PIXELPROOF: VISUAL QUESTION GENERATION THROUGH FORWARD–INVERSE AGREEMENT

Anonymous authors
Paper under double-blind review

ABSTRACT

From pre-training on human-written tokens to reinforcement learning on verifiable rewards (RLVR), obtaining cheaply verifiable data is the fuel for current large language models. RLVR often works in mathematics and similar domains because the labels are verifiable by construction and establish a strong self-supervision loop that bootstraps model capabilities. Constructing this loop outside math domains, especially in vision, is an interesting open challenge. We introduce PIXELPROOF, a method that takes human-input prompts and automatically synthesizes novel visual-question images via Python where labels are verifiable by the generation loop itself (without human supervision). That is, PIXELPROOF parameterizes a set of questions by 3 Python programs: (1) a Forward program that translates scene specifications into an expected answer; (2) a Renderer that renders an image; and (3) an Inverse program that progressively transforms the input image using image processing tools to derive its own verification answer independently. Comparing the answers between the Forward (what is designed) vs. Inverse programs (what is observed) self-supervises the generation. Generation can be steered toward (a) a human-defined topic e.g., music; (b) where in the image the answer's evidence lies; and (c) harder questions. As an initial demonstration of PIXELPROOF, five coding agents working across nine visual reasoning directions produced over 2,400 verified questions. Fine-tuning three open-weight models on the newly generated questions improves their performance on held-out questions by an average of $+10.7$ points, and by $+1.9$ points on average across 16 external benchmarks. PIXELPROOF shows a promising loop for automatically synthesizing visual questions for evaluating and training vision–language models. Code and data are at: projectpixelproof.github.io

1 INTRODUCTION

Decades of machine learning research point to the same lesson: data quality matters (Sutskever, 2024; Sutton, 2019). The key question is how to acquire high-quality, **verifiable** ground-truth labels cheaply, which are valuable to both *evaluation* and *training* (Guo et al., 2025). That is, a closed data-generation loop has been useful for synthesizing environments and training better agents (Wang et al., 2019; 2020a), automatically discovering weaknesses in models (Lu et al., 2025; Li et al., 2025b), and self-playing and training models without relying on human-curated datasets (Zhao et al., 2025; He et al., 2026; Guo et al., 2025).

Code generation and mathematics are easier domains to verify with predefined tests, but ensuring the validity of data in *images* is significantly more challenging. One approach for automated verification is to use vision–language models (VLMs) to judge whether a question has a clear answer (Chen et al., 2024; Li et al., 2025a). VLMs can be useful for filtering image–text data (Wang et al., 2024), but are known to have issues in their perception (Rahmanzadehgervi et al., 2024) and visual reasoning (Hu et al., 2026), as well as biases in their answers (Vo et al., 2026; Zou et al., 2026).

We introduce PIXELPROOF, an automated visual data creation framework that directly integrates answer verification into the generation of visual questions. For each question, a coding agent writes code that generates synthetic images, a *forward* program that computes each answer from the parameters used to draw the image, and an *inverse* program that must recover the same answer from

Reviewers: please read the Reviewer Guidelines (iclr.cc/Conferences/2027/ReviewerGuidelines) and the AI Policy for Reviewers (iclr.cc/Conferences/2027/AIPolicyForReviewers). If you used AI to expand, edit, or polish your review, please provide the input text to the LLM. Better still, consider skipping the LLM and submitting your original text: we, and the authors, are much more interested in your unedited thoughts than in what an LLM has to say. AI-assisted or not, you are putting your name and reputation behind your review: LLM-generated falsehoods, hallucinations or misrepresentations are subject to disciplinary action, which may include desk-rejecting all papers you have authored.

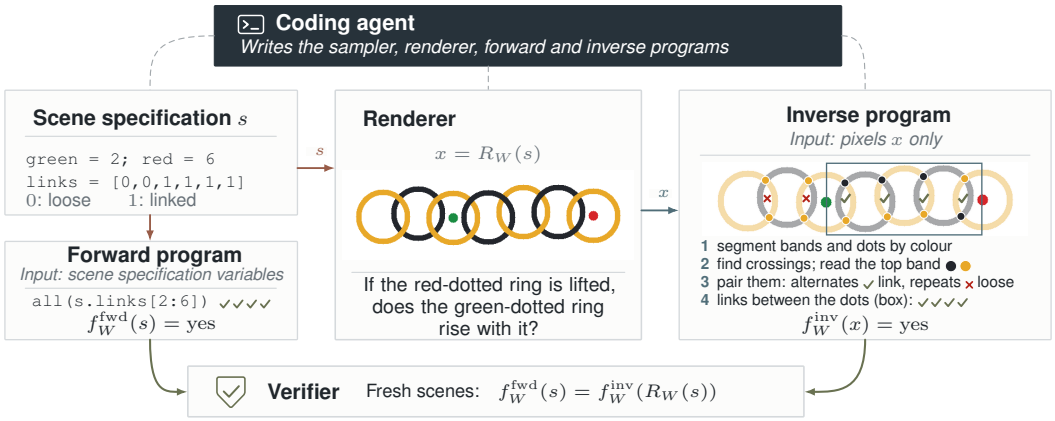


Figure 1: PIXELPROOF generates visual questions together with programs that test whether their answers can be recovered from the images. Here, the question asks whether lifting the red-marked ring also lifts the green-marked ring. The forward program answers from the specified ring connections; the inverse program reconstructs them from the image alone, shown here with its own detected crossings and junctions. Both programs correctly answer "yes".

the rendered image alone (Fig. 1). The key idea is that each answer must be recoverable from the image, not only from how the image was built.

Our main findings are:

1. Inverse program helps agents find and fix errors: for 29% of the questions that ended up in the collection, the agent had fixed its code after the inverse program failed, nearly half by changing how images were drawn or sampled (Sec. 4.3).
2. The generation process can be steered using different signals including spatial constraints on how the information needed to answer a question is distributed across an image (Sec. 4.2), and model-evaluation feedback used as a proxy for difficulty (Sec. 4.6).
3. Fine-tuning VLMs on the generated data improves their performance by $+10.7$ percentage points (pp) on held-out questions of the same types, $+8.5$ pp when an entire type of question is withheld from training, $+5.3$ pp on a harder, separately generated test set, and $+1.9$ pp on 16 external benchmarks (Sec. 4.7).

2 RELATED WORK

Visual question generation and validation. Visual question generation creates questions and answers about images for evaluation or training. Procedural approaches render images with programs and compute answers from the information used to create them: CLEVR and Super-CLEVR generate questions about synthetic scenes (Johnson et al., 2017; Li et al., 2023), and DynaMath and AlgoPuzzleVQA generate mathematical problems and puzzles (Zou et al., 2025; Ghosal et al., 2025). Other methods take answers from known image operations, such as recorded tile permutations in Jigsaw-R1 (Wang et al., 2025) or executable image transformations in EVE (Heng et al., 2026). These approaches produce many labeled examples without annotating each one. Closest to our setting, Multimodal Self-Instruct and CoSyn have language models write code that renders charts, diagrams, and abstract images, and take the answers from that code (Zhang et al., 2024; Yang et al., 2025). These pipelines correspond to PIXELPROOF without its inverse program, which recovers each answer from the rendered pixels.

Model-based approaches generate questions about existing images, or images together with questions, and validate them in several ways. SynthRL filters rewritten questions by comparing model responses with the original answers (Wu et al., 2026). VisPlay and MM-Zero obtain answers by solver consensus (He et al., 2026; Li et al., 2026), and VisionFoundry uses a VLM to check whether generated images support the intended answers (Zhou et al., 2026). PROVE generates programs that query scene graphs

derived from detailed captions, plus model-based visual checks (Prabhu et al., 2025). PIXELPROOF instead has its inverse program recover each answer from the image and compares it with the forward program's answer on new instances.

Visual programming. Visual programming uses executable programs to perform tasks involving images or videos. These programs combine perceptual operations, such as locating objects or estimating depth, with computations such as counting, comparison, and logical reasoning. NS-VQA separates learned scene parsing from symbolic reasoning (Yi et al., 2018). Later approaches use language models to generate the programs: VISPROG addresses question answering, object tagging, and image editing (Gupta & Kembhavi, 2023); ViperGPT supports visual grounding, question answering, and video reasoning (Surís et al., 2023); and CodeVQA composes visual operations for questions involving one or more images (Subramanian et al., 2023).

Beyond generating a program for an individual query, VADAR constructs reusable visual functions (Marsili et al., 2025), and ViUniT tests programs on generated images to support selection and repair (Panagopoulou et al., 2025). PIXELPROOF uses visual programming to validate each generated question: its inverse program must answer many instances of that question, and its outputs are checked against the forward program's answers before the question passes verification.

3 METHOD

In PIXELPROOF, a coding agent writes code that generates visual questions with their answers, and a second program that must recover each answer from the image alone. A question is kept only if the two agree, and the outcome of each attempt guides the next generation.

3.1 QUESTION WORLDS AND VERIFICATION

The code an agent writes is organized as a *question world*. Each world contains a visual question (such as whether or not two links in a chain are connected like in Fig. 1) in addition to code that generates many instances of that description. The question stays the same across instances, while the elements in the image, their attributes, and their arrangement vary, so that each instance is unique.

A world W generates each instance in two steps. A sampler outputs a *scene specification* s with variables describing the objects and their relationships, and a renderer turns those variables into an actual image, $x = R_W(s)$.

Two programs compute the decision needed to answer the question from different inputs. The *forward program* uses the scene specification, and the *inverse program* uses the rendered image alone. A fixed mapping converts each decision into an answer for each question text; f_W^{fwd} and f_W^{inv} denote these answer functions, including the mapping. In the ring example, the forward program reads the specified connections, and the inverse program identifies those connections only from the image. The inverse program receives only an image and may not access the scene specification or import the sampler, renderer, or forward program.

For the ring world in Fig. 1, the agent submits a sampler, a renderer, question text, a forward program, an inverse program, an optional ambiguity rule, and tests with saved examples (App. D). The sampler chooses which rings are marked and which neighbors are linked. The forward program checks whether all links between the marked rings are present. The inverse program segments the rings, reads which ring passes over the other at each crossing, and applies the same test.

A fixed program, the *verifier*, renders new scenes from fixed seeds and compares the forward and inverse answers. Scenes near a decision threshold can become unreadable after antialiasing. For example, if the question asks whether two circles overlap, circles that barely touch and circles that barely overlap can look the same in the image. As an additional quality check, the agent may declare a deterministic *ambiguity rule* that flags such scenes; the verifier recomputes it and excludes them from the comparison (Sec. F.4).

For every remaining tested scene, it requires $f_W^{\text{fwd}}(s) = f_W^{\text{inv}}(R_W(s))$. We call a world that passes these checks *pixel-level answerable*, meaning that its expected answers can be recovered from the rendered images.

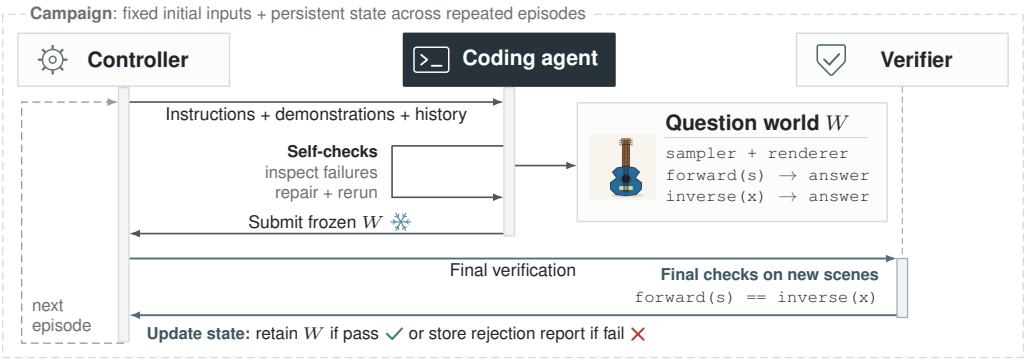


Figure 2: Sequence diagram of the generation loop in PIXELPROOF. A coding agent receives generation instructions, code for previously accepted worlds, and verification reports from earlier attempts. It then writes a new world and tests its forward and inverse programs, repairing failures while time remains. After submission, the verifier reruns the checks in a clean container and compares the two programs' answers. The controller keeps accepted worlds and passes their code, or the failure report, to the next episode.

3.2 THE AGENT-DRIVEN GENERATION LOOP

PIXELPROOF grows the collection through repeated attempts by fresh coding agents (Fig. 2). At the start of each attempt, the agent receives (a) the generation instructions; (b) three implementation demonstrations; (c) time budget of 20 minutes, and (d) information from earlier attempts in the same campaign. This information consists of the code of previously accepted worlds and, when available, the most recent rejected submission with its verification report. The agent then tries to build one new question world.

While developing the world, the agent can run *self-checks* and repair any failures it finds. These checks cover execution, reproducibility, and agreement between the forward and inverse programs, and can be rerun as often as time allows. To submit the world for verification, the agent must pass the checks and finalize its code before the session ends (Secs. D.3 and E.1).

