Bongards at the Boundary of Perception and Reasoning: Programs or Language?

Cassidy Langenfeld (cml297@cornell.edu)[1], Claas Beger[1], Gloria Geng[1], Wasu Top Piriyakulkij[1], Keya Hu[2], Yewen Pu[3] & Kevin Ellis[1]
[1]Cornell University
[2]Shanghai Jiao Tong University
[3]Nanyang Technological University

Abstract

Vision-Language Models (VLMs) have made great strides in everyday visual tasks, such as captioning a natural image, or answering commonsense questions about such images. But humans possess the puzzling ability to deploy their visual reasoning abilities in radically new situations – a skill rigorously tested by the classic set of visual reasoning challenges known as the Bongard problems. We present a neurosymbolic approach to solving these problems: given a hypothesized solution rule for a Bongard problem, we leverage LLMs to generate parameterized programmatic representations for the rule and perform parameter fitting using Bayesian optimization. We evaluate our method on classifying Bongard problem images given the ground truth rule, as well as on solving the problems from scratch.

Keywords: Visual Reasoning; LLMs; Program Induction

Introduction

Visual reasoning lives at the boundary of perception and thinking. For this reason, it is a central object of study in artificial intelligence and cognitive science (Andreas et al., 2016; Ullman, 1987, *inter alia.*). Vision-Language Models (VLMs) have made great strides in everyday visual tasks, such as captioning a natural image, or answering commonsense questions about such images. But humans possess the puzzling ability to deploy their visual reasoning abilities in radically new situations, far outside everyday experience (LeGris et al., 2025), ranging from visualizing curved geometry to making sense of cubist artwork. How close is AI to capturing these abilities?

As a microcosm of this broadly-general visual reasoning, we turn to the Bongard problems (BPs), a few-shot visual classification dataset from 1970 (Bongard & Hawkins, 1970). Despite their age, the BPs have been studied by each successive wave of AI, from symbolic to probabilistic to neural to LLMs (Depeweg et al., 2024; Foundalis, 2006; Maksimov, 1975; Nie et al., 2020; Saito & Nakano, 1996; Wüst et al., 2025). The BPs involve perceiving novel visual features on-the-fly, such as the 'neck' in Figure 1 A(i), and then reasoning about those features to discriminate between two categories. Unlike visual reasoning tasks such as Raven's Progressive Matrices, the essence of a BP is to fluidly define new perceptual primitives for each problem, forcing learners to generalize precisely at the boundary of perception and reasoning.

Through a close examination of the BPs, our study explores how two different modes of reasoning – reasoning over programs and over natural language – can complement each other and push us closer to a solution for difficult concept learning tasks.

In total, we contribute the following:

1. An evaluation of State-of-the-Art VLMs on two distinct tasks: classifying BP images when the ground truth rule is present and solving the BPs by eliciting ground truth rules.
2. A new model that leverages the synthesis of parameterized programs alongside natural language reasoning to enable the selection of correct natural language rules.
3. An analysis of human data to compare VLMs and our model against human behavior.

The Bongard Problems

The Bongard problems (BPs) are a set of abstract visual reasoning puzzles first presented in Mikhail Bongard's 1970 book *Pattern Recognition* (Bongard & Hawkins, 1970). Each BP consists of twelve total images – six examples of a positive concept and six examples of a negative concept. The negative concept may be the simple negation of the positive concept, or it may be a different concept entirely. Furthermore, unlike reasoning puzzles like Raven's Progressive Matrices (Raven & Raven, 2003), there is no sequential or causal relationship between the different positive and negative examples; each image is independently an example of the underlying positive or negative concept. A solution to a BP is a natural language rule that completely separates the positive examples from the negative examples (Hofstadter, 1999); for instance, "has a smooth border" would be an acceptable solution to BP #9 shown in Figure 1. Although BPs are intentionally designed puzzles created with particular solutions in mind, multiple correct solutions may exist, as long as they properly separate the positive and negative examples (Depeweg et al., 2024).

The concepts tested by the BPs are generally abstract and geometric in nature. Some of these concepts, such as "triangles" vs. "quadrilaterals" in the solution of several BPs, may already be familiar to the solver. Others, like the aforementioned solution to BP #19 (horizontal "neck" vs. vertical "neck"), involve a combination of concepts that the solver is not likely to have encountered before. Furthermore, the particular features of an image that are relevant to the correct solution vary greatly between BPs, making them resistant to solution via image preprocessing.

