feedback is useful only when the solver model has enough reasoning capacity to revise its first hypothesis. In many failed cases, the judge identifies a plausible correction, but the solver does not implement the correction precisely and instead remains close to its initial interpretation of the task.

The memory condition also produces non-uniform gains. It is the best configuration for GPT-4o-mini and Qwen3-Coder-480B, but not for GPT-5.4-mini. This supports the hypothesis that reusable procedural knowledge can narrow the search space, but it does not solve the central ARC problem by itself. General knowledge about frames, masks, recoloring, object movement, or symmetry is helpful only if the model can convert that knowledge into an exact output grid. The results indicate a separation between abstract rule recognition and precise matrix generation.

Model	Cfg	Solved	GOOD checks	GOOD but failed	BAD checks	BAD but solved
4o-mini	M3	5	3	2	39	4
4o-mini	M4	0	5	5	30	0
5.4-mini	M3	5	17	12	24	0
5.4-mini	M4	8	24	17	10	1
Qwen3-480B	M3	10	11	4	31	3
Qwen3-480B	M4	3	9	6	32	0

Table 2: Relationship between qualitative solution checks and exact-match outcomes. Cfg denotes configuration, Solv. denotes solved tasks, G-fail denotes GOOD but failed, and B-solv. denotes BAD but solved.

The coding condition is the most model-dependent augmentation. It improves GPT-5.4-mini from 5 solved tasks in M3 to 8 in M4, but it reduces Qwen3-Coder-480B from 10 to 3 and GPT-4o-mini from 5 to 0. This does not necessarily mean that program synthesis is a weak approach. Rather, the M4 workflow is stricter than the direct matrix workflows: generated code must reproduce every training output exactly before it is accepted, and any mismatch is sent back to the solver as a failure. By contrast, M3 can occasionally produce the correct test matrix even from an incomplete or partly accidental rule description. M4, therefore, penalizes brittle hypotheses earlier and more consistently.

Table 2 shows why qualitative rule quality and exact task success should be reported separately. Several tasks receive a positive qualitative check while still failing the exact-match evaluation. These cases are especially important for ARC-AGI because they show that the model may infer the correct family of transformation while still failing at object localization, color assignment, boundary handling, or matrix construction. Conversely, a small number of solved tasks can occur even when the rule explanation is weak, because direct output prediction can sometimes succeed without a stable generalizable explanation.

The cost analysis strengthens the interpretation that more agentic tools are not automatically better. For GPT-5.4-mini, M4 is the most accurate configuration, but it also costs $4.47 for the 50-task

Model	Cfg	Solved	Total cost	Cost/ solved	Total runtime	Avg runtime/ task
4o-mini	M1	3	$0.09	$0.03	1079s	21.6s
4o-mini	M2	3	$0.16	$0.05	1434s	28.7s
4o-mini	M3	5	$0.22	$0.04	2970s	59.4s
4o-mini	M4	0	$0.54	n/a	1204s	24.1s
5.4-mini	M1	4	$0.42	$0.11	189s	3.8s
5.4-mini	M2	7	$0.83	$0.12	345s	6.9s
5.4-mini	M3	5	$1.18	$0.24	415s	8.3s
5.4-mini	M4	8	$4.47	$0.56	1204s	24.1s
Qwen3-480B	M1	6	$0.87	$0.14	544s	10.9s
Qwen3-480B	M2	9	$1.42	$0.16	708s	14.2s
Qwen3-480B	M3	10	$2.56	$0.26	804s	16.1s
Qwen3-480B	M4	3	$6.97	$2.32	1658s	33.2s

Table 3: Cost and runtime by model and solver configuration. C/S denotes cost per solved task, Run denotes total runtime, and Avg denotes average runtime per task.

run, compared with $0.83 for M2 and $1.18 for M3. For Qwen3-Coder-480B, the highest-cost run is M4 at $6.97, yet it solves only 3 tasks; the M3 memory condition solves 10 tasks for $2.56. The practical conclusion is that the best harness should be selected by marginal utility, not by architectural complexity.

The Wanderer ARC-AGI Solver records each provider call, token count, cost, runtime, predicted matrix, rule explanation, judge verdict, and code-verification result in a structured database. This makes the system useful not only as a solver but also as an experimental harness: failures can be inspected at the level of model reasoning, retrieved memory, generated code, and exact grid output. The database-backed design is what enables the present analysis of solve rates, qualitative checks, runtime, and cost from the same experimental runs.