After submission, the verifier runs the *final checks* on the frozen submission, in a clean container without network access. These are the same checks the agent could run during its session (agreement on 72 scenes from three fixed seeds, answer invariance, the ambiguity rule, and rejection of corrupted examples), but the agent can no longer change its code (Sec. D.4). A world is *accepted*, that is, added to the collection, only if it passes these checks, was properly finalized, and is not a direct copy of an earlier world (Sec. D.5). We call one agent session followed by this verification an *episode*. A *campaign* is a sequence of episodes run with the same initial inputs, coding-agent configuration, and generation instructions.

3.3 MODELS AND AGENT CONFIGURATIONS

We use five coding-agent configurations with PIXELPROOF: Sol (high), Sol (max), Luna, Opus 5, and DeepSeek V4 Flash. Sol and Luna run through Codex, Opus 5 through Claude Code, and DS-V4-Flash through OpenCode. Additional model details are reported in Tab. A3.

4 EXPERIMENTS AND RESULTS

We first test whether PIXELPROOF can scale to large numbers of questions across many question types. Then, we study how far the generation process can be steered, both toward questions whose answers depend on requested image regions and toward questions that are harder for current models. We also test whether the inverse program catches errors in the rendered images. Finally, we evaluate open- and closed-source VLMs on the generated questions and study how fine-tuning on this data affects their performance.

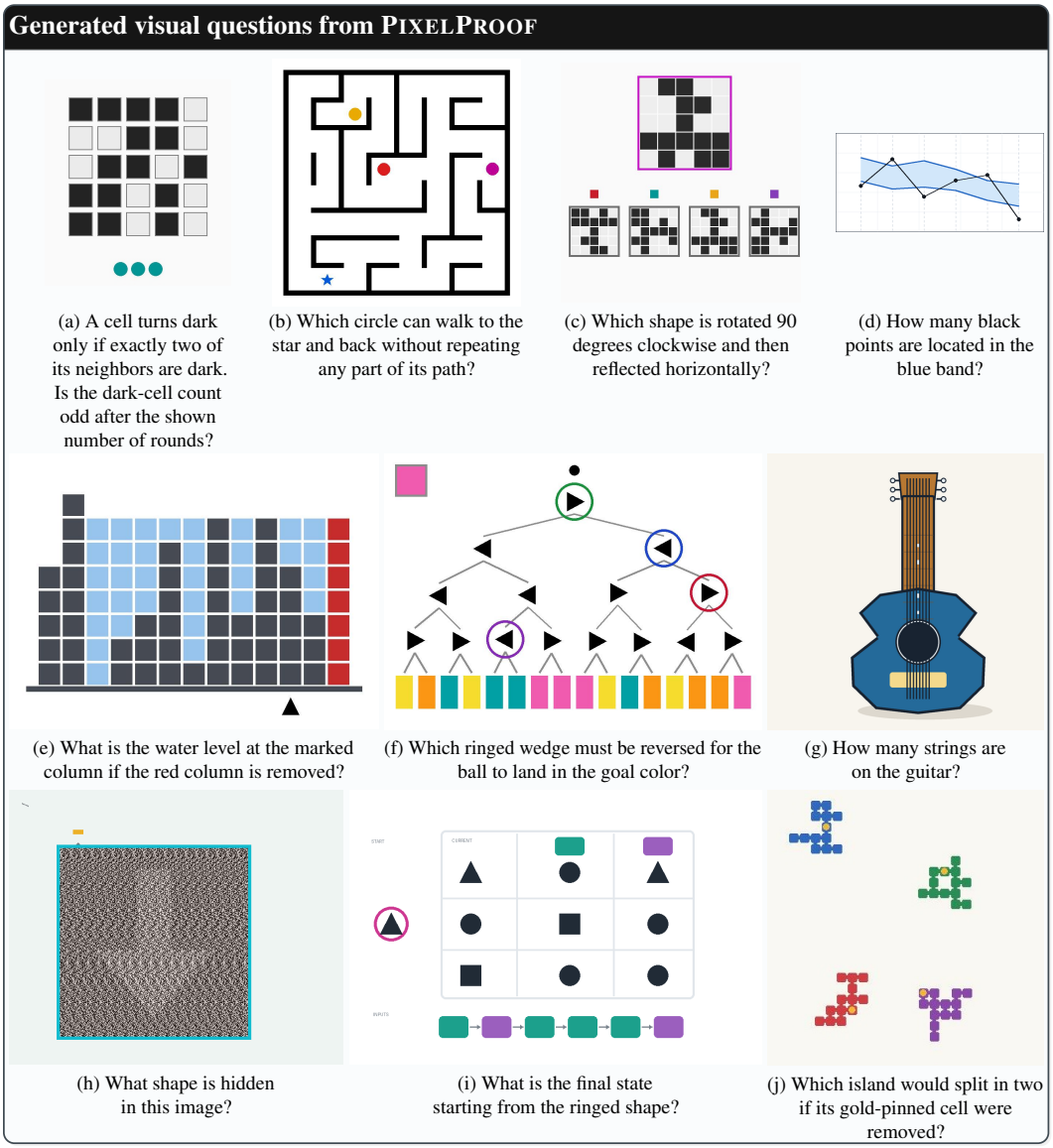


Figure 3: Selected images generated by PIXELPROOF with summarized text describing each image's question. App. A provides full information about each data sample.

4.1 CODING AGENTS REPEATEDLY GROW THE QUESTION COLLECTION WITH PIXELPROOF

A useful generation process should keep expanding the collection, not only producing a few successful questions. We therefore test whether coding agents can repeatedly construct new verified worlds across different campaigns and different directions of visual reasoning.

Experiment: By default, agents may converge on similar questions that are easiest to implement. To encourage data diversity, we seed the data generation process with nine *discovery profiles*, which contain examples of a wide range of question types (Tab. A2).

We run each of five coding-agent configurations on all nine profiles, giving 45 independent campaigns. Each configuration receives a budget of six hours of cumulative agent-session time per profile, with individual sessions capped at 20 minutes. We measure accepted worlds, their rate per episode-hour, and whether new worlds keep being added as campaigns progress. Episode-hours include agent-session and provider waiting time but exclude final verification and later analysis. We also replay each accepted world on 200 new scenes (Sec. E.5). We check whether worlds repeat one another

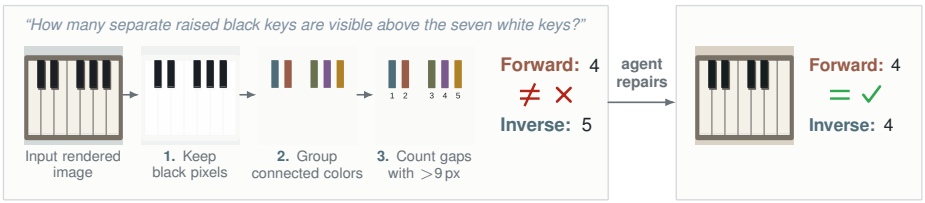


Figure 4: Left: Given a rendered input image (here, a piano octave), the inverse image-to-answer program *deterministically* processes the image and arrives at the answer 5. This does not match the output 4 of the forward program, revealing a bug in the scene-specification-to-image renderer. This mismatch provides feedback to the agent, helping it repair the renderer (right). Without the inverse program, this world would have generated a wrong label 4 for a 5-black-key image.

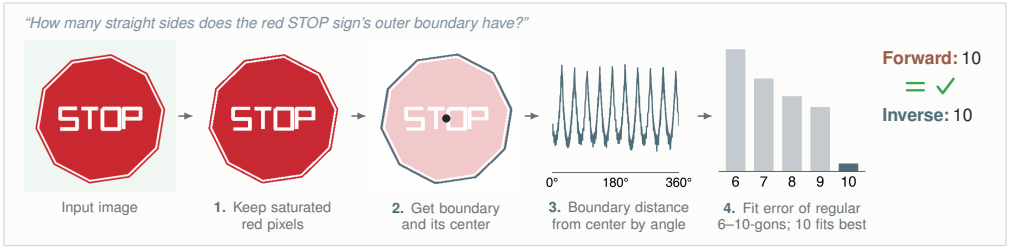


Figure 5: The agent writes an inverse program to measure the stop sign's outline from the image alone and recover its number of sides, matching the forward program's answer of 10.

exactly, with matching question texts and images, or approximately, by comparing DINOv2 image embeddings (Oquab et al., 2024) and MiniLM question embeddings (Wang et al., 2020b) (Sec. G.3).

Results: Coding agents repeatedly expanded the collection in every campaign, and 1,331 of 1,426 episodes resulted in an accepted world (Fig. 3d,g–i show examples). Generation continued throughout the six-hour budgets: for four of the five configurations, throughput was at least as high in the second half of a campaign as in the first half, with only Opus 5 slowing down modestly (Sec. G.1.1). Of the 1,331 accepted worlds, 1,301 (97.8%) also passed the 200-scene replay (Sec. G.1.2).

Worlds rarely repeat one another: we found (a) no two worlds share a question text; (b) pixel-identical images occur in only two pairs of worlds; and (c) only 16 of 845,650 pairs exceed 0.90 similarity in both image and question embeddings. Within a world, 96.3% of replay scenes are distinct (Sec. G.3.1). A full breakdown of each agent's token usage and cost is reported in Sec. E.4.

4.2 Agents Can Control Which Parts of an Image a Question Depends On

Visual questions differ in how much of the image they require. For example, counting the strings on a guitar needs only the neck of the instrument, while following a maze route needs the whole image. We investigate whether PixelProof can be steered toward changing where in the image the crucial information occurs, as a test of whether the generation process can be guided.

Experiment: We use the coding-agent configurations, discovery profiles, and campaign budgets as in Sec. 4.1. The only change is that each episode also requests one of nine spatial patterns for where the answer's evidence should lie. These patterns combine three scales (focal, regional, distributed) with three shapes (compact, pathlike, multipart). Fig. A2 shows examples of all nine patterns.

After generation, we measure which regions each answer depends on by masking parts of the image and checking whether the inverse answer changes or fails. We then measure how often the resulting spatial pattern matches the request and whether requests widen the range of spatial patterns compared with unsteered generation.

Results: Agents followed the requested spatial pattern in the vast majority of the worlds. Of the 874 replay-verified worlds, 835 (95.5%) matched the requested region size and shape (Sec. G.2.3). Spatially-steered worlds also covered a broader range of region sizes and shapes. Across matched

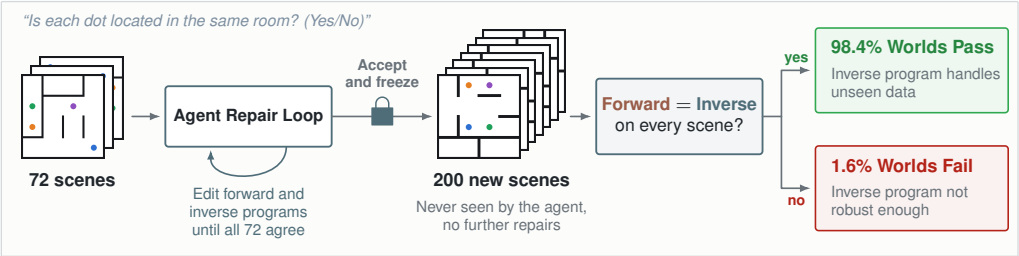


Figure 6: During the standard generation loop, the agent iterates until the forward and inverse answers agree on all 72 of the world's randomly generated scenes. To test the robustness of the inverse program, we took every world and generated 200 fresh scenes that the agent had never seen, and ran the verification checks again.

coding-agent–profile campaigns, they covered 7.09 of 9 categories on average, compared with 5.20 without requests (Sec. G.2.4).

4.3 The Inverse Program Catches Errors That Other Checks Miss

Given the importance of the inverse program to PixelProof, we measure how many errors it catches, whether the failures it reports are genuine, and how it reads images.

Experiment: Agents keep iterating until the forward and inverse programs agree on all 72 scenes of a world. To test whether passing on these 72 scenes is enough, we replay 2,234 worlds on 200 new, held-out scenes (that agents did not see during generation), and rerun the verification checks. To test whether the inverse program catches rendering errors that the other checks miss, we take 100 worlds and replace each image with that of another valid scene while keeping the recorded answer. We then check whether the inverse program correctly rejects the replacements that change the answer. Finally, to better understand how these inverse programs work, we utilize Luna to read 225 inverse programs from profile-steered generation and summarize the steps that each one uses.

Results: During generation, the inverse program frequently caught real errors. For 29% of the worlds, the agent's inverse program had flagged errors at least once, and the agent fixed its code before submitting, such as in Fig. 4. Nearly half of these fixes changed how the image was drawn or sampled, meaning in many cases the original image was rendered incorrectly before being fixed.

In the replay stress-test (Fig. 6), the forward and inverse programs disagree for 1.6% of previously verified worlds (Secs. G.1.2 and G.2.2). These worlds are removed before evaluation and training. In the replacement test on 100 worlds (300 trials), the inverse program correctly rejected 100% of replacements that changed the answer and accepted all worlds that did not (Sec. F.1). Luna's summary shows that most inverse programs combine standard image processing, such as picking pixels by color and grouping them into connected regions, with steps specific to the question, such as rebuilding a graph before tracing a path (Sec. F.3, Sec. F.4).