The original Bongard problems were designed to be solved by humans in sequence, and give a natural curriculum that

In G. Boleda, I. Dautriche, M. Maldonado, C. Saldana and T. Torrent (Eds.), *Proceedings of the 48th Annual Conference of the Cognitive Science Society*

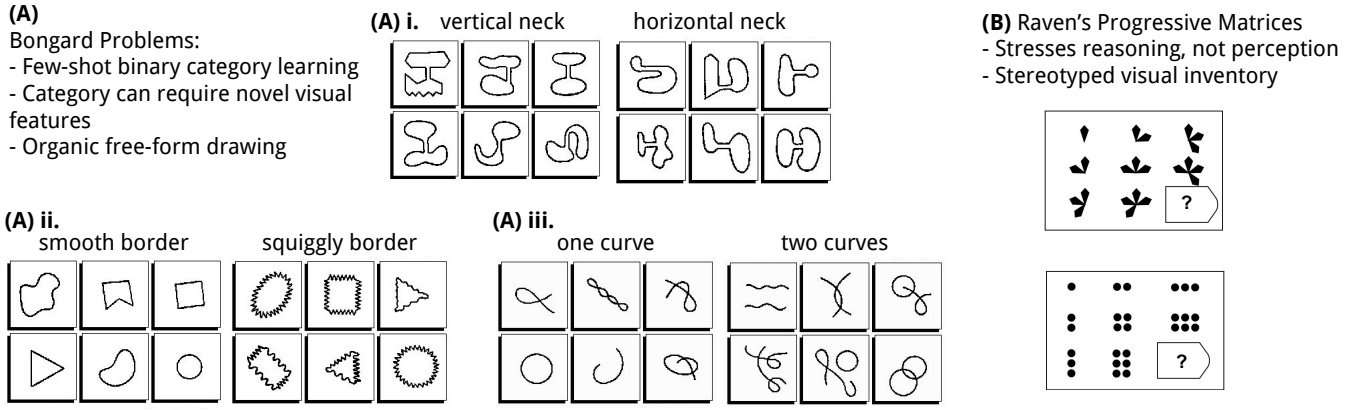

Figure 1: (A) Example Bongard problems, each of which consists of 6 positive examples (left drawings) and 6 negative examples. The natural language descriptions shown above each category are not provided to the learner. (B) In comparison, the well-known Raven's Progressive Matrices trade perceptual richness for deeper logical composition.

gradually increases problem complexity while introducing new concepts one-by-one. The system that we describe exploits the curriculum structure of the Bongard problems.

Related Work

Human Performance on BPs. The BPs test a wide variety of unfamiliar concepts, but human performance is generally strong in spite of these challenges: Wüst et al. (2025) find that the average human solves approximately 47 of BPs #2-#100, while the top 5 human solvers averaged approximately 63 problems.

Automatically Solving BPs. There is a long history of attempting to solve the BPs in AI (Foundalis, 2006; Maksimov, 1975; Saito & Nakano, 1996). Recent attempts to solve the BPs have included Bayesian inference over a formal language (Depeweg et al., 2024) and program synthesis coupled with inductive logic programming (Sonwane et al., 2021), as well as attempts to reimagine the BPs as a more traditional ML-based classification task (Raghuraman et al., 2023), an RL task setting (Youssef et al., 2022), or as inspiration for new datasets that test the reasoning capabilities of VLMs (Nie et al., 2020; Wu et al., 2023). An evaluation of VLMs on the Bongard problems was notably absent until Wüst et al. (2025), which tested a number of VLMs on solving the Bongards, as well as a number of variations on the Bongard task. The best tested model, o1 (Jaech et al., 2024), solved 43 BPs – a significant improvement over previous attempts to solve BPs automatically that still falls short of average human performance.

Reasoning with Code. As LLMs emerge as ever more capable tools for code generation (Y. Li et al., 2022; Novikov et al., 2025) and as they continue to struggle with complex reasoning tasks, many different systems for generating executable code for solving reasoning problems have been proposed (Gao et al., 2023; C. Li et al., 2023). For these systems, the answer to a reasoning problem is a program that, when run on an appropriate input, produces the desired answer. Reasoning using code has the clear advantage of being exact, interpretable, and (in most cases) deterministic, and so approaches that leverage LLMs' natural language ability to generate hypotheses and then translate these hypotheses into code are increasingly common (Wang et al., 2024). However, formal programs have an inherent expressivity problem: there are reasoning problems that can be easily formulated in natural language yet are difficult or impossible to express in code.