6 Discussion

The central finding is that external structure helps LLMs on ARC-AGI-1 (Chollet, 2019), but it does not remove the main bottleneck. General memory can store reusable procedural abstractions (Ho et al., 2025; Lewis et al., 2021), and the judge can often identify what should change (Li et al., 2025), yet the solver frequently fails to execute the suggested revision with sufficient precision. This indicates that ARC-AGI performance is limited not only by access to knowledge, but by the model's ability to revise a hypothesis and produce an exact spatial output.

The models can produce effective heuristic rules for solving the task by leveraging memory, but ultimately fail to implement them. Program synthesis is especially useful for the newest and most developed models, such as GPT-5.4-mini. However, the costs rise significantly as the complexity of the harness around the model increases. Runtime shows a similar trend, but it does not depend solely on the harness; rather, the model has the greatest effect. This is why GPT-5.4-mini solves tasks in half the time it would take 4o-mini to solve them, even in the most complex configuration.

7 Conclusion

This study evaluated a modular agentic harness on a 50-task subset of ARC-AGI-1 across three models. The results show modest but meaningful gains from judge and memory augmentation, while code-based verification is beneficial only for the model that can reliably translate inferred rules into executable transformations. The broader conclusion is that ARC-AGI remains difficult because recognizing a general transformation family is not sufficient: the solver must also implement that rule exactly at the matrix level. Future work should therefore focus on better hypothesis revision, more robust object and relation extraction, and tighter integration between symbolic verification and model-generated reasoning.

Limitations

Several limitations should be noted. The memory base was prebuilt before the comparative runs, and all workflows received the same first 50 ARC-AGI-1 evaluation tasks in the same order. This controls task order across workflows, but the 50-task subset remains an exploratory sample rather than a full benchmark evaluation. Exact-match scoring is intentionally strict and does not capture partial progress. The qualitative GOOD/BAD checks used in the analysis are judge-authored and researcher reviewed, so they should be treated as interpretive labels rather than independent ground truth. Cost values are also time- and provider-dependent.

For this reason, the claims of this study are deliberately cautious. The results provide evidence of limited transfer and model-dependent benefit from agentic augmentation, but they do not show that memory, judging, or code execution solves generalization on ARC-AGI. The more robust conclusion is that these components expose different bottlenecks: hypothesis generation, hypothesis revision, retrieval of reusable abstractions, and exact ma-

trix execution (Chen et al., 2023; Olausson et al., 2024).

References

Mikel Bober-Irizar and Soumya Banerjee. 2024. Neural networks for abstraction and reasoning: Towards broad generalization in machines. In Review.

Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared D. Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, and 12 others. 2020. Language models are few-shot learners. In *Advances in Neural Information Processing Systems*, volume 33, pages 1877–1901.

Yang Cao, Yubin Chen, Xuyang Guo, Zhao Song, Song Yue, Jiahao Zhang, and Jiale Zhao. 2025. Evaluating frontier llms on phd-level mathematical reasoning: A benchmark on a textbook in theoretical computer science about randomized algorithms. *Preprint*, arXiv:2512.13978.

Davide Castelvecchi. 2025. Deepmind and openai models solve maths problems at level of top students. *Nature*, 644(8075):20–20.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. 2023. Teaching large language models to self-debug. *Preprint*, arXiv:2304.05128.

François Chollet. 2019. On the measure of intelligence. *Preprint*, arXiv:1911.01547.

François Chollet, Mike Knoop, Gregory Kamradt, Bryan Landers, and Henry Pinkard. 2025. Arc-agi-2: A new challenge for frontier ai reasoning systems. *Preprint*, arXiv:2505.11831.

Fabrizio Dell'Acqua, III McFowland, Edward, Ethan Mollick, Hila Lifshitz-Assaf, Katherine C. Kellogg, Saran Rajendran, Lisa Krayer, François Candelon, and Karim R. Lakhani. 2023. Navigating the jagged technological frontier: Field experimental evidence of the effects of ai on knowledge worker productivity and quality. Harvard Business School Technology & Operations Management Unit Working Paper, 24–013.

Matthew Ho, Chen Si, Zhaoxiang Feng, Fangxu Yu, Yichi Yang, Zhijian Liu, Zhiting Hu, and Lianhui Qin. 2025. Arcmemo: Abstract reasoning composition with lifelong llm memory. *Preprint*, arXiv:2509.04439.

Filip Ilievski, Barbara Hammer, Frank van Harmelen, Benjamin Paassen, Sascha Saralajew, Ute Schmid, Michael Biehl, Marianna Bolognesi, Xin Luna Dong, Kiril Gashteovski, Pascal Hitzler, Giuseppe Marra, Pasquale Minervini, Martin Mundt, Axel-Cyrille Ngonga Ngomo, Alessandro Oltramari, Gabriella Pasi, Zeynep G. Saribatur, Luciano Serafini, and 6 others. 2025. Aligning generalization between humans and machines. *Nature Machine Intelligence*, 7(9):1378–1389.