4.4 Most Generated Questions Are Clear and Answerable to Human Reviewers

To assess the correctness and clarity of generated questions, we conduct a human review of the data.

Experiment: Two reviewers independently assess the same 800 generated instances across different campaigns and flag any concern with the instructions, image, or generated answer (App. J).

Results: Most reviewed questions were clear and answerable to both reviewers. Neither reviewer identified a concern for 91.6% of instances while 6.8% were flagged by one reviewer and 1.6% by both reviewers (Tab. A21). Examples of flagged instances are shown in Sec. J.3.

Table 1: Accuracy (%) of each evaluator on questions produced by each coding agent, pooled over five instances per world and all nine profiles. Frontier models correctly answer most generated questions while open-weight models can correctly answer about half.

			Frontier evaluators			Open-weight evaluators		
	Coding agent	# Worlds	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Profile-steered	Sol (high)	392	96.4	93.2	95.8	55.6	60.1	53.7
	Sol (max)	250	96.7	93.8	97.0	48.3	53.2	48.7
	Luna	196	95.3	94.5	95.9	65.4	68.3	62.8
	Opus 5	211	96.0	92.8	94.4	43.6	45.3	40.4
	DS-V4-Flash	249	95.9	94.7	96.3	72.4	78.0	71.6
	Overall	1,298	96.1	93.7	95.9	56.9	61.0	55.4
Spatially-steered	Sol (high)	341	94.1	85.6	94.3	51.1	54.8	54.5
	Sol (max)	231	92.9	87.9	94.2	50.8	53.3	49.8
	Luna	129	93.0	86.7	91.2	56.0	60.0	58.1
	Opus 5	149	94.0	84.7	92.6	31.5	35.2	34.4
	DS-V4-Flash	24	95.8	96.7	98.3	77.5	80.0	80.8
	Overall	874	93.7	86.5	93.6	49.1	52.5	51.1

4.5 Generated Questions Challenge Open Models More Than Frontier Models

To determine the difficulty of the generated questions for VLMs, we run evaluations across multiple frontier closed-source models and open-weight models.

Experiment: We evaluate three frontier VLMs—Sol (high), Opus 5, and Gemini 3.7 Flash—and three open-weight VLMs—Gemma 4 12B, Gemma 4 31B, and Qwen 3.8 27B. To avoid running evaluations over the entire large set of 852,081 generated image–question pairs, we take a stratified sample of 5 rendered instances per world for a total of 10,860 image–question pairs.

Results: On average, frontier models answer 93.7% of the generated questions correctly, while open-weight models answer about half (55.0%; Tab. 1). Spatially-steered questions are harder for every evaluator. This gap may reflect both model scale and reasoning mode, since only the frontier models reason before answering. Frontier evaluators show no advantage on questions from their own model family. For example, Opus 5 answers 92.8% of Opus 5-generated profile-steered instances, compared with 93.7% of all profile-steered instances. Across both evaluation sets, declared-transform questions that require geometric transformations, such as rotating and stretching the input image (Sec. E.2.9), were the most difficult for all three frontier models (Tab. A12). The same profile also has the highest rate of reviewer concerns in human review (Sec. 4.4), so ambiguity may account for part of its difficulty.

4.6 PixelProof Can Incorporate Model Feedback to Generate Harder Questions

In Sec. 4.2, agents were steered by a request about the image itself. We also test whether generation can be steered by how well current VLMs answer the questions, using their accuracy as a proxy for difficulty. During the generation loop, frontier VLMs answer the questions from each newly verified world, and their errors are passed to the question generating agents in later episodes. We ask whether agents can use this feedback to generate harder questions.

Experiment: The evaluator frontier models we utilize are Sol (high), Opus 5, and Gemini 3.7 Flash. To bootstrap this experiment, we select two disjoint sets of ten worlds from the previous evaluation (Sec. 4.5) on which at least two of these models answered two or more of the five instances incorrectly, and provide their code in place of the discovery profiles. During each campaign, the frontier models answer five instances from every newly verified world, and their predictions and errors are passed to later episodes as an ongoing difficulty signal.

Results: Questions generated from these campaigns were harder than those from the first two experiments. Open-weight models answered an average of 30.7% correctly, close to the 28.0% expected by random chance. The frontier models answered 93.5% correctly, −1.7 pp relative to profile-steered questions (Tab. A20). In 21 of the 252 worlds with complete frontier evaluations (8.3%), at least two frontier models missed two or more of the five instances, compared with 4.5% of worlds from the first two experiments (App. H).

These questions are difficult because their answers cannot be read from any single part of the image. Many describe a procedure that must be carried out step by step, such as updating a grid over several rounds (Fig. 3a), composing two transformations (c), or predicting the water level after a wall is removed (e). Others depend on the layout of the whole image, such as finding a route that returns without repeating a path (b) or the one wedge whose reversal sends a ball to the goal (f).

4.7 Fine-tuning VLMs on PixelProof images improves accuracy on held-out benchmarks

To test the quality of data and their utility in improving VLMs, we fine-tune multiple open-weight models on PixelProof data and evaluate the fine-tuned models' performance.

Experiment: We fine-tune Gemma 4 12B, Gemma 4 31B, and Qwen 3.8 27B using supervised fine-tuning (SFT) with LoRA (Hu et al., 2022). Training uses 748 profile-steered worlds for a total of 27,762 training samples. Because instances from the same world share a generator, we hold out whole worlds rather than specific instances to avoid data leakage. We train each model four times on random data splits to ensure that results do not depend on the choice of held-out worlds.

We evaluate each trained model on its own held-out split of 45 worlds, one per model–profile campaign, with about 1,700 instances (Sec. K.4). To test a withheld question type, we also train nine adapters per model, each without one discovery profile, and evaluate each on the profile it excluded (Sec. K.6). We also evaluate on 1,260 instances from 252 model-feedback-steered worlds (Sec. 4.6) and on 16 external benchmarks covering perception and visual reasoning (App. K).

Results: Fine-tuning improves the models' held-out world accuracy by an average of +10.7 pp (Tab. A28). When a whole discovery profile was excluded from training, accuracy on that profile still improved in 26 of the 27 model–profile combinations, by +8.5 pp on average (Sec. K.6). On the harder model-feedback-steered collection, the average gain over the base models was +5.3 pp (Tab. A26). To rigorously evaluate the utility of generated data, we evaluate fine-tuned VLMs on a diverse set of 16 external benchmarks and report all results. Across three fine-tuned VLMs, we observe a large consistent accuracy gain (from +4.5 to +30.8) on five external benchmarks. Results are mixed on the other benchmarks, including real images, documents, and 3D renders (Tab. A27).

5 Discussion and Conclusion

PixelProof uses code to generate visual questions at scale without VLM judges, and several additional checks support the quality of the resulting data. After acceptance, the forward and inverse programs disagree for only 1.6% of worlds when they are replayed on 200 new images. When we replace an image with one that has a different answer but keep the original recorded answer, the inverse program correctly rejects every replacement. The generated questions remain challenging for open-weight models, and fine-tuning these models on PixelProof data improves performance not only on held-out generated questions, but also on external benchmarks. PixelProof can therefore serve as a source of synthetic data for both evaluating and training VLMs.

It is quite possible that downstream performance would improve further if PixelProof is extended to additional types of questions. Future work could also test whether longer-running campaigns further improve the quality of the generated questions and whether PixelProof can be extended to synthetic 3D challenges.

Limitations. All generated data from PixelProof are synthetic two-dimensional images. Findings may not extend to natural images. Agreement between the forward and inverse programs shows that an answer can be recovered from the image, but not necessarily that the question is clearly worded.

REPRODUCIBILITY STATEMENT

The full PIXELPROOF code, the fine-tuned models, and an interactive viewer of the generated samples can be found at https://projectpixelproof.github.io/.

ETHICS STATEMENT

PIXELPROOF generates synthetic images and questions from code written by coding agents. No human subjects, personal data, or scraped web content were involved.

AI USE STATEMENT

In this work, we used generative AI tools to generate synthetic datasets, implement the PIXELPROOF code, and support qualitative data analysis. We have not used generative AI tools to propose hypotheses, design experiments, or interpret results. Additionally, we used generative AI tools to edit the manuscript for readability. We have reviewed all AI-assisted work. The framework code was tested by the authors. We take responsibility for the final content of this work, including text, claims or artifacts produced with the aid of generative AI.

REFERENCES

David G. T. Barrett, Felix Hill, Adam Santoro, Ari S. Morcos, and Timothy P. Lillicrap. Measuring abstract reasoning in neural networks. In *International Conference on Machine Learning*, 2018.

Ted Byrt, Janet Bishop, and John B. Carlin. Bias, prevalence and kappa. *Journal of Clinical Epidemiology*, 46(5):423–429, 1993.

Dongping Chen, Ruoxi Chen, Shilin Zhang, Yaochen Wang, Yinuo Liu, Huichi Zhou, Qihui Zhang, Yao Wan, Pan Zhou, and Lichao Sun. MLLM-as-a-Judge: Assessing multimodal LLM-as-a-Judge with vision-language benchmark. In *International Conference on Machine Learning*, 2024.

Yew Ken Chia, Vernon Toh, Deepanway Ghosal, Lidong Bing, and Soujanya Poria. PuzzleVQA: Diagnosing multimodal reasoning challenges of language models with abstract visual patterns. In *Findings of the Association for Computational Linguistics: ACL 2024*, 2024.

Domenic V. Cicchetti and Alvan R. Feinstein. High agreement but low kappa: II. Resolving the paradoxes. *Journal of Clinical Epidemiology*, 43(6):551–558, 1990.

Alvan R. Feinstein and Domenic V. Cicchetti. High agreement but low kappa: I. The problems of two paradoxes. *Journal of Clinical Epidemiology*, 43(6):543–549, 1990.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A. Smith, Wei-Chiu Ma, and Ranjay Krishna. BLINK: Multimodal large language models can see but not perceive. In *Proceedings of the European Conference on Computer Vision*, 2024.

Deepanway Ghosal, Vernon Toh, Yew Ken Chia, and Soujanya Poria. AlgoPuzzleVQA: Diagnosing multimodal reasoning challenges of language models with algorithmic multimodal puzzles. In *Proceedings of the 2025 Conference of the Nations of the Americas Chapter of the Association for Computational Linguistics: Human Language Technologies (Volume 1: Long Papers)*, 2025.

Daya Guo, Dejian Yang, Haowei Zhang, et al. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638, 2025. doi: 10.1038/s41586-025-09422-z. URL https://doi.org/10.1038/s41586-025-09422-z.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Kilem Li Gwet. Computing inter-rater reliability and its variance in the presence of high agreement. *British Journal of Mathematical and Statistical Psychology*, 61(1):29–48, 2008.

Harbor Framework Team. Harbor: A framework for evaluating and optimizing agents and models in container environments. Software, version v0.16.1, Zenodo, 2026. URL https://doi.org/10.5281/zenodo.20988961.

Yicheng He, Chengsong Huang, Zongxia Li, Jiaxin Huang, and Yonghui Yang. VisPlay: Self-evolving vision-language models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Yongrui Heng, Chaoya Jiang, Han Yang, Shikun Zhang, and Wei Ye. EVE: Verifiable self-evolution of MLLMs via executable visual transformations. *arXiv preprint arXiv:2604.18320*, 2026.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022.

Yushi Hu, Reyhane Askari-Hemmat, Melissa Hall, Emily Dinan, Luke Zettlemoyer, and Marjan Ghazvininejad. Multimodal RewardBench 2: Evaluating omni reward models for interleaved text and image. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2026.

Drew A. Hudson and Christopher D. Manning. GQA: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Justin Johnson, Bharath Hariharan, Laurens van der Maaten, Li Fei-Fei, C. Lawrence Zitnick, and Ross Girshick. CLEVR: A diagnostic dataset for compositional language and elementary visual reasoning. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2017.

Baiqi Li, Zhiqiu Lin, Wenxuan Peng, Jean de Dieu Nyandwi, Daniel Jiang, Zixian Ma, Simran Khanuja, Ranjay Krishna, Graham Neubig, and Deva Ramanan. NaturalBench: Evaluating vision-language models on natural adversarial samples. In *Advances in Neural Information Processing Systems*, 2024.

Lei Li, Yuancheng Wei, Zhihui Xie, Xuqing Yang, Yifan Song, Peiyi Wang, Chenxin An, Tianyu Liu, Sujian Li, Bill Yuchen Lin, Lingpeng Kong, and Qi Liu. VL-RewardBench: A challenging benchmark for vision-language generative reward models. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025a.

Xiang Lisa Li, Farzaan Kaiyom, Evan Zheran Liu, Yifan Mai, Percy Liang, and Tatsunori Hashimoto. AutoBencher: Towards declarative benchmark construction. In *International Conference on Learning Representations*, 2025b.

Zhuowan Li, Xingrui Wang, Elias Stengel-Eskin, Adam Kortylewski, Wufei Ma, Benjamin Van Durme, and Alan Yuille. Super-CLEVR: A virtual benchmark to diagnose domain robustness in visual reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2023.