Induction and Transduction. The strengths and weaknesses of reasoning over both formal programs and natural language indicate room for hybrid approaches. W.-D. Li et al. (2024) formulate the problem as a question of induction vs. transduction. Induction, here corresponding to the program induction approach to reasoning, is defined as the paradigm in which, before predicting outputs for the test examples, the learner must explicitly construct a function that produces the correct outputs given the training examples as inputs. On the other hand, transduction, here corresponding roughly to test-time training or the Chain of Thought (CoT) (Wei et al., 2022) approach to reasoning, outputs a prediction for the test examples given the training examples, without explicitly searching for a latent function. For the Abstraction and Reasoning Corpus (ARC) (Chollet, 2019), another task which, like the BPs, requires pattern recognition and conceptual generalization, the two reasoning paradigms were found to be complementary (W.-D. Li et al., 2024), with induction and transduction excelling at different problems. Drawing inspiration from this work, we also combine programmatic and CoT reasoning to solve the BPs, but the hand-drawn and inexact nature of the BPs leads us to synthesize parameterized programs instead of exact ones.

Method

We experiment with two different tasks:

1. **Verification**: Given 5 positive/negative image pairs for a

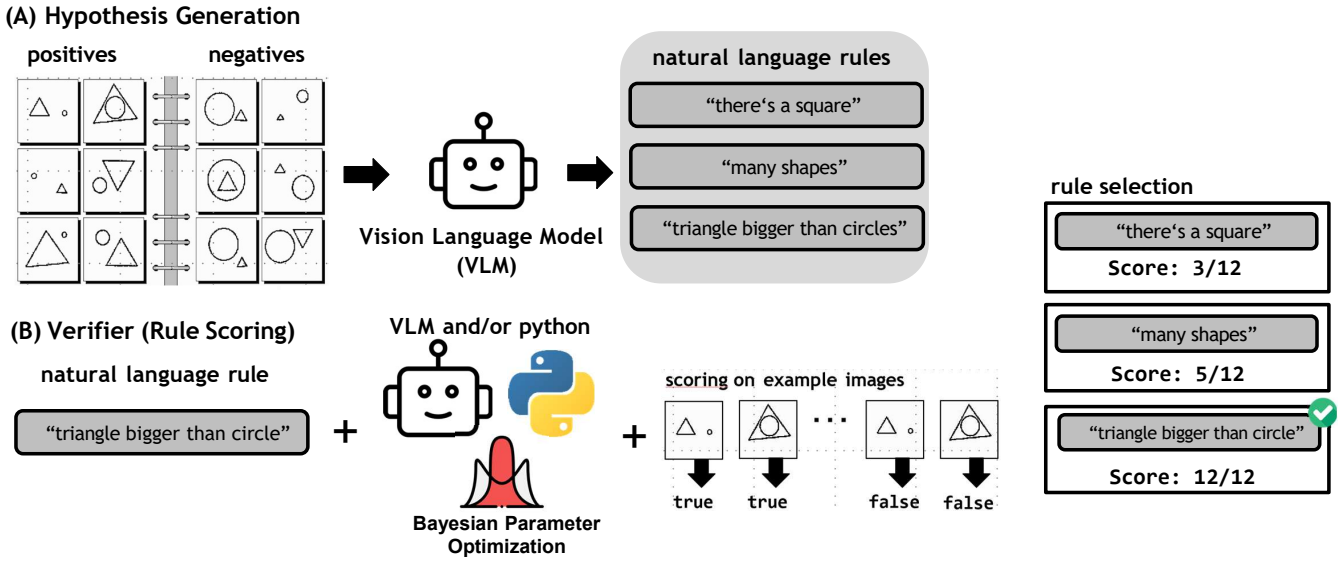

Figure 2: Our system comprises (A) a hypothesis generator that samples possible rules from a VLM and (B) a verifier that combines natural language and code to score and select the best rules.

BP as training examples *along with the ground truth rule*, classify the held-out positive/negative image pair.

2. **Solution**: Given all BP images, output a rule that correctly distinguishes positive examples from negative ones.

To more accurately measure performance on the verification task, we repeat the task six times, holding out the positive and negative images at index 0 to 5 exactly once.

System Overview

We propose a BP solver with two main components: a hypothesis generator that generates several possible rules and a verifier that determines which of these hypothesized solutions are correct (Figure 2). The verification task tests the verifier only: the ground truth rule and BP images are provided as input to the verifier, which then scores that rule. In contrast, for the solution task, our hypothesis generator produces several candidate rules, which are then scored by the verifier.

Hypothesis Generator. We sample possible solutions from a VLM, providing as input all positive and negative images for the BP and three sample rules drawn from other BPs. Because the Bongard problems were designed to be solved by humans sequentially, and have a natural curriculum ordering, we sample 6 rules given the rules of the three previous BPs as in-context examples. For increased diversity we sample another 6 rules with the solutions of three random BPs as in-context examples.