Isaac Joffe and Chris Eliasmith. 2025. Vector symbolic algebras for the abstraction and reasoning corpus. Version 1.

Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. 2021. Retrieval-augmented generation for knowledge-intensive nlp tasks. *Preprint*, arXiv:2005.11401.

Dawei Li, Bohan Jiang, Liangjie Huang, Alimohammad Beigi, Chengshuai Zhao, Zhen Tan, Amrita Bhattacharjee, Yuxuan Jiang, Canyu Chen, Tianhao Wu, Kai Shu, Lu Cheng, and Huan Liu. 2025. From generation to judgment: Opportunities and challenges of llm-as-a-judge. *Preprint*, arXiv:2411.16594.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pages 46534–46594.

Theo X. Olausson, Jeevana Priya Inala, Chenglong Wang, Jianfeng Gao, and Armando Solar-Lezama. 2024. Is self-repair a silver bullet for code generation? *Preprint*, arXiv:2306.09896.

Kurt Shuster, Spencer Poff, Moya Chen, Douwe Kiela, and Jason Weston. 2021. Retrieval augmentation reduces hallucination in conversation. *Preprint*, arXiv:2104.07567.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. In *Advances in Neural Information Processing Systems*, volume 30.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. 2023. Judging llm-as-a-judge with mt-bench and chatbot arena. In *Advances in Neural Information Processing Systems*, volume 36, pages 46595–46623.

A Prompt Templates

M3 used the M2 solver prompt with additional retrieved task-memory, cross-task-memory, and knowledge base context inserted by the workflow. M4 in this paper corresponds to the code-verification configuration.

wanderer-arc-agi-solver/backend/app/workflows/prompts/m1_baseline_solver.txt
You are solving ARC-AGI-1 logical tasks.

These tasks may have solutions available online, but web search and any other external resource usage are strictly forbidden.

The system provides two or more example input-output matrix pairs that all fit the same small set of simple logical rules.
Your task is to infer those rules so that the same rules applied to all example input matrices reproduce the corresponding output matrices exactly.
The rules may change matrix size, shapes, and colors depending only on some visible, computable property of the matrix. Nothing else may change.
There is always a valid solution and a rule set that leads to the correct result.
Finally, there is a test task: the last input matrix has no output matrix, and you must produce the missing output matrix for evaluation.

Reason only from the provided examples.
Prefer simple, consistent rules over complex ones.
The rule set must fit all provided examples exactly.
Do not use memorized task IDs, web knowledge, or external references.

Return only valid JSON with this exact shape.
The matrix below is only a tiny schema example, not a size hint. Replace it with the full predicted output grid for the test input. The output grid may have any row and column count implied by the examples, and every cell must be an integer color value from 0 to 9.

```
{
  "solution_text": "short explanation of the inferred transformation",
  "output_grid": [[0, 1, 4], [2, 3, 9]],
  "confidence": 0.0
}
```

`confidence` must be a real number between `0.0` and `1.0` indicating how confident you are that your predicted `output_grid` is correct. A value closer to `1.0` means higher confidence, and a value closer to `0.0` means lower confidence.

Do not include markdown fences or any text before or after the JSON object.

Figure 3: M1 baseline solver prompt

wanderer-arc-agi-solver/backend/app/workflows/prompts/m2_solver.txt
You are solving an ARC-AGI-1 task.

These are ARC-AGI-1 logical tasks whose solutions can be found online, therefore web search and any other external resource usage are strictly forbidden.

The system provides two or more example input-output matrix pairs that all fit the same small set of simple logical rules.
Your task is to infer these simple logical rules so that the same rules applied to all example input matrices reproduce the corresponding output matrices exactly.
The rules may change matrix size, shapes, and colors depending only on some visible, computable property of the matrix. Nothing else may change.
There is always a valid solution and a rule set that leads to the correct result.
Finally, there is a test task: the last input matrix has no output matrix, and you must produce the missing output matrix for evaluation.

Reason only from the provided examples.
Prefer simple, consistent rules over complex ones.
The rule set must fit all provided examples exactly.
Do not use memorized task IDs, web knowledge, or external references.
If judge feedback from a previous attempt is provided, use it to produce a new full hypothesis rather than a partial correction.