Zongxia Li, Hongyang Du, Chengsong Huang, Xiyang Wu, Lantao Yu, Yicheng He, Jing Xie, Xiaomin Wu, Zhichao Liu, Jiarui Zhang, and Fuxiao Liu. MM-Zero: Self-evolving multi-model vision language models from zero data. *arXiv preprint arXiv:2603.09206*, 2026.

Fangyu Liu, Guy Emerson, and Nigel Collier. Visual spatial reasoning. *Transactions of the Association for Computational Linguistics*, 11:635–651, 2023.

Cong Lu, Shengran Hu, and Jeff Clune. Automated capability discovery via foundation model self-exploration. *arXiv preprint arXiv:2502.07577*, 2025.

Pan Lu, Hritik Bansal, Tony Xia, Jiacheng Liu, Chunyuan Li, Hannaneh Hajishirzi, Hao Cheng, Kai-Wei Chang, Michel Galley, and Jianfeng Gao. MathVista: Evaluating mathematical reasoning of foundation models in visual contexts. In *International Conference on Learning Representations*, 2024.

Damiano Marsili, Rohun Agrawal, Yisong Yue, and Georgia Gkioxari. Visual agentic AI for spatial reasoning with a dynamic API. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Minesh Mathew, Dimosthenis Karatzas, and C. V. Jawahar. DocVQA: A dataset for VQA on document images. In *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision*, 2021.

Maxime Oquab, Timothée Darcet, Théo Moutakanni, Huy Vo, Marc Szafraniec, Vasil Khalidov, Pierre Fernandez, Daniel Haziza, Francisco Massa, Alaaeldin El-Nouby, Mahmoud Assran, Nicolas Ballas, Wojciech Galuba, Russell Howes, Po-Yao Huang, Shang-Wen Li, Ishan Misra, Michael Rabbat, Vasu Sharma, Gabriel Synnaeve, Hu Xu, Hervé Jegou, Julien Mairal, Patrick Labatut, Armand Joulin, and Piotr Bojanowski. DINOv2: Learning robust visual features without supervision. *Transactions on Machine Learning Research*, 2024.

Artemis Panagopoulou, Honglu Zhou, Silvio Savarese, Caiming Xiong, Chris Callison-Burch, Mark Yatskar, and Juan Carlos Niebles. ViUniT: Visual unit tests for more robust visual programming. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2025.

Viraj Prabhu, Senthil Purushwalkam, An Yan, Caiming Xiong, and Ran Xu. Trust but verify: Programmatic VLM evaluation in the wild. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2025.

Pooyan Rahmanzadehgervi, Logan Bolton, Mohammad Reza Taesiri, and Anh Totti Nguyen. Vision language models are blind. In *Proceedings of the Asian Conference on Computer Vision*, 2024.

Dustin Schwenk, Apoorv Khandelwal, Christopher Clark, Kenneth Marino, and Roozbeh Mottaghi. A-OKVQA: A benchmark for visual question answering using world knowledge. In *Proceedings of the European Conference on Computer Vision*, 2022.

Sanjay Subramanian, Medhini Narasimhan, Kushal Khangaonkar, Kevin Yang, Arsha Nagrani, Cordelia Schmid, Andy Zeng, Trevor Darrell, and Dan Klein. Modular visual question answering via code generation. In *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, 2023.

Dídac Surís, Sachit Menon, and Carl Vondrick. ViperGPT: Visual inference via Python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, 2023.

Ilya Sutskever. Sequence to sequence learning with neural networks: What a decade. Test of Time Award talk, Advances in Neural Information Processing Systems (NeurIPS), 2024. URL https://neurips.cc/virtual/2024/test-of-time/105032.

Richard Sutton. The bitter lesson. http://www.incompleteideas.net/IncIdeas/BitterLesson.html, 2019. Accessed: 2026-03-26.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai Charitha Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, Ziteng Wang, Rob Fergus, Yann LeCun, and Saining Xie. Cambrian-1: A fully open, vision-centric exploration of multimodal LLMs. In *Advances in Neural Information Processing Systems*, 2024.

An Vo, Khai-Nguyen Nguyen, Mohammad Reza Taesiri, Vy Tuong Dang, Anh Totti Nguyen, and Daeyoung Kim. Vision language models are biased. In *International Conference on Learning Representations*, 2026.

Rui Wang, Joel Lehman, Jeff Clune, and Kenneth O. Stanley. POET: Open-ended coevolution of environments and their optimized solutions. In *Proceedings of the Genetic and Evolutionary Computation Conference*, 2019.

Rui Wang, Joel Lehman, Aditya Rawal, Jiale Zhi, Yulun Li, Jeffrey Clune, and Kenneth Stanley. Enhanced POET: Open-ended reinforcement learning through unbounded invention of learning challenges and their solutions. In *International Conference on Machine Learning*, 2020a.

Weizhi Wang, Khalil Mrini, Linjie Yang, Sateesh Kumar, Yu Tian, Xifeng Yan, and Heng Wang. Finetuned multimodal language models are high-quality image-text data filters. *arXiv preprint arXiv:2403.02677*, 2024.

Wenhui Wang, Furu Wei, Li Dong, Hangbo Bao, Nan Yang, and Ming Zhou. MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. In *Advances in Neural Information Processing Systems*, 2020b.

Zifu Wang, Junyi Zhu, Bo Tang, Zhiyu Li, Feiyu Xiong, Jiaqian Yu, and Matthew B. Blaschko. Jigsaw-R1: A study of rule-based visual reinforcement learning with jigsaw puzzles. *Transactions on Machine Learning Research*, 2025.

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal LLMs. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2024.

Zijian Wu, Jinjie Ni, Xiangyan Liu, Zichen Liu, Hang Yan, and Michael Qizhe Shieh. SynthRL: Scaling visual reasoning with verifiable data synthesis. In *Findings of the Association for Computational Linguistics: ACL 2026*, 2026.

Yue Yang, Ajay Patel, Matt Deitke, Tanmay Gupta, Luca Weihs, Andrew Head, Mark Yatskar, Chris Callison-Burch, Ranjay Krishna, Aniruddha Kembhavi, and Christopher Clark. Scaling text-rich image understanding via code-guided synthetic multimodal data generation. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2025.

Kexin Yi, Jiajun Wu, Chuang Gan, Antonio Torralba, Pushmeet Kohli, and Joshua B. Tenenbaum. Neural-symbolic VQA: Disentangling reasoning from vision and language understanding. In *Advances in Neural Information Processing Systems*, 2018.

Chi Zhang, Feng Gao, Baoxiong Jia, Yixin Zhu, and Song-Chun Zhu. RAVEN: A dataset for relational and analogical visual rEasoNing. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019.

Wenqi Zhang, Zhenglin Cheng, Yuanyu He, Mengna Wang, Yongliang Shen, Zeqi Tan, Guiyang Hou, Mingqian He, Yanna Ma, Weiming Lu, and Yueting Zhuang. Multimodal Self-Instruct: Synthetic abstract image and visual reasoning instruction using language model. In *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, 2024.

Andrew Zhao, Yiran Wu, Yang Yue, Tong Wu, Quentin Xu, Yang Yue, Matthieu Lin, Shenzhi Wang, Qingyun Wu, Zilong Zheng, and Gao Huang. Absolute Zero: Reinforced self-play reasoning with zero data. In *Advances in Neural Information Processing Systems*, 2025.

Guanyu Zhou, Yida Yin, Wenhao Chai, Shengbang Tong, Xingyu Fu, and Zhuang Liu. VisionFoundry: Teaching VLMs visual perception with synthetic images. *arXiv preprint arXiv:2604.09531*, 2026.

Chengke Zou, Xingang Guo, Rui Yang, Junyu Zhang, Bin Hu, and Huan Zhang. DynaMath: A dynamic visual benchmark for evaluating mathematical reasoning robustness of vision language models. In *International Conference on Learning Representations*, 2025.

Xiaohan Zou, Roshan Sridhar, Mohammadtaher Safarzadeh, and Dan Roth. When vision-language models judge without seeing: Exposing informativeness bias. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, 2026.

Appendix

APPENDIX CONTENTS

A Complete Questions for the Main-Text Gallery **17**

B Additional Generated Questions **20**

B.1 Profile-steered examples across nine profiles 20

B.2 Spatially-steered examples 26

B.3 Model-feedback-steered examples 27

C Terminology **30**

D How PIXELPROOF Works **31**

D.1 What the agent receives 31

D.2 What the agent builds 31

D.3 How the agent tests and submits a world 32

D.4 How the submitted world is verified 32

D.5 How the controller decides what to keep 33

E Experimental Inputs and Shared Procedures **34**

E.1 Implementation demonstrations 34

E.2 Discovery profiles 39

E.3 Model configurations 49

E.4 Budgets and episode-hour accounting 49

E.5 Post-generation replay procedure 49

F How the Inverse Programs Solve Questions **51**

F.1 Controlled image-substitution test 51

F.2 Repairs prompted by the inverse program during generation 52

F.3 Source analysis of inverse programs 55

F.4 Worked examples 58

G Collection Growth and Generation Behavior **69**

G.1 Profile-steered generation 69

G.2 Spatially-steered generation 71

G.3 Repetition and cross-world similarity 78

H Generation Counts and Model Accuracy **81**

H.1 Generation summary across both experiments 81

H.2 Results by discovery profile 81

H.3 Results by generating configuration 83

H.4 Generation and accuracy for every profile–configuration combination 84
H.5 Evaluation coverage and eligibility for model-feedback-steered generation 87
H.6 Evaluation cost 87

I Model-Feedback-Steered Generation **89**
I.1 Feedback example sets and evaluator-error summaries 89
I.2 Generation outcomes, replay, and excluded records 89
I.3 Frontier-model and post-run open-weight evaluation 92
I.4 Worlds hard for frontier models and evaluator overlap 92
I.5 Differences across coding-agent configurations, feedback example sets, and declared mechanisms 92
I.6 Evaluator cost 93

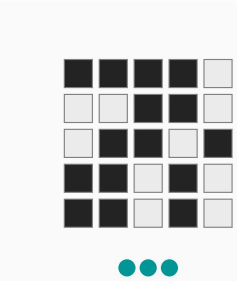
J Human Review **96**
J.1 Review procedure and sampling 96
J.2 Review outcomes and classification 96
J.3 Illustrative concerns 98

K Training on Generated Questions **99**
K.1 Training data and optimization 99
K.2 Transfer to model-feedback-steered questions 99
K.3 External benchmark transfer 100
K.4 Evaluation on held-out profile-steered worlds 100
K.5 Direct training on model-feedback-steered questions 101
K.6 Transfer to excluded profiles 101

A Complete Questions for the Main-Text Gallery

The examples below correspond to panels (a)–(j) of Fig. 3. Each entry gives the uncropped image, original question, recorded answer, and generating model.

(a) Grid updates

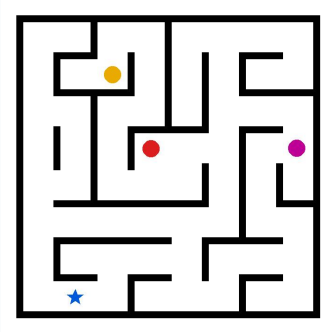


Question: "A 5 by 5 grid of square cells, each dark or pale, is drawn on a card. The small teal dot-key below tells how many rounds to run. In one round every cell is replaced: a cell becomes dark exactly when two of its eight neighbouring cells (the ones touching it along an edge or at a corner) are dark, and becomes pale otherwise. A dark cell does not automatically stay dark. Iterate the whole grid this many times, feeding each output grid into the next round. After the final round, is the number of dark cells in the grid odd? Answer with exactly one of: yes, no."

Recorded answer: yes.

Generated by: DS-V4-Flash. Model-feedback-steered generation.

(b) Returning to the star

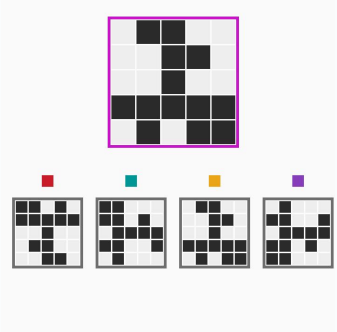


Question: "The white floor is all one room: from the blue star you can walk to every disk without crossing a black wall. For exactly one disk you can walk out to it and back to the star without ever using the same stretch of corridor twice. Reaching either of the other disks forces you to repeat a stretch. Which disk is it? Answer red, gold, or magenta."

Recorded answer: magenta.

Generated by: Opus 5. Model-feedback-steered generation.

(c) Rotate and reflect

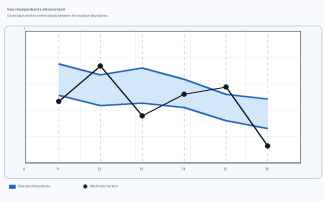


Question: "A reference 5-by-5 glyph is drawn inside the magenta-bordered frame at the top. Four framed squares below each hold another 5-by-5 glyph, and each square carries a coloured tag just above it: one crimson, one teal, one amber, one violet. Transform the reference glyph as follows: first turn it a quarter turn clockwise, then flip it over like a mirror across the horizontal midline so that the top edge swaps with the bottom edge. Exactly one of the four tagged glyphs equals the reference after this two-step transformation. Which colour tag belongs to that matching glyph? Answer with exactly one of: crimson, teal, amber, violet."