Verifier. The correctness of these hypothesized rules is judged by the verifier, which combines natural language and Python programs to score each proposed rule. Given a set of training examples X_{train}, for each image-label pair (x_{train}, y_{train}) and a rule, the verifier first attempts to synthesize a program π such that $\pi(x_{train}) = y_{train}$. A candidate program's score is then

$$score(\pi) = \sum_{(x_{\text{train}}, y_{\text{train}}) \in X_{\text{train}}} \frac{\mathbb{1}\{\pi(x_{train}) = y_{train}\}}{|X_{train}|}$$

Let P denote the set of all candidate programs. The set of *accepted programs* A is defined as

$$A = \{\pi \in P \mid score(\pi) = \max_{\phi \in P}(score(\phi)) \wedge score(\pi) \geq 0.9\}$$

If A is nonempty, then programs will be used to determine the labels of the test images, as described in the following sections. Otherwise, the verifier switches to the CoT approach, using as the output label the result of prompting the VLM with positive and negative examples provided as in-context examples.

Verification with Programs. The verifier takes a rule, 5 positive and negative image pairs, suggested method stubs, and in-context examples as input. *Suggested method stubs* are function signatures for helper methods for detecting a specific object (e.g., *detect_triangle*), auto-generated by asking the model which objects are most important to detect based on the rule. For the in-context example, we use Retrieval Augmented Generation (RAG) (Lewis et al., 2020) in order to improve the quality of VLM-generated code. We generate a set of programs for 59 different Bongard problems and use embedding similarity between the currently proposed rule[1] and the ground truth rules associated with each of the 59 problems (excluding the current BP) to select the most relevant example

[1]For the verification task, this will always be the ground truth rule for the current problem.

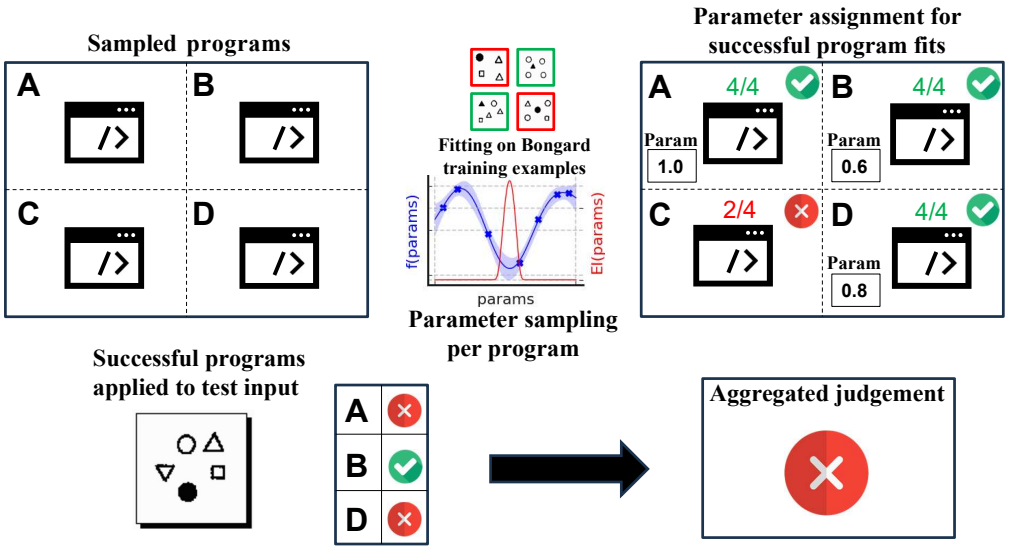

Figure 3: Overview of the program synthesis module of our verifier. Programs are sampled and undergo parameter fitting via Bayesian optimization. If we can successfully optimize programs (i.e., find programs that score at least 0.9 on the training examples), the highest-scoring programs are evaluated on the test examples, with the majority label from these evaluations serving as the output label.

and provide it as the in-context example in our prompt. We sample n programs from the VLM with these inputs.

For the inductive component of our verifier, the objective is to synthesize a 'classify_image' method that, given an input image and the additional parameters explained in the following section, will return the binary image classification.