Return strict JSON only with this shape:

```
{
  "solution_text": "short explanation of the inferred transformation",
  "output_grid": [[0, 1], [1, 0]],
  "confidence": 0.0,
  "operation": {
    "type": "marker_shape_recolor"
  }
}
```

Rules:
- The matrix above is only a tiny schema example, not a size hint. Replace it with the full predicted output grid for the test input.
- The output grid may have any row and column count implied by the examples, and every cell must be an integer color value from 0 to 9.
- `confidence` must be a real number between `0.0` and `1.0`.
- `operation` is optional. Include it when the transformation matches a clear executable family such as `consistent_color_mapping`, `closed_rectangle_fill`, `marker_shape_recolor`, `component_compact_layout`, `masked_complement_tiling`, `component_size_filter`, or `line_connection`.
- If you include `operation`, add useful parameters when you can infer them, but do not invent unsupported details. The backend may execute the operation exactly and compare candidate grids.

Figure 4: M2 solver prompt

wanderer-arc-agi-solver/backend/app/workflows/prompts/m2_judge.txt

You are judging a solver's full ARC-AGI-1 hypothesis.

Do not use web search or any external resource.
Reason only from the provided ARC task, the solver output, and the current attempt context.
Treat the provided parsed solver response JSON as the single solver hypothesis for this attempt.
It may include `solution_text`, `output_grid`, and confidence fields.

Your primary job is to judge whether the solver's predicted `output_grid` is supported by the training examples.
Evaluate whether the solver's full hypothesis is consistent with all provided training examples.
Also check whether the solver's predicted `output_grid` actually follows the solver's own stated `solution_text`.
Identify concrete contradictions, mismatches, or missing structure that would make the predicted output unreliable.
Decide whether the hypothesis should be accepted or rejected.
If rejected, provide structured feedback that helps the solver produce a new full attempt rather than a partial patch.

Important judging rules:
- Prioritize the predicted `output_grid` over the elegance of the explanation.
- Check both:
  1. consistency between the training examples and the predicted `output_grid`
  2. consistency between the solver's `solution_text` and the predicted `output_grid`
- Do not reject only because the solver's `solution_text` is informal, incomplete, underspecified, or not maximally general.
- Do not reject only because the solver uses exact pattern matching, template matching, or lookup-style reasoning. In ARC, such reasoning can be valid if it fits all provided examples consistently.
- Accept if the predicted output is plausibly supported by the training examples, even when the verbal explanation is imperfect.
- Reject if the solver's own `solution_text` implies one output but the provided `output_grid` shows a different transformation.
- Reject only when you can point to specific contradictions or missing support that make the predicted output unreliable.
- Prefer concrete example-based criticism over abstract complaints such as "too ad hoc", "not general enough", or "too memorized".
- If the invariant transformation appears correct and only the explanatory wording is weak, prefer acceptance.
- Do not generate a corrected output grid. Your role is to judge and provide structured feedback for the next full solver attempt.

Return strict JSON only with this shape:

```
{
  "acceptance": false,
  "failure_reasons": ["short reason"],
  "contradictions": ["specific contradiction against the examples"],
  "suggested_corrections": ["guidance for a new full hypothesis"],
  "confidence": 0.0
}
```

Rules:
- `acceptance` must be a boolean.
- `failure_reasons`, `contradictions`, and `suggested_corrections` must always be arrays of strings.
- `confidence` must be either a number between `0.0` and `1.0` or `null`.
- Keep the JSON compact and parseable.
- When rejecting, each `failure_reasons` entry must be a concrete blocking issue for the predicted output, not just a stylistic criticism of the explanation.
- When relevant, explicitly mention contradictions between the solver's stated solution and the actual predicted grid.

Figure 5: M2 judge prompt

wanderer-arc-agi-solver/backend/app/workflows/prompts/m4_solver.txt
You are solving an ARC-AGI-1 task by writing executable Python transformation code.

Important constraints:
- Use only the provided ARC examples and Knowledge Base guidance.
- Web search and any external resources are strictly forbidden.
- Your code must define exactly one function named `transform`.
- `transform(grid)` receives a rectangular list of lists of integers.
- `transform(grid)` must return the predicted output grid as a rectangular list of lists of integers.
- NumPy is available as `np` and `numpy`; use it only for deterministic grid computation.
- Do not read files, write files, use network access, spawn processes, or depend on randomness.
- Prefer simple, general code that reproduces every training example exactly.

Before producing the final JSON, internally verify your transform:
1. Apply `transform` to every training input.
2. Compare each generated training output against the provided training output cell by cell.
3. If any generated training output differs in shape, color, or cell value, revise the code.
4. Only after your code reproduces every training output exactly, provide the final JSON.

If a derived mask/tile analysis is provided, inspect it before writing code. It is computed only from the training examples and may reveal whether output tiles correspond to the input mask, its complement, or another simple mask relationship.