Recorded answer: teal.

Generated by: DS-V4-Flash. Model-feedback-steered generation.

(d) Trace inside a band

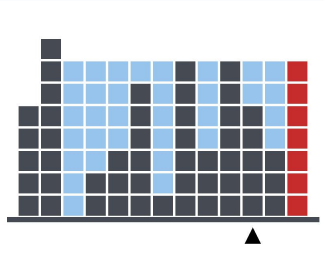


Question: "How many of the polyline vertices lie inside the blue ribbon rather than above or below it? Answer 0, 1, 2, 3, 4, 5, or 6."

Recorded answer: 1.

Generated by: Luna. Profile-steered generation.
Discovery profile: Measurement.

(e) Water after removal

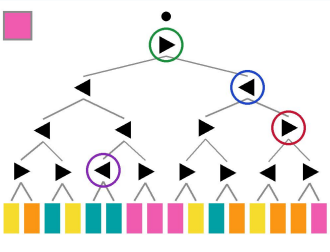


Question: "The dark blocks make a landscape of columns standing on the ground, and the pale blue blocks are the rain water trapped in that landscape as it is drawn. Water is only held where taller columns stand on both sides of it; water that can reach either open end of the landscape runs off. Exactly one column is red. Take that whole red column away, down to the ground, leave every other column exactly as it is, and let the water settle again. How many blocks deep is the water standing directly above the column the black arrow points at? Answer with exactly one of: depth-0, depth-1, depth-2, depth-3, depth-4, depth-5."

Recorded answer: depth-0.

Generated by: Opus 5. Model-feedback-steered generation.

(f) Repairing a route

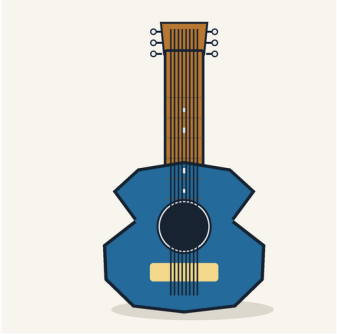


Question: "As drawn, the ball released at the top follows the wedges down and misses the goal color shown in the top-left square. You may reverse exactly one of the four ringed wedges, leaving every other wedge as drawn; the ball then runs again from the top. Only one of the four choices makes it land in a cup of the goal color. Name that ring's color. Answer blue, crimson, green, or violet."

Recorded answer: blue.

Generated by: Opus 5. Model-feedback-steered generation.

(g) Guitar strings

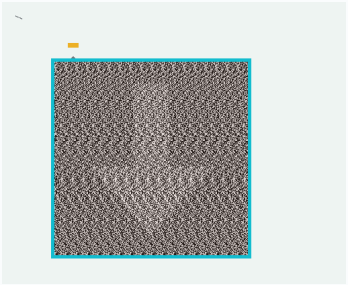


Question: "How many dark strings visibly run from the guitar's headstock, along the neck, and across its body? Count each continuous thin path once. Answer 4, 5, 6, 7, or 8."

Recorded answer: 8.

Generated by: Sol (high). Profile-steered generation.
Discovery profile: Prior conflict.

(h) Window averaging

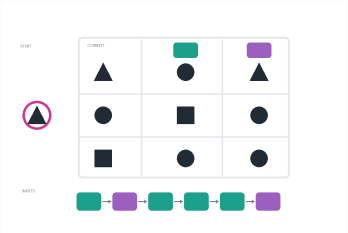


Question: "Inside the cyan frame, slide a horizontal window exactly as wide as the gold bar from left to right. Replace each full window with its average and ignore the narrow incomplete edge windows. Which large symbol appears: up arrow, right arrow, down arrow, or left arrow? Keep the image orientation unchanged."

Recorded answer: down arrow.

Generated by: Sol (max). Profile-steered generation.
Discovery profile: Declared transform.

(i) State transitions

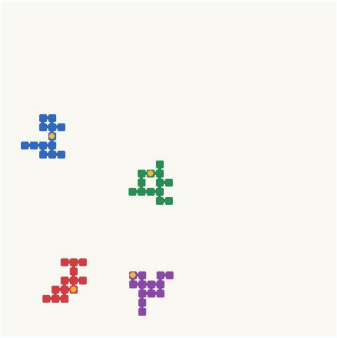


Question: "Start at the shape inside the magenta ring. For each input card from left to right, use the table row for the current shape and the column whose header color matches the card; the shape in that cell becomes the new state. Which shape is the final state?"

Recorded answer: circle.

Generated by: Sol (high). Profile-steered generation.
Discovery profile: State tracking.

(j) Removing a connecting cell



Question: "Four colored cell islands are shown, each with one gold pin. Which color's island would split into separate pieces if the single cell holding its gold pin were removed? Exactly one qualifies. Answer red, blue, green, or purple."

Recorded answer: blue.

Generated by: Sol (max). Spatially-steered generation.
Discovery profile: Search.

B Additional Generated Questions

These additional examples show questions from each generation experiment.

B.1 Profile-steered examples across nine profiles

Fig. A1 shows one world per discovery profile. The complete questions and recorded answers follow the overview.

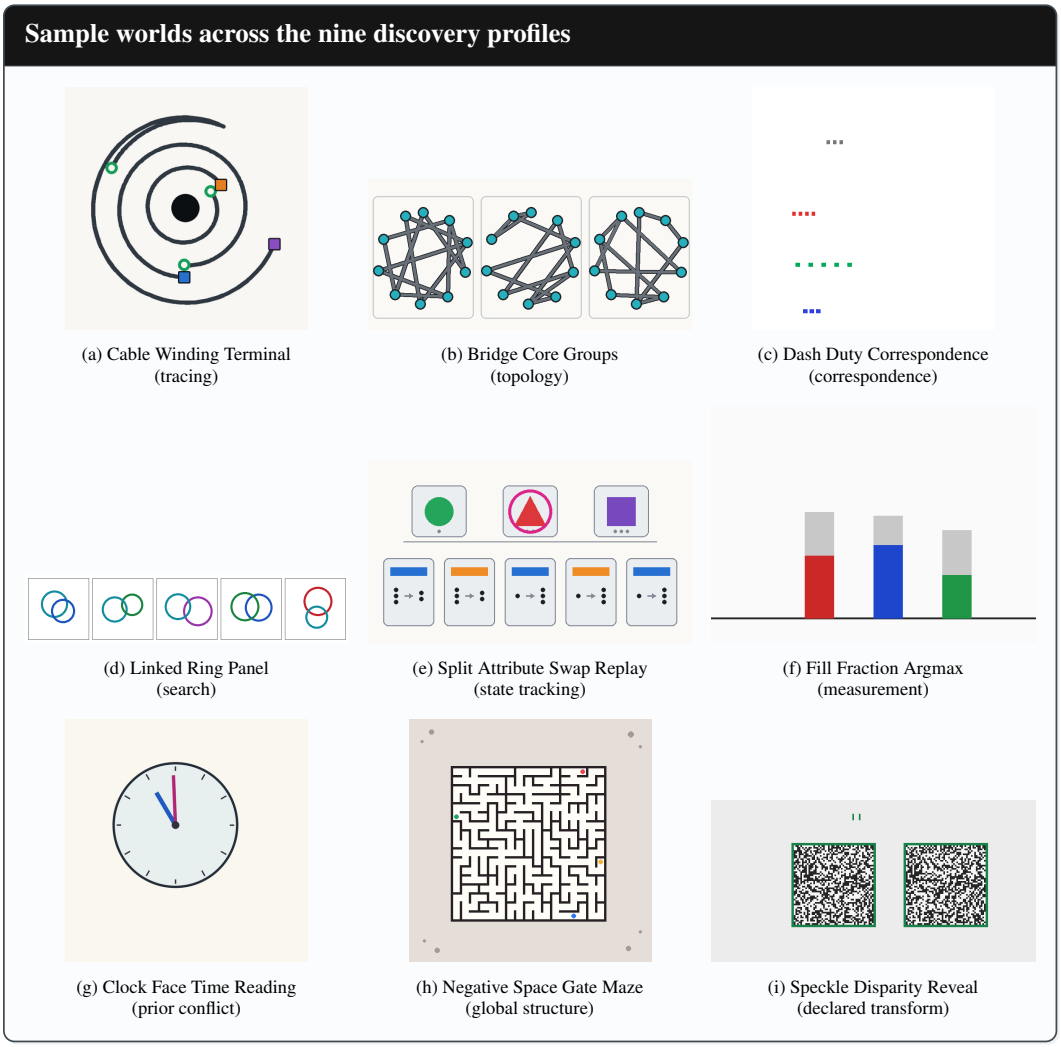


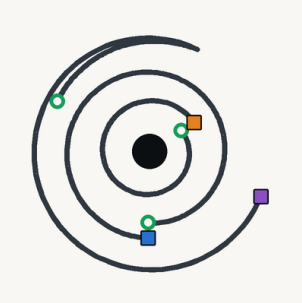
Figure A1: One profile-steered example per discovery profile. Complete questions and recorded answers follow below.

Complete questions

The following entries match the nine panels of Fig. A1 in reading order. Each image is paired with its verbatim question and recorded answer. The ten main-text examples are documented separately in App. A.

(a) Cable Winding Terminal

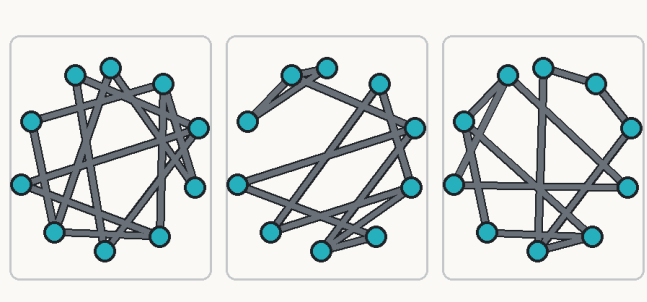
Generated by: Sol (high). **Experiment:** Profile-steered generation.
Discovery profile: Tracing.



Question: "Starting at each ring, follow its dark cable to the colored square. Which cable makes exactly one complete clockwise turn around the black hub? Answer orange, purple, or blue."
Recorded answer: blue.

(b) Bridge Core Groups

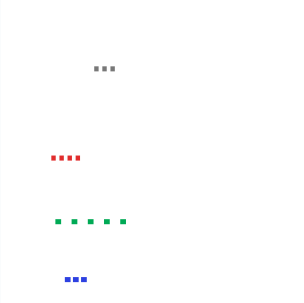
Generated by: Sol (high). **Experiment:** Profile-steered generation.
Discovery profile: Topology.



Question: "Gray links meet only at cyan round stations; a crossing without a station is not a connection. A bridge is a gray link whose removal disconnects its panel. Imagine removing every bridge link at once. A loop group is a remaining connected group of stations that contains a loop. Which panel has exactly two loop groups? Answer left, center, or right."
Recorded answer: left.

(c) Dash Duty Correspondence

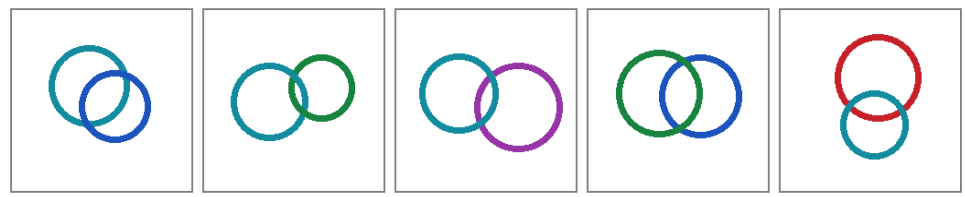
Generated by: DS-V4-Flash. **Experiment:** Profile-steered generation.
Discovery profile: Correspondence.



Question: "The gray dashed bar at the top is a pattern of solid dashes separated by empty gaps. Below it are three colored dashed bars: one is red, one is green, and one is blue, and each has a different dash-to-gap ratio (how long each solid dash is compared to the empty gap to its right). Which colored bar has exactly the same dash-to-gap ratio as the gray bar? Answer red, green, or blue."
Recorded answer: red.

(d) Linked Ring Panel

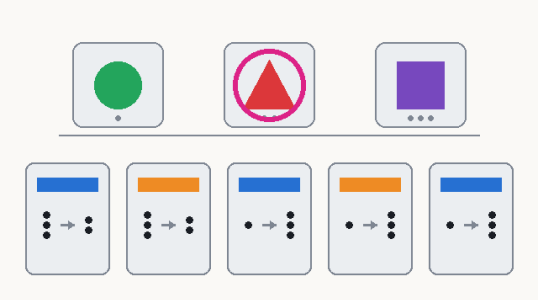
Generated by: Opus 5. **Experiment:** Profile-steered generation.
Discovery profile: Search.