Parameterized Programs and Optimization. Unlike grid-based datasets like ARC, the BPs are organic, hand-drawn figures whose categories are defined by approximate rather than exact properties. Humans have an intuitive idea of when an image belongs to a particular concept, but expressing this as a rule in a general-purpose formal language is challenging. For example, in BP #11 spotting the difference between 'elongated' and 'compact' shapes is not overly challenging, nor is calculating the 'circularity' of an object or observing that the class of 'compact' objects is more similar to circles (see Figure 4). However, the exact numerical dividing line between 'elongated' and 'compact' figures would be exceptionally difficult to determine through guesswork, making it impractical to require our VLM to attempt to synthesize this value along with the rest of the program. Instead, we leverage the pretraining knowledge of the VLM to specify a range of possible values for a given parameter and perform just 15 steps of Bayesian optimization (Frazier, 2018; Shahriari et al., 2015) to find the value that maximizes the program score. The parameters that require optimization are determined by the VLM and are given as additional inputs to the synthesized 'classify_image' function. If optimizing a given program does not result in a perfect score on the training examples, the VLM is prompted to revise the program once, and the new program undergoes the optimization process.

Should the verifier successfully synthesize one or more programs that obtain a score ≥ 0.9 on the training data, we obtain an output classification for each held-out example by running all programs that have achieved the maximum score for the problem on the example and outputting the majority label (Figure 3). If no such program was synthesized, our verifier instead uses transduction to predict the output labels of the test examples. Given the training examples as in-context examples, the VLM uses Chain of Thought (CoT) reasoning (Wei et al., 2022) to label each held-out example as either positive or negative.

Results

Baselines

We evaluate GPT-4o and Claude 3.7 Sonnet on all three tasks. Full results are available in Table 1. For the solution task, two human raters judged whether model outputs were correct or incorrect, with no partial credit awarded. A rule is counted as correct only if both raters agreed on its correctness. We find that Claude's performance is stronger on all three tasks. This is especially evident on the solution task, where the accuracy of Claude with CoT is just below the average human performance on the BPs reported by Wüst et al. (2025) (47 correct).

Verification Results

We experiment with both Claude and GPT-4o for both the program induction and transduction components of our verifier. Table 1 compares the performance of our method with a version of the verifier that ablates the natural language component, as well as with the performance of the base VLMs. For all results in Table 1, the number of programs synthesized

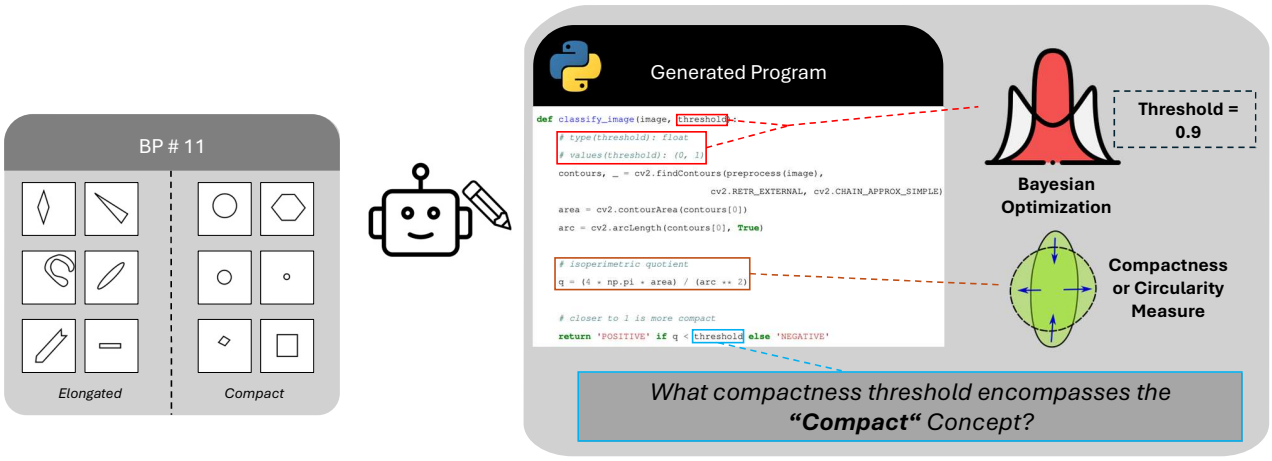

Figure 4: Solving 'Elongated' vs. 'Compact' requires a parameter threshold optimized via Bayesian optimization.

per problem was 10.