Return strict JSON only with this shape:
```
{
  "solution_text": "short explanation of the inferred transformation",
  "transform_code": "def transform(grid):\n    return grid",
  "confidence": 0.0
}
```

Rules:
- `transform_code` must be a complete Python function definition for `transform`.
- Do not include markdown fences.
- Do not include commentary outside the JSON object.
- Every returned grid cell must be an integer color value from 0 to 9.

Figure 6: M4 solver prompt

B Qualitative Case Studies

The following examples support the interpretation in the main text; they are not additional benchmark results. GOOD/BAD labels are judge-authored and researcher-reviewed.

B.1 Memory Helped Where the Baseline Did Not

This example is a strong case where memory changes the model's hypothesis class. The M1 baseline proposes a shallow color-substitution rule, while the M3 memory condition identifies the task as a framed-object interior-filling problem. This supports the interpretation that retrieved procedural abstractions can guide the model toward the correct transformation family, not merely improve the final answer by chance.

Field	Content
Model / task	gpt-4o-mini; task 009d5c81
M1 result	**FAILED.** The transformation involves replacing the color 8 with 6, and the color 1 with 4, while keeping the rest of the colors unchanged.
M3 result	**SOLVED.** The output matrix is generated by filling the interiors of 2-bordered rectangles with specific colors based on the frame. The top and bottom frames are filled with color 3, while the middle frames are filled with color 2, preserving the borders and any inner 2-cells as holes.

Table 4: Memory helped where the baseline did not.

B.2 Judge Feedback Helped Over the Baseline

This case illustrates the positive side of the solver-judge loop: feedback can cause a successful second or third hypothesis when the model is capable of revising the initial rule.

Field	Content
Model / task	gpt-4o-mini; task 03560426
M1 result	**FAILED.** The output matrix is derived by shifting the non-zero values in the input matrix upwards and leftwards, filling empty spaces with zeros. The values are preserved in their relative positions, and the highest values are prioritized in the output.
M2 result	**SOLVED.** The transformation involves shifting the colors in the matrix based on their positions while maintaining the overall structure. The colors are adjusted to create a more compact layout, with certain colors being moved to fill gaps and align with the previous patterns observed in the examples.

Table 5: Judge feedback helped over the baseline.

B.3 Correct or Near-Correct Rule, Failed Exact Matrix

This case is central to the core argument. The model can identify a useful rule but still fails at exact-match evaluation because cell-level execution is imprecise.

Field	Content
Model / task	gpt-4o-mini; task 03560426
M3 result	**FAILED.**
M3 check	**GOOD.** Compacting top rows upward is basically right.
M3 explanation	The output matrix is generated by keeping the topmost row of each colored block, shifting upward to form a compact arrangement while preserving their colors.

Table 6: Correct or near-correct rule, but failed exact matrix.

B.4 Code Verification Helped

This case shows the intended benefit of M4: executable transformations can convert an inferred rule into a precise output when the model writes correct code.

Field	Content
Model / task	gpt-5.4-mini; task 00576224
M3 result	**FAILED.** The 2x2 input is expanded by tiling it 3 times in both directions, with each original cell becoming a 1x1 tile repeated in a 3x3 arrangement of the whole 2x2 pattern.
M4 result	**SOLVED.** The previous rule failed because the output is not a checkerboard of flipped copies. The training outputs show a plain 3x3 tiling of the 2x2 input pattern, but with one extra transformation: every odd-numbered tile row is horizontally reversed, while tile columns are not reversed. That is why the first row repeats as 8,6 / 8,6 / 8,6, but the third output row becomes 6,8 / 6,8 / 6,8 instead of staying 8,6.

Table 7: Code verification helped.

B.5 Code Verification Was Stricter Than Direct Output Prediction

This case shows why M4 can underperform M3. The code must reproduce all training outputs exactly, whereas direct matrix prediction can occasionally succeed with an incomplete rule.

Field	Content
Model / task	gpt-4o-mini; task 009d5c81
M3 result	**SOLVED.** The output matrix is generated by filling the interiors of 2-bordered rectangles with specific colors based on the frame. The top and bottom frames are filled with color 3, while the middle frames are filled with color 2, preserving the borders and any inner 2-cells as holes.
M4 result	**FAILED.** The previous transformation failed because it did not accurately identify and fill the interiors of the 2-bordered rectangles based on their specific sizes and shapes. The expected output requires that the largest frames be filled with color 3, the medium frames with color 2, and the smallest frames with color 8, while preserving the borders and any inner holes.

Table 8: Code verification was stricter than direct output prediction.