Question: "This image shows five framed square panels of identical size standing side by side in one row, numbered 1 to 5 from left to right. Every panel holds exactly two rings of the same thickness, drawn in two different flat colours, and in every panel the two rings cross each other at exactly two places, so ring count, ring size, colour, position and the amount of ink carry no information. The rings are opaque, so at each crossing one ring passes in front and hides a short piece of the other one, leaving a plain break in the ring behind. Two rings are interlocked when neither ring stays in front the whole way round: one ring passes in front at one crossing and behind at the other, so each of the two rings carries exactly one break and each is still a single connected curve. Two rings are merely stacked when the same ring passes in front at both crossings, so that front ring carries no break at all while the ring behind it is broken twice and falls into two separate pieces. Exactly one panel holds a pair of interlocked rings. In each of the other four panels the pair is merely stacked, and the piece of the ring behind that lies between its two breaks is at least 26 pixels long, so both of its breaks are plainly visible. Which ring is drawn in which colour, which ring is in front at a given crossing, and where the pair sits inside its panel are all free to vary. Which numbered panel holds the two interlocked rings, that is the panel in which each of the two rings carries exactly one break? Answer with exactly one of: panel-1, panel-2, panel-3, panel-4, panel-5."
Recorded answer: panel-2.

(e) Split Attribute Swap Replay

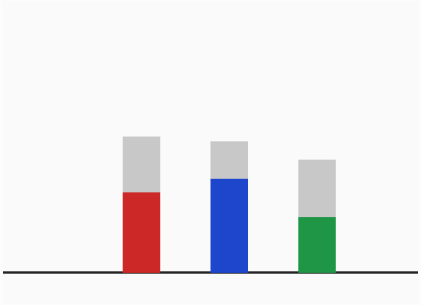
Generated by: Sol (high). **Experiment:** Profile-steered generation.
Discovery profile: State tracking.



Question: "The three fixed slots start with the shown colored shapes. Read the five cards from left to right. The dot counts name two slots. A BLUE card swaps only the colors in those slots; an ORANGE card swaps only the shapes. The other attribute stays. What color and shape finish in the magenta-ringed slot?"
Recorded answer: purple square.

(f) Fill Fraction Argmax

Generated by: DS-V4-Flash. **Experiment:** Profile-steered generation.
Discovery profile: Measurement.



Question: "Three colored vertical bars (red, blue, green) stand on a black baseline. Each bar is a light-gray full-height skeleton with a solid colored fill rising from the baseline. Which colored bar is filled to the greatest fraction of its own total height, meaning the largest ratio of colored fill height to that same bar's full gray height? Answer with only the color: red, blue, or green."
Recorded answer: blue.

(g) Clock Face Time Reading

Generated by: Luna. **Experiment:** Profile-steered generation.
Discovery profile: Prior conflict.



Question: "What time is shown on the analog clock face? Read the short hand as the hour and the long hand as minutes at the nearest quarter-hour; answer in H:MM format."
Recorded answer: 12:00.

(h) Negative Space Gate Maze

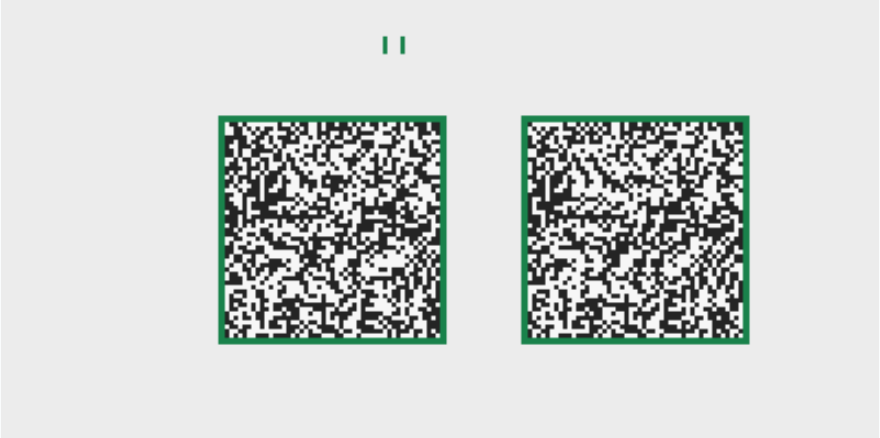
Generated by: Sol (max). **Experiment:** Profile-steered generation.
Discovery profile: Global structure.



Question: "Which two colored gates are connected by one continuous white path through the charcoal maze? Paths may bend; corner-only touching does not connect. Answer as color-color, for example red-blue."
Recorded answer: blue-yellow.

(i) Speckle Disparity Reveal

Generated by: Opus 5. **Experiment:** Profile-steered generation.
Discovery profile: Declared transform.



Question: "The picture shows two square plates side by side, each enclosed by a coloured frame and each printed with a fine black-and-white speckle. Both plates cover the same 7 rows and 7 columns of equal square cells, and the two plates are the same size, so every cell of the right plate has one matching cell of the left plate. The right plate repeats the left plate's speckle cell by cell, in one of two ways. In a plain cell the speckle is copied straight across, so that cell of the right plate is exactly the matching cell of the left plate. In a slid cell the speckle is copied from further to the right: what the right plate shows at a point of a slid cell is what the left plate shows the slide distance further right of the matching point, in the same row. The slide distance is the same in every slid cell and is printed above the plates as the distance between the two small coloured tick marks. Each row of the left plate is a loop, so a strip that runs past its right edge continues from its left edge. Neither plate on its own shows anything but speckle; only comparing them tells the slid cells from the plain ones. On the 7 by 7 grid the slid cells form a shape that most closely matches one of these five symbols: crescent, funnel, pinwheel, spiral, staircase. Which symbol is it? Answer with one word."
Recorded answer: pinwheel.

B.2 SPATIALLY-STEERED EXAMPLES

Examples of spatial patterns. Fig. A2 provides selected replay-verified examples from each of the nine spatial patterns (Sec. 4.2). The labels report the category measured after submission, not a category chosen by the coding agent. Scale describes the spatial span of regions whose masking changes the inverse decision or causes an error. Shape describes whether those affected regions are compact, pathlike, or split into several components.

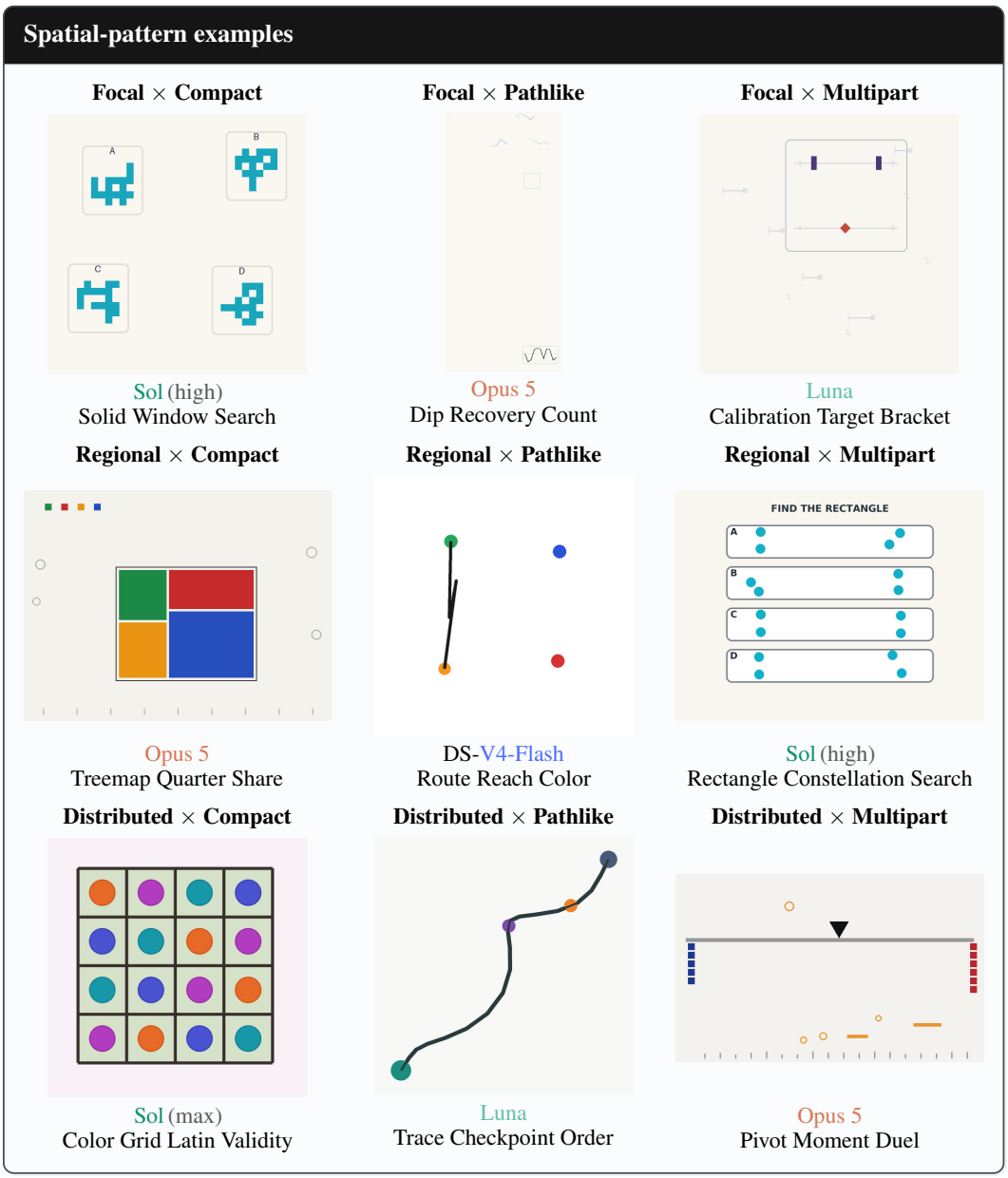


Figure A2: Replay-verified examples spanning the nine spatial patterns. Images are pattern illustrations. Labels identify the coding-agent configuration and world.

B.3 MODEL-FEEDBACK-STEERED EXAMPLES

We show four worlds from model-feedback-steered generation (Sec. 4.6), selected by the authors from the 21 worlds hard for frontier models. A world is hard for frontier models if at least two of the three frontier models answer two or more of its five evaluation instances incorrectly. Each entry gives a one-line summary, the verbatim question and recorded answer for the image shown, and the frontier-model scores over the world's five evaluation instances, which do not include the image shown. All four worlds passed the 200-scene replay. Five further model-feedback-steered worlds appear in the main-text gallery (Fig. 3a–c,e,f), with full questions in App. A.

Twisted Rim Cover

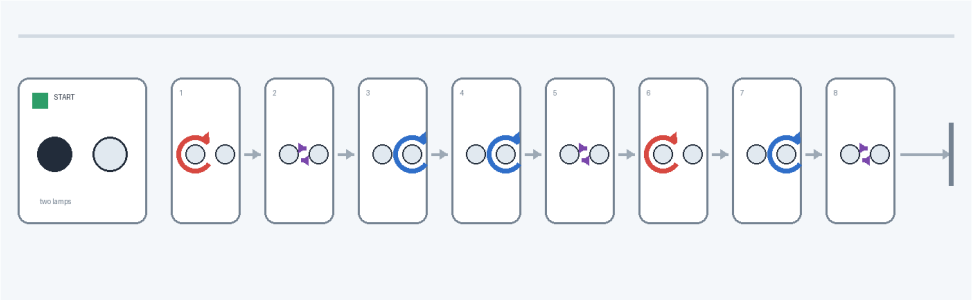
In short: Overlay three maze plans, then follow the path through colored holes that switch between two layers; which exit can be reached?
Generated by: Sol (high).



Question: "Overlay the plans: walls shown once stay; walls shown twice vanish. Paired holes join matching rim colors. The visible key says whether each letter KEEPS or SWAPS layer I/II; no turning inside a hole. START is on I. Which labeled exit-side is reachable?"
Recorded answer: C.
Frontier models, instances correct: Sol (high) 0/5, Opus 5 1/5, Gemini 3.7 Flash 0/5.

Ordered Two-Lamp Gate Process

In short: Apply eight flip and swap gates to two lamps, in order; which lamps end up lit?
Generated by: Luna.



Question: "Begin with the two-lamp state shown in START. Read the eight colored gate glyphs from left to right: a red circular arrow around the left lamp flips left; a blue circular arrow around the right lamp flips right; a violet double arrow exchanges the lamps. Apply each gate to the current state. What is the final state: both-dark, left-lit, right-lit, or both-lit?"
Recorded answer: both-lit.
Frontier models, instances correct: Sol (high) 2/5, Opus 5 3/5, Gemini 3.7 Flash 2/5.

Keyed Column Weave

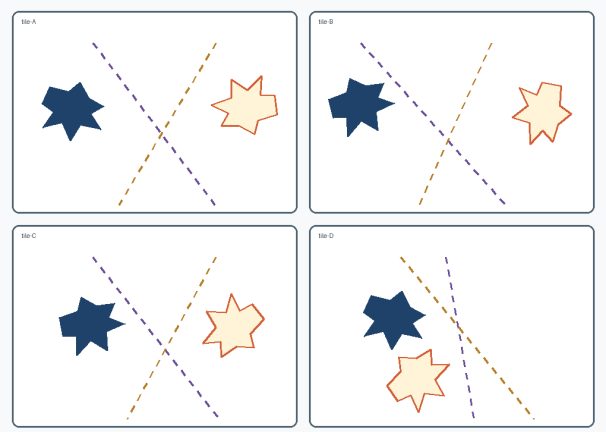
In short: Weave each source card's two strips into a grid; which card's grid matches its check grid?
Generated by: Luna.