The Claude model is able to verify a similar number of BPs using either programmatic verification or CoT on its own. However, as highlighted in Table 2, which analyzes the average performance of each verifier on different categories of BPs (as introduced in Wüst et al. (2025)), these two methods differ in the types of problems they are able to solve. Programmatic verification is advantageous for problems dealing with similarity between objects and with spatial concepts. This is unsurprising given the way in which spatial reasoning problems lend themselves to geometric or mathematical reasoning that can easily be encoded as formal programs. Similarly, CoT outperforms programmatic verification on tasks requiring high-level conceptual knowledge; the gap between the two methods on number problems appears to be due to the fact that many number BPs require each image to be broken down into a unique set of subparts that may be difficult to encode as a program. The strong results of both program verification and CoT, along with the complementary nature of the different problem categories they solve, lead to our combined method achieving improved results in nearly every BP category.

With GPT-4o, program verification underperforms CoT by a wider margin and the categories solved overlap more, so the combined method gains less. This suggests the hybrid approach's effectiveness depends partially on the underlying model's program synthesis ability.

Solution Results

We test our full BP solving system with Claude and GPT as hypothesis generators. For the verification component, we utilize the verifier described above. With Claude as the VLM, our system improves to **51** problems solved, exceeding the average human accuracy reported by Wüst et al., 2025. Utilizing our method also improves on 4o's score, although it remains far below human accuracy.

Comparison with Human Data. Figure 5 illustrates examples of where human and model performance shows strong (dis)agreement. Problems that both humans and our system perform well on tend to be conceptually simple, while the problems they both fail on either test unusual combinations of visual concepts (BP #44, 'small circles on different arcs') or present a very simple ground truth concept in an intentionally complicated manner (BP #90, 'three' vs. 'five'). The problems where humans succeed and our system fails often either require perceptual reasoning capabilities that VLMs struggle with, such as depth estimation (BP #46, 'triangle on top of circle'), or require the solver to compose a few concepts the VLM can independently recognize, highlighting a deficit in compositional reasoning. BP #75 ('triangle on concave side of arc') is one example of VLMs' struggle with compositional reasoning; the hypothesis generator was unable to generate the correct concept even though the average VLM performance on earlier problems involving concavity (BP #4) and triangles (BP #6, #10) was similar to or exceeded that of humans.

Table 1: "Verification" is classifying images given the ground truth label, "Solution" is outputting the correct natural language rule, and "Inversion" is the verification problem with positive and negative examples swapped. Human data from Wüst et al., 2025. For consistency with human data, only solution results for BPs #2–#100 are reported. We report accuracy for verification and inversion tasks and number of problems solved for the solution task.

Model	Verif.	Solved	Inv.
Human Avg.	–	47	–
GPT-4o + CoT	0.775	24	0.771
GPT-4o + programs	0.727	–	–
GPT-4o + both	**0.79**	**31**	–
Claude 3.7 Sonnet + CoT	0.835	44	0.838
Claude 3.7 Sonnet + programs	0.821	–	–
Claude 3.7 Sonnet + both	**0.865**	**51**	–