C Screenshots

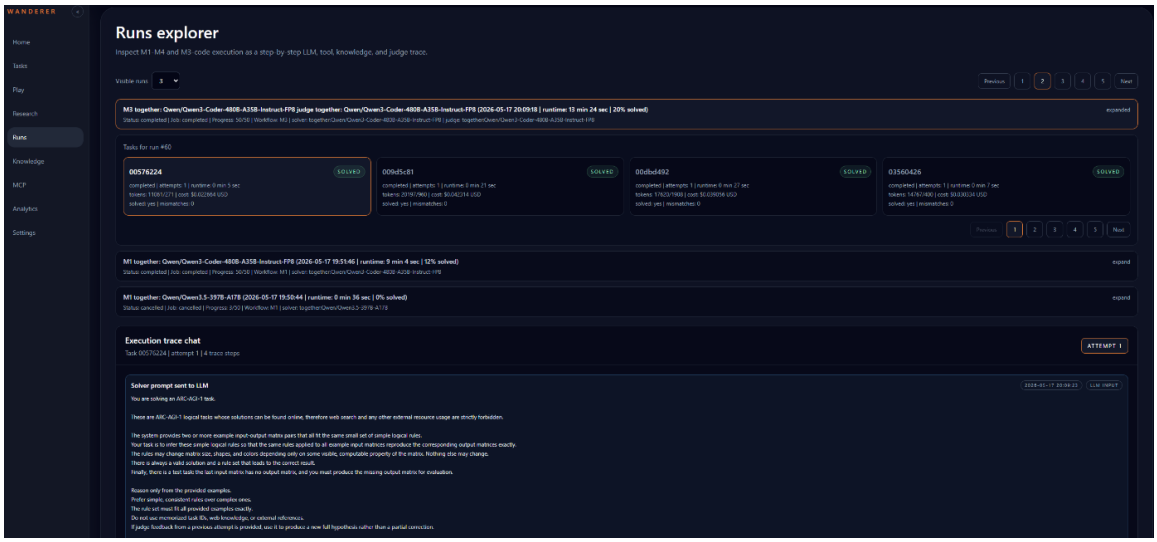

Figure 7: The conducted runs shown on the ARC-AGI interface – M3 workflow for the Qwen3-Coder-480B model.

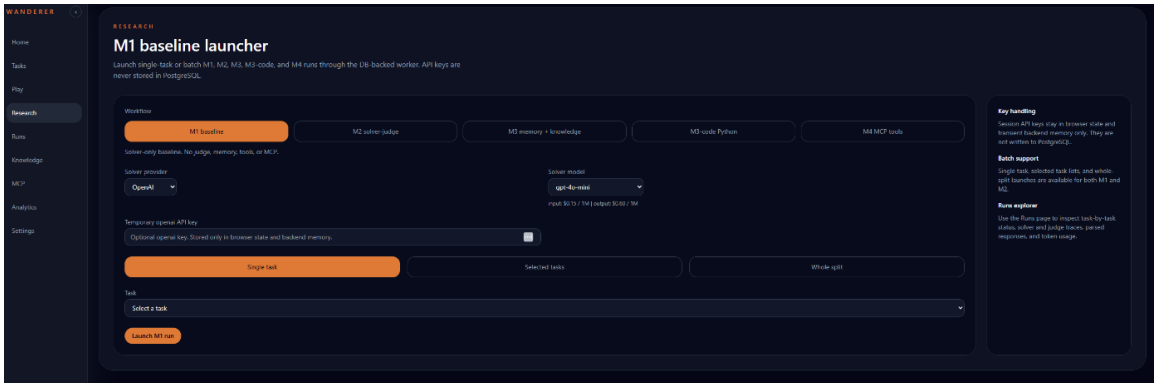

Figure 8: The experiment launcher interface for all workflows with provider and model selection options.