Question: "Match each SOURCE key in the three shuffled rows. For one key, place its ODD strip in columns 1, 3, and 5, and its EVEN strip in columns 2 and 4. Select the SOURCE card whose woven grid exactly matches its keyed CHECK grid. Answer card-1, card-2, card-3, card-4, or card-5."
Recorded answer: card-4.
Frontier models, instances correct: Sol (high) 5/5, Opus 5 0/5, Gemini 3.7 Flash 3/5.

Serial Two-Axis Reflection

In short: Reflect each token across two dashed lines in turn; in which tile does it land on the target?
Generated by: Luna.



Question: "Four tiles are labeled A-D. Each tile has a solid token, a violet dashed line, a gold dashed line, and a hollow target. Reflect the token across the violet line, then across the gold line. Which tile's target is the result?"
Recorded answer: tile-D.
Frontier models, instances correct: Sol (high) 3/5, Opus 5 4/5, Gemini 3.7 Flash 1/5.

C TERMINOLOGY

Term	Definition
Question world (world)	A visual question with a sampler, a renderer, a forward program, and an inverse program.
Sampler	Uses a random seed to produce a scene specification: the objects in one instance and their properties.
Instance	One scene specification, its rendered image, and the world's question.
Forward program	Computes the answer from the scene specification.
Inverse program	Computes the answer from the rendered image alone.
Verifier	Runs the final checks on a submitted world, including forward–inverse agreement on 72 scenes.
Episode	One fresh agent session followed by verification.
Campaign	A sequence of episodes with the same coding-agent configuration and generation instructions.
Collection	The set of accepted worlds.
Discovery profile	Generation instructions that define a family of questions by the computation they require, with five text-only examples, such as following a path (Tracing) or recovering connectivity (Topology).

D How PixelProof Works

Three components take part in each episode. The coding agent writes and tests a question world. The verifier checks the submitted world after the agent session ends. The controller prepares each episode and decides which worlds to keep. This section follows one episode from its inputs to the final decision, expanding Sec. 3.

D.1 What the agent receives

Each episode starts a fresh agent session with no memory of earlier sessions. The agent receives:

- the generation instructions and the three implementation demonstrations (Sec. E.1);
- the self-checks it can run;
- the time allowed for the session, written as `timeout_seconds` in `TASK_SPEC.toml`. This value is fixed when the episode starts; it is not a live countdown;
- the history of its campaign: the full code and structured summaries of accepted worlds, and the most recent rejected submission with its verifier report.

The controller carries this history through files, not through a continuing conversation.

Each experiment adds its own inputs (Tab. A1). In profile-steered generation, the agent receives one fixed profile with five text-only example questions. Spatially-steered generation uses the same profile and examples and adds a request for which image regions the inverse program should rely on. In model-feedback-steered generation, the agent instead receives ten earlier question implementations and a summary of how evaluator models answered them, which is updated during the campaign. These ten implementations are not counted as newly generated worlds.

Table A1: Experiment-specific episode inputs. Every episode also receives the implementation demonstrations, generation instructions, self-checks, campaign history, and the time allowed for the current session. Campaign history contains accepted code and summaries, plus the most recent rejected submission and its verifier report.

Experiment	Fixed inputs	Additional guidance updated between episodes
Profile-steered	One profile with five text-only example questions	None beyond campaign history and the session time allowance
Spatially-steered	One profile with five text-only example questions	Spatial-pattern requests and guidance on building worlds for them
Model-feedback-steered	Ten feedback examples with initial evaluator feedback and a card describing them	Evaluator feedback and reasoning assignments

Campaigns do not share accepted worlds or rejection reports, even when they run at the same time. Each profile-steered or spatially-steered campaign uses one coding-agent configuration and one profile and starts with no generated worlds. Each model-feedback-steered campaign uses one configuration and a fixed set of ten feedback examples selected from earlier experiments.

The controller saves the episode inputs in the agent's workspace, and the first prompt tells the agent which files to read. The task specification includes the seeds configured for stress tests. It also contains a copy of the verifier's gate script, so the agent can run the same checks as the final verification. The verifier's offline and process-log checks and the protected comparison records are supplied only after submission. The saved inputs show what was available to the agent, not which files it actually read.

D.2 What the agent builds

The agent writes one executable question world with these parts:

- a sampler that produces a scene specification, and a renderer that draws the image
- a forward program that computes a decision from the scene specification, and an inverse program that recovers a decision from the image
- fixed question texts and a shared answer mapping that turns each decision into an answer

- command-line code for generating instances, tests, a self-check script, and saved examples with run records

The verifier compares the two final answers, not intermediate values. For example, the forward program may compute a length from exact coordinates while the inverse program estimates it from pixels. Only the resulting answers must agree.

The inverse program receives only the image. It may not receive the scene specification or import the sampling, rendering, answer-mapping, generation, self-check, or forward-program code. The verifier checks the program's declared inputs and imports, but these source checks cannot detect every dependency loaded at run time.

Each saved example records the random seed, scene specification, image path, question text, answer options, and the recorded decision and answer, together with the scene's distance from the decision threshold and whether the ambiguity rule excludes it. These records let the verifier reproduce the examples; the inverse program never sees them.

The agent must also list any changes to the scene specification that should leave the image unchanged, or explain why there are none. The verifier uses this list to check that the same image never receives different answers.

Finally, the agent describes the world: what it asks, how its scenes are built, and how the answer is computed. It fills in a structured signature with the decision variable, scene structure, decision type, and mechanism, names the most similar recorded world, and explains how the new world differs. The controller uses this description in its retention checks (Sec. D.5).

D.3 How the agent tests and submits a world

While time remains, the agent can run self-checks and revise its code. The self-checks test the required interface, execution, reproducibility, and forward–inverse agreement on saved and newly sampled instances, and their reports tell the agent what to fix. Passing them allows submission but does not replace final verification.

The agent may test many revisions but can submit only once per episode. After the self-checks pass, it runs a finalization step that records a hash of the final code. If it changes the code afterward, it must rerun the self-checks and finalization before the session ends.

D.4 How the submitted world is verified

Final verification starts after the agent session ends. Harbor (Harbor Framework Team, 2026) keeps the declared submission files and moves them to a clean container built from the fixed task image, with no network access, provider credentials, conversation history, or agent scratch files. It then supplies the verifier's entry script, its offline and process-log checks, and the protected comparison records. The verifier checks the saved examples, generates new instances, and runs four groups of checks:

1. **Execution and reproducibility.** Files must meet size and format limits, and the code must run offline. The same inputs must give the same outputs, and stored images and answers must match their regenerated versions.
2. **Forward–inverse agreement.** On 72 instances generated from the three seeds listed in the task specification (101, 2027, and 9929; 24 scenes each), the verifier recomputes the forward answer, runs the inverse program on the image, and compares the two answers. It also checks the inverse program's inputs and imports.
3. **Scene consistency and ambiguity.** Declared changes that leave the image unchanged must leave the answer unchanged. The verifier checks that decision thresholds have visible evidence and recomputes the ambiguity rule. For each question text in each tested batch, the scenes that the rule does not exclude must contain at least two answer classes. The protocol does not cap how many scenes the rule may exclude.
4. **Rejection of corrupted examples.** The verifier changes stored answers and replaces images with blank ones, then runs the submitted checking script, which must reject them. This tests the script, not whether the inverse program abstains on every corrupted input.

The gate script that runs these four groups is the same file the agent receives in its workspace, and the seeds are listed in the task specification, so an agent could run the same checks during its session. The agent-facing protocol text described these seeds as hidden, but the task specification listed them in every paper campaign; we report this as a deviation from the protocol text. What the agent could not see or change are the protected comparison records used by the reuse checks (Sec. D.5) and the final offline run of its frozen submission. Instances that no agent saw are tested only by the later 200-scene replay (Sec. E.5). The verdict is final for the episode, meaning the agent cannot make further repairs. A later episode may receive the rejected submission and its report as history. Passing verification shows that the two programs agree under these tests, but it does not show that humans can answer the question or that the world is new.

D.5 How the controller decides what to keep

Passing verification is not enough for a world to be kept. The controller also requires that the agent finalized the submitted code and that the world passes three retention checks against a comparison set. This set starts with previously implemented worlds and submissions from earlier exploratory runs, including failed ones, and grows with the campaign's own accepted worlds. It does not include worlds from other campaigns running at the same time.

The three retention checks are:

1. The combination of decision variable, scene structure, and decision type must not exactly match a recorded entry.
2. The agent must name the most similar recorded world and explain the difference.
3. A separate comparison checks the world's descriptions and mechanism against protected records of earlier worlds. It flags specified combinations of operations and similarity scores above the threshold stored for each earlier world; it does not compare source code. The mechanism field is used in this comparison but not in the exact-match test of the first check.

These checks limit reuse of recorded descriptions and mechanisms, but they do not establish that a world is new.

The finalization record must match the hash of the submitted code. If it is missing or does not match, the world is recorded as budget-incomplete, even if it passes verification and retention. Finalization must happen before the session deadline, but the controller can decide afterward. Incomplete submissions, timeouts, and infrastructure failures are recorded separately from accepted and rejected worlds. Automatic acceptance is not human approval and does not add a world to the released dataset.

E EXPERIMENTAL INPUTS AND SHARED PROCEDURES

All three generation experiments supply the same three implementation demonstrations. Profile-steered and spatially-steered generation additionally supply text-only discovery profiles. We describe these inputs first, followed by configurations, budgets, and post-generation replay. The feedback example sets, feedback, and differences in feedback processing in model-feedback-steered generation are described in Sec. I.1.

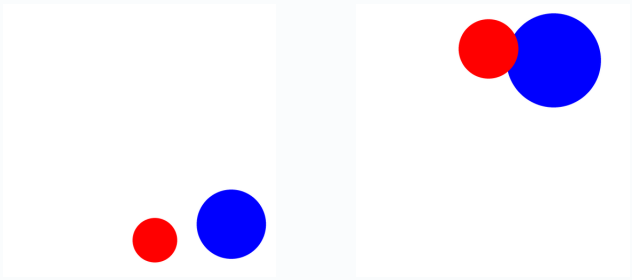
E.1 IMPLEMENTATION DEMONSTRATIONS

The three implementation demonstrations provide few-shot examples of how to implement visual questions. Every episode in all three generation experiments receives the same source files for circle contact, angle acuteness, and counting with distractors. Each example contains `DESIGN.md`, `world.toml`, `renderer.py`, `prompts.py`, and `oracle.py`. These are importable reference modules, not complete standalone bundles. Generation and verification entry points and tests are not included. Separate task instructions and self-checks explain how to develop, repair, and submit a new world. Fig. A3 illustrates the supplied implementations.

The pixel-only entry point, `decision_from_image(image)`, recovers a quantity that the question mapping converts to an answer. The separate `verify_example(image, scene)` wrapper compares image measurements with the scene specification; it does not use the image alone. The inverse functions use fixed public palette and style assumptions. The expected quantities are computed within the supplied renderer modules and mapped to answers in `prompts.py`, rather than in a separate forward-program file.

Implementation demonstrations: scene generation and image-only answers

Circle contact. "Are the two circles overlapping? Answer yes or no."



Left: no. Right: yes.

The forward computation compares the center distance with the sum of the radii. The inverse program segments the colored disks, extracts their exposed rims, and fits circles by least squares to recover the same comparison. Neither displayed instance is near contact.

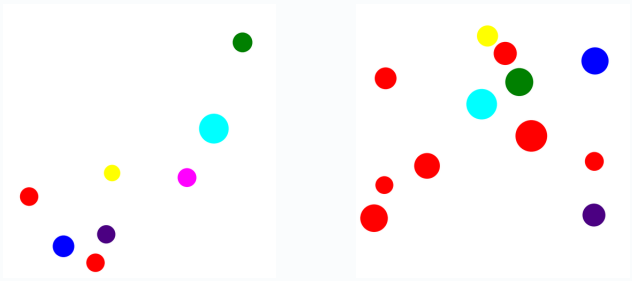
Angle acuteness. "Is the angle acute? Answer yes or no."



Left: yes. Right: no.

The forward computation compares the specified angle with 90°. The inverse program segments the strokes, fits two lines using RANSAC, and refines their directions using principal component analysis. It recovers the angle between the rays; neither example is near a right angle.

Counting with distractors. "How many red dots are there? Answer with a number."



Left: 2. Right: 7.

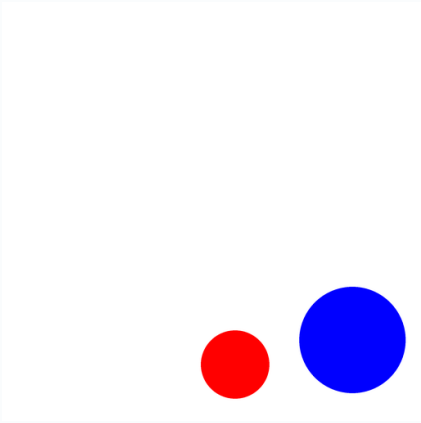
The forward computation counts target disks in the scene specification. The inverse program classifies pixels using the fixed palette and counts four-connected red components. A separate comparison wrapper checks their fitted shapes. The shown disks are separated to avoid merged components.