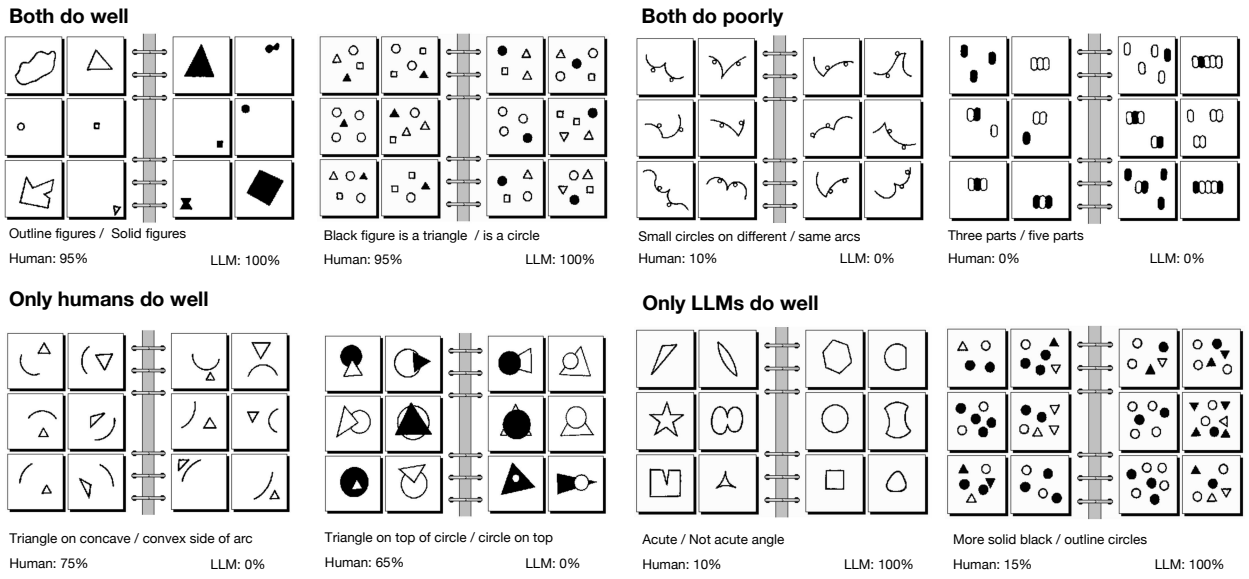

Figure 5: Examples of problems with similar/dissimilar human/model performance

Table 2: Performance across BP categories from Wüst et al., 2025. "+ programs" indicates that our method for verification with programs was used, while "+ both" indicates that both natural language and programs were used by the verifier. GPT refers to GPT-4o, Claude refers to Claude 3.7 Sonnet. Categories stand for Concept, Number, Same Size, Spatial.

Model	Conc.	Num.	Same	Size	Spat.
GPT + CoT	**0.867**	0.794	**0.821**	0.931	**0.700**
GPT + prog.	0.773	0.689	0.810	0.875	0.667
GPT + both	0.846	**0.833**	0.702	**0.958**	0.681
Claude + CoT	0.904	**0.878**	0.845	**0.972**	0.744
Claude + prog.	0.858	0.756	**0.893**	0.931	0.792
Claude + both	**0.919**	0.833	**0.893**	0.958	**0.815**

Memorization. The strong out-of-the-box performance of VLMs with CoT raises questions about the extent to which the solutions to the BPs have been memorized. To check for memorization, we invert the verification task: the positive concept and associated images are now negative examples, and likewise for the negative concept. We find this has little impact on model performance (Table 1), suggesting memorization does not fully explain VLM performance.

Discussion

Natural language and code as complementary reasoning approaches. Our analysis of the BP categories solved by natural language and code revealed that the shortcoming of one method was often the strength of the other. In particular, reasoning over natural language appears to be advantageous when high-level conceptual thinking is required, while reasoning over formal programs is preferable for exact calculations.

Optimizable programmatic solutions to visual problems. Generating *parametric* programs (and optimizing the parameters) proved important. We note that this is conceptually similar to generating the structure of a probabilistic program and then optimizing its parameters (Wong et al., 2025).

Rules and Interpretability. Prior work such as Wüst et al., 2025 largely focuses on whether models can articulate abstract rules in natural language and then apply them. While this is informative, it leaves open the question of whether the stated rule actually governs the model's predictions. A model might claim to separate "spirals from non-spirals," yet rely on spurious cues in practice. By contrast, formalizing rules as executable programs provides a verifiable link between reasoning and outcome. Their structure also exposes intermediate operations that reveal how a solution is implemented and make failures easier to diagnose.

Conclusion

Formal programs and natural language are two distinct mediums for representation and reasoning, each with their own strengths and weaknesses.

Through our investigation, we found that combining the two allows us to reason more accurately about the visual concepts central to these puzzles, allowing us to exceed average human accuracy—for the first time, to the best of our knowledge. Our hope is that this model, which generates concepts in language and code, is a step toward AI systems that acquire and use new perceptual concepts in humanlike ways.

References

Andreas, J., Rohrbach, M., Darrell, T., & Klein, D. (2016). Neural module networks. *CVPR*.

Bongard, M., & Hawkins, J. (1970). *Pattern recognition*. Spartan Books.

Chollet, F. (2019). On the measure of intelligence.

Depeweg, S., Rothkopf, C. A., & Jäkel, F. (2024). Solving bongard problems with a visual language and pragmatic constraints. *Cognitive Science*, *48*(5), e13432.

Foundalis, H. E. (2006). *Phaeaco: A cognitive architecture inspired by bongard's problems.* [Doctoral dissertation, ProQuest Information & Learning].

Frazier, P. I. (2018). A tutorial on bayesian optimization. *arXiv preprint arXiv:1807.02811*.

Gao, L., Madaan, A., Zhou, S., Alon, U., Liu, P., Yang, Y., Callan, J., & Neubig, G. (2023). Pal: Program-aided language models. *International Conference on Machine Learning*, 10764–10799.

Hofstadter, D. R. (1999). *Gödel, escher, bach: An eternal golden braid*. Basic books.

Jaech, A., Kalai, A., Lerer, A., Richardson, A., El-Kishky, A., Low, A., Helyar, A., Madry, A., Beutel, A., Carney, A., et al. (2024). Openai o1 system card. *arXiv preprint arXiv:2412.16720*.