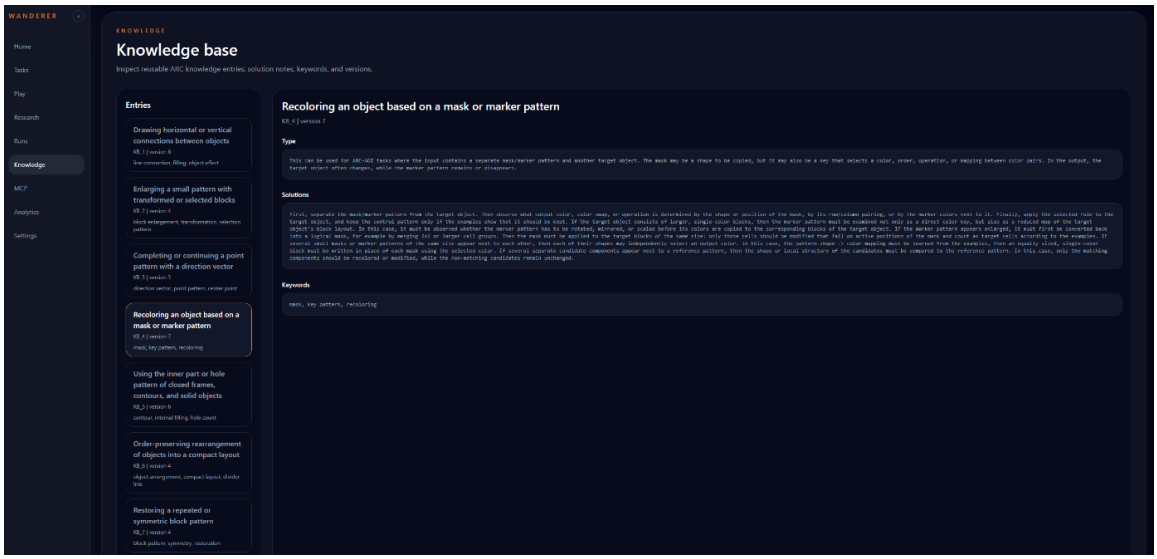

Figure 9: The accumulated knowledge base entries, highlighting KB_4, version 7.

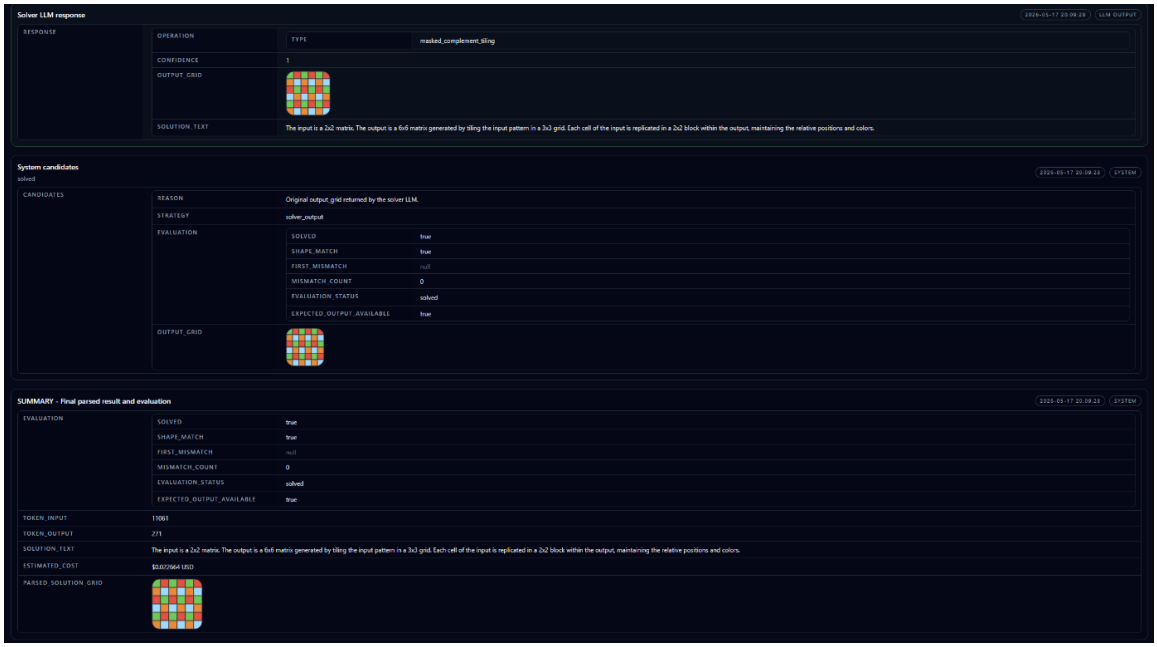

Figure 10: Correctly solved task with the M3 workflow.