LeGris, S., Vong, W. K., Lake, B. M., & Gureckis, T. M. (2025). A comprehensive behavioral dataset for the abstraction and reasoning corpus. *Scientific Data*, *12*(1), 1380.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., et al. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. *Advances in neural information processing systems*, *33*, 9459–9474.

Li, C., Liang, J., Zeng, A., Chen, X., Hausman, K., Sadigh, D., Levine, S., Fei-Fei, L., Xia, F., & Ichter, B. (2023). Chain of code: Reasoning with a language model-augmented code emulator. *arXiv preprint arXiv:2312.04474*.

Li, W.-D., Hu, K., Larsen, C., Wu, Y., Alford, S., Woo, C., Dunn, S. M., Tang, H., Naim, M., Nguyen, D., et al. (2024). Combining induction and transduction for abstract reasoning. *arXiv preprint arXiv:2411.02272*.

Li, Y., Choi, D., Chung, J., Kushman, N., Schrittwieser, J., Leblond, R., Eccles, T., Keeling, J., Gimeno, F., Dal Lago, A., et al. (2022). Competition-level code generation with alphacode. *Science*, *378*(6624), 1092–1097.

Maksimov, V. (1975). A system for teaching the classification of geometric patterns. *Modeling of Learning and Behavior*, 29–120.

Nie, W., Yu, Z., Mao, L., Patel, A. B., Zhu, Y., & Anandkumar, A. (2020). Bongard-logo: A new benchmark for human-level concept learning and reasoning. *Proceedings of the 34th International Conference on Neural Information Processing Systems*.

Novikov, A., Vũ, N., Eisenberger, M., Dupont, E., Huang, P.-S., Wagner, A. Z., Shirobokov, S., Kozlovskii, B., Ruiz, F. J., Mehrabian, A., et al. (2025). Alphaevolve: A coding agent for scientific and algorithmic discovery. *arXiv preprint arXiv:2506.13131*.

Raghuraman, N., Harley, A. W., & Guibas, L. (2023). Support-set context matters for bongard problems. *Transactions on Machine Learning Research*.

Raven, J., & Raven, J. (2003). Raven progressive matrices. In *Handbook of nonverbal assessment* (pp. 223–237). Springer.

Saito, K., & Nakano, R. (1996). A concept learning algorithm with adaptive search. In *Machine intelligence 14: Applied machine intelligence* (pp. 347–363).

Shahriari, B., Swersky, K., Wang, Z., Adams, R. P., & De Freitas, N. (2015). Taking the human out of the loop: A review of bayesian optimization. *Proceedings of the IEEE*, *104*(1), 148–175.

Sonwane, A., Chitlangia, S., Dash, T., Vig, L., Shroff, G., & Srinivasan, A. (2021). Using program synthesis and inductive logic programming to solve bongard problems. https://arxiv.org/abs/2110.09947

Ullman, S. (1987). Visual routines. In *Readings in computer vision* (pp. 298–328). Elsevier.

Wang, R., Zelikman, E., Poesia, G., Pu, Y., Haber, N., & Goodman, N. D. (2024). Hypothesis search: Inductive reasoning with language models. *The Twelfth International Conference on Learning Representations*.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., & Zhou, D. (2022). Chain-of-thought prompting elicits reasoning in large language models. *Advances in Neural Information Processing Systems*, *35*, 24824–24837.

Wong, L., Collins, K. M., Ying, L., Zhang, C. E., Weller, A., Gerstenberg, T., O'Donnell, T. J., Lew, A. K., Andreas, J., Tenenbaum, J. B., & BrookeWilson, T. (2025). Modeling open world cognition as on-demand synthesis of probabilistic models. *NeurIPS 2025 Workshop on Bridging Language, Agent, and World Models for Reasoning and Planning*. https://openreview.net/forum?id=3Ti0IzHuOI

Wu, R., Ma, X., Zhang, Z., Wang, W., Li, Q., Zhu, S.-C., & Wang, Y. (2023). Bongard-openworld: Few-shot reasoning for free-form visual concepts in the real world. *arXiv preprint arXiv:2310.10207*.

Wüst, A., Tobiasch, T., Helff, L., Ibs, I., Stammer, W., Dhami, D. S., Rothkopf, C. A., & Kersting, K. (2025). Bongard in wonderland: Visual puzzles that still make ai go mad? https://arxiv.org/abs/2410.19546

Youssef, S., Zečević, M., Dhami, D. S., & Kersting, K. (2022). Towards a solution to bongard problems: A causal approach. *arXiv preprint arXiv:2206.07196*.