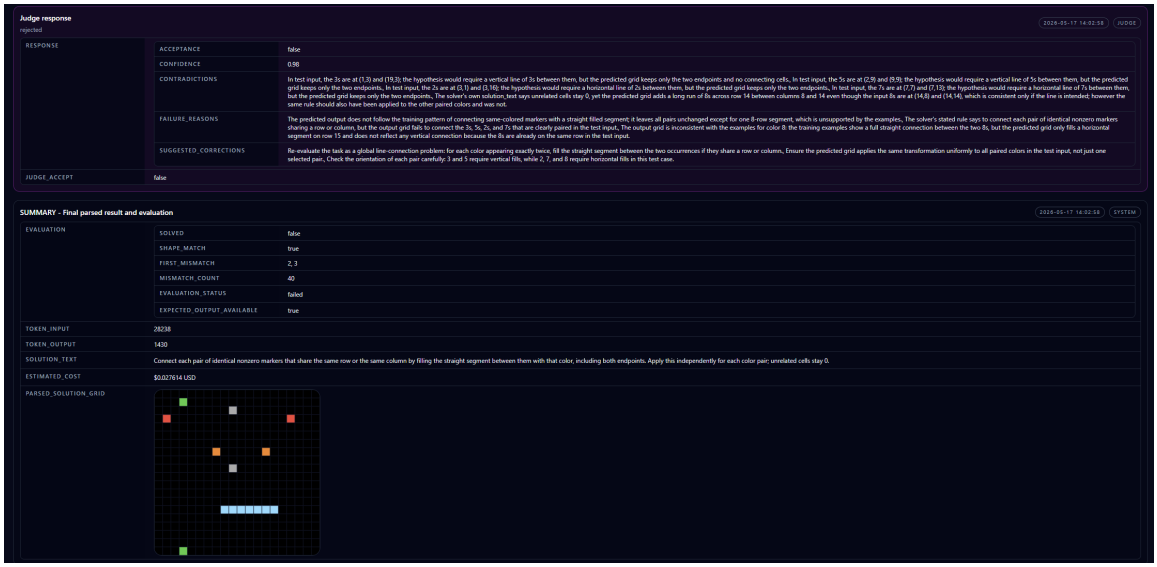

Figure 11: Judge response in the M2 workflow.

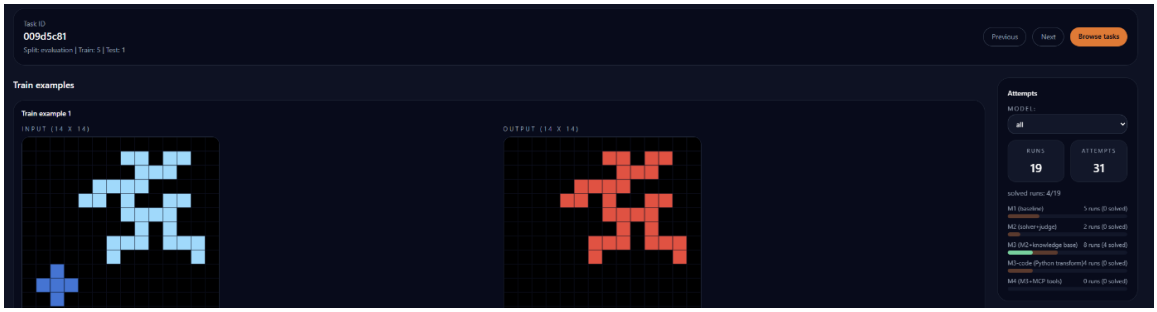

Figure 12: The summary of attempts for task 009d5c81; only the M3 runs result in correct solutions.

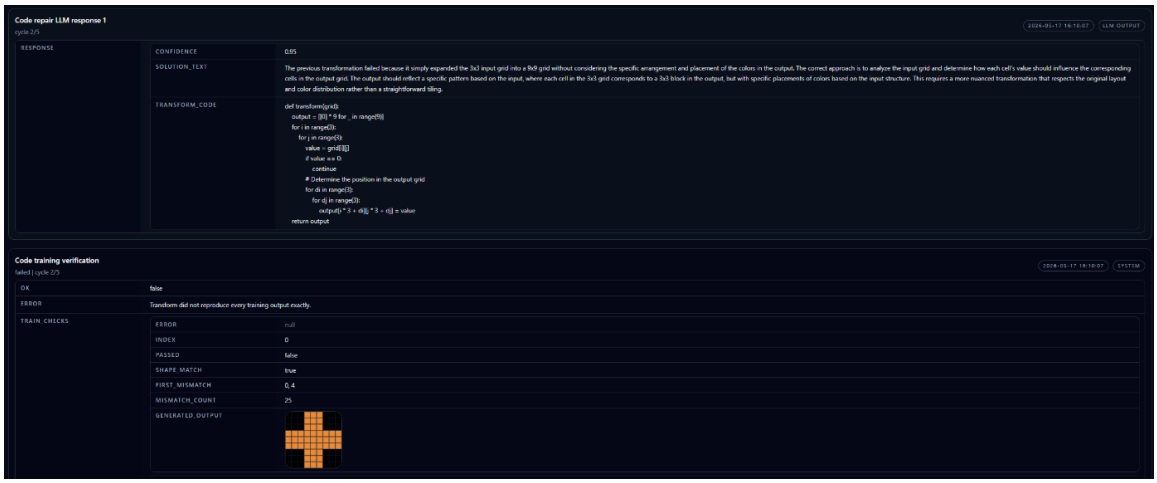

Figure 13: Code repair response in the M4 workflow.