Figure A3: Instances rendered for this paper from the unchanged implementation demonstrations, not images supplied to agents. Both answer computations agree for all six instances. Each pair uses the same question with different scene specifications. Implementation details follow below.

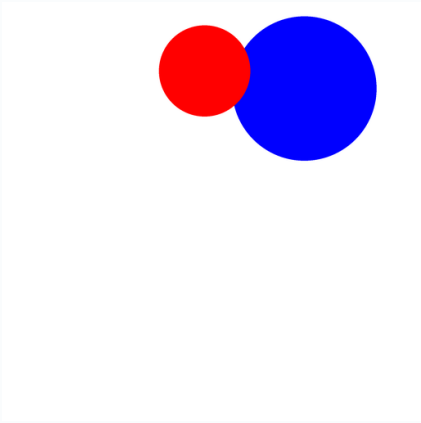
E.1.1 CIRCLE CONTACT: RECOVERING GEOMETRY UNDER PARTIAL OCCLUSION

Implementation demonstration: circle contact

Question. "Are the two circles overlapping? Answer yes or no."



Answer: no. Separate circles.



Answer: yes. Overlapping circles.

Scene generation. The sampler chooses two radii, their relative direction, and a position on the canvas. A specified signed gap controls how far apart the circles are. Placements are resampled until both circles fit within the image and neither is almost entirely hidden by the other. The renderer draws a blue disk, then a red disk over it, on a white background. Antialiasing smooths their edges. Thus an overlapping instance contains only part of the blue rim.

Forward program. The scene specification gives centers c_1, c_2 and radii r_1, r_2. The forward computation returns the signed gap

$$m = \|c_1 - c_2\|_2 - r_1 - r_2.$$

The supplied answer mapping returns yes when $m \leq 0$ and no otherwise. This convention includes exact contact in yes; the displayed instances are well away from contact. Their signed gaps are 40 and -20 pixels.

Inverse program. The inverse program first assigns each pixel to the nearest blue, red, or white palette color. For each disk, it selects colored pixels adjacent to the white background. This excludes the red–blue interface, which is an occlusion edge rather than the hidden blue circle's rim. It fits a circle to each exposed rim by algebraic least squares, removes large-residual points, and refits. The recovered centers and radii give an image-based signed gap, to which the same answer mapping is applied. The measured gaps are 40.61 and -19.31 pixels, giving the answers above without access to the scene specification.

Handling occlusion. The inverse program fits only exposed circle rims; the boundary between the red and blue regions includes an occlusion edge. A separate comparison wrapper checks the fitted geometry and visible areas against the scene specification. The inverse decision function does not receive that specification.

These images were rendered for the paper from the supplied source files. Agents receive the implementation demonstration, not these illustrations.

E.1.2 ANGLE ACUTENESS: RECOVERING THE ANGLE BETWEEN TWO RAYS

Implementation demonstration: angle acuteness

Question. "Is the angle acute? Answer yes or no."



Answer: yes. Angle: $50°$.

Answer: no. Angle: $125°$.

Scene generation. The scene specification contains a vertex, the direction of the first ray, the angle between the rays, their lengths, and their stroke width. The sampler varies position, orientation, lengths, and width while ensuring that both strokes fit inside the canvas. The renderer draws two blue rectangular strokes sharing a vertex on a white background. Changing these variables produces different images without changing the question.

Forward program. The specified angle θ determines the decision directly. The forward computation returns $m = 90° - \theta$, and the answer mapping returns yes exactly when $m > 0$. The two instances have margins of $40°$ and $-35°$, respectively. An exact right angle is not acute; neither displayed instance is close to that decision threshold.

Inverse program. The inverse program separates blue ink from the white background using the fixed palette. Random sample consensus (RANSAC) fits one dominant line and then a second line to the remaining ink. Principal component analysis refines the line directions. It excludes pixels near the corner during refinement, where the two thick strokes merge, then estimates the vertex from the intersection of the fitted lines. It orients both directions away from that vertex and computes their angle from the dot product. This orientation step distinguishes the rays' angle from its supplement. The recovered angles are $49.95°$ and $125.06°$. Applying the same acute-angle predicate gives yes and no.

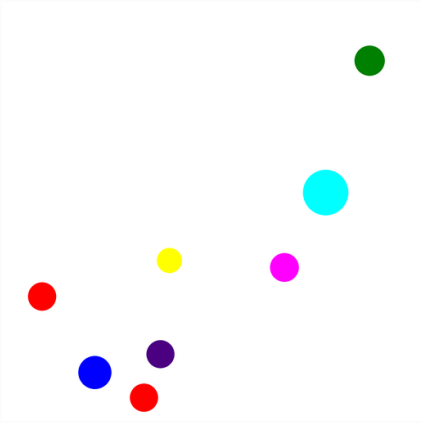
Recovering ray directions. Fitting two lines is not enough: the inverse program must determine which way each ray points to distinguish the angle from its supplement. A separate comparison wrapper checks the recovered vertex, endpoints, angle, and colored area against the scene specification.

These images were rendered for the paper from the supplied source files. The example demonstrates world implementation; episode instructions and self-checks separately explain how to develop and submit a new world.

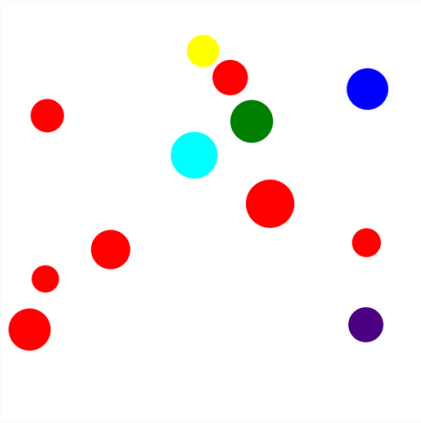
E.1.3 COUNTING WITH DISTRACTORS: SELECTING AND COUNTING TARGET OBJECTS

Implementation demonstration: counting with distractors

Question. "How many red dots are there? Answer with a number."



Answer: 2. Six distractors.



Answer: 7. Five distractors.

Scene generation. The sampler chooses between two and seven red target disks and between one and six distractor disks in other palette colors. Their centers and radii vary. Placement rejects disks that overlap or extend beyond the canvas; a spacing constraint keeps adjacent disks visually separate. The renderer draws the filled disks on white with antialiased edges. The scene specification stores separate lists of targets and distractors.

Forward program. The forward computation is the length of the target list. The answer mapping converts this integer into the requested numerical answer: 2 on the left and 7 on the right. Other supplied question texts apply predicates to the same count, such as whether it exceeds three. Spacing affects whether disks can be distinguished in the image, but it is not the answer to the counting question.

Inverse program. The inverse program assigns pixels to the nearest color in a fixed palette, then labels connected components using union–find. Pixels are connected through horizontal or vertical neighbors. Components smaller than 300 pixels are recorded as speckle rather than counted as disks. For each remaining component, it fits a circle to its rim by algebraic least squares. It returns the number of red components, independently of the scene's target list. The two images yield counts of 2 and 7. Distractor components are measured separately and do not contribute to these answers.

Keeping target disks separate. Connected-component counting recovers the intended count only while target disks remain separate. The inverse function returns that count. A separate comparison wrapper checks the counts, circle-fit residuals, areas, positions, and spacing against the scene specification.

These images were rendered for the paper from the supplied source files. This implementation demonstration is distinct from a discovery profile, whose text-only questions guide what the agent should generate.

E.2 DISCOVERY PROFILES

Each discovery profile provides instructions about the kinds of questions to generate and five text-only example questions (Tab. A2). The assigned profile remains fixed throughout a campaign, and all five examples are supplied together. These inputs are separate from the three implementation demonstrations.

Table A2: Discovery profiles shared by profile-steered and spatially-steered generation. Each profile defines a family of questions through an intended computation and five text-only examples. The table illustrates each profile with a question. The full examples follow in Sec. E.2.1 onward.

Profile	Target computation	Illustrative question
Tracing	Follow a visible path or network.	Which checkpoint occurs first on the marked path?
Topology	Recover holes, enclosure, or connectivity.	Does removing the marked link disconnect the endpoints?
Correspondence	Match parts or attributes across panels.	Which part preserves the reference relation across panels?
Search	Find the unique item meeting the constraints.	Which object alone satisfies all shown constraints?
State tracking	Track identity, position, or state over steps.	Where does the marked object end after the updates?
Measurement	Read and compare quantities.	Which item exceeds both displayed thresholds?
Prior conflict	Report visible features despite expectations.	How many sides does the pictured sign visibly have?
Global structure	Combine distributed evidence or negative space.	Which symbol is formed by the negative space?
Declared transform	Apply an explicitly stated rearrangement.	Which symbol results after the stated unwrapping?

Profile construction and selection. We generated 88 candidate seed questions with author review. Each candidate was implemented with a renderer and a pixel-only inverse program, and we checked that the two agreed. Of the 88, 27 were adapted from VLMsAreBlind (Rahmanzadehgervi et al., 2024) and VLMsAreBiased (Vo et al., 2026). The rest cover basic visual operations (path tracing, topology, correspondence, search, and state tracking), scientific figures and charts, global structure, and declared transformations. We grouped the candidates by the computation they require into the nine profiles in Tab. A2. Author review removed 16 candidates, leaving 72. For each profile, we then selected five with a fixed seeded hash ranking. Profiles with exactly five candidates kept all of them. The resulting 45 questions and profile instructions were fixed before the campaigns began. Profiles may overlap and do not cover all possible visual tasks.

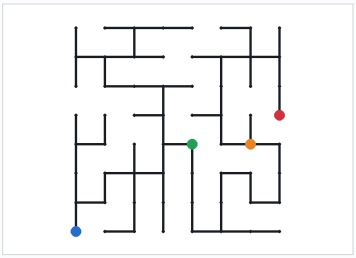
Ten of the 45 example questions are adapted from these two benchmarks, written as new question texts without copying benchmark images: five from VLMsAreBlind (Tracing 1–2, Topology 1–2, and Global structure 1) and all five prior-conflict questions from VLMsAreBiased. The other 35 come from the remaining candidates. Because both benchmarks also appear in the external evaluation (Sec. K.3), transfer to them is closer to the training distribution than transfer to the other benchmarks.

The following galleries (Fig. A4 onward) show all 45 example questions, one per row. Agents received the profile instructions and the question text marked *Given to agent*, not the images, answers, or their implementations. Each image is one stored instance that illustrates the example for readers. A dagger (†) marks examples whose stored instance uses a reworded, more detailed version of the question. Agents could also receive code and images from worlds generated during the campaign, as described in Sec. D.1.

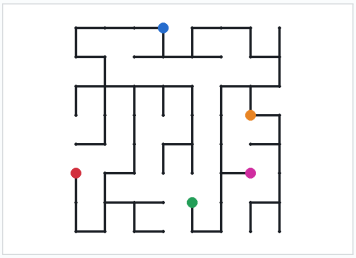
E.2.1 TRACING

Follow a visible path or network to determine its order, destination, or connections.

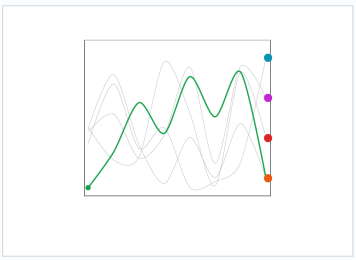
Tracing: five example questions



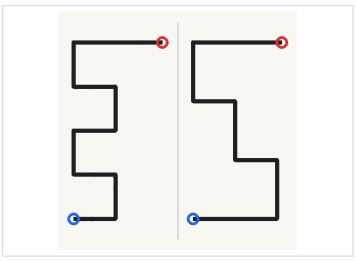
1 Given to agent: "Which terminal is farthest from green along the lines?"
Answer: red



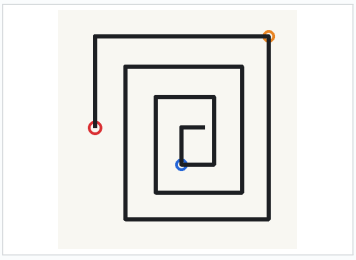
2 Given to agent: "Which colored node lies on the green-to-red route?"
Answer: blue



3 Given to agent: "Which colored terminal belongs to the green-started trace?"
Answer: orange



4 Given to agent: "Between the two ringed routes, which route has fewer bends from start to terminal?"†
Answer: right



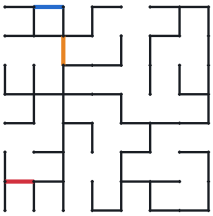
5 Given to agent: "Starting at the ringed port, which marked checkpoint is encountered first along the path?"†
Answer: orange

Figure A4: Tracing: the five example questions, each with one stored image. †: the stored image uses a reworded, more detailed question.

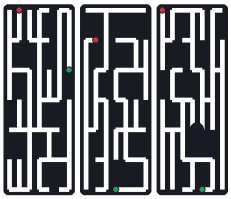
E.2.2 TOPOLOGY

Recover holes, enclosure, or connectivity from the image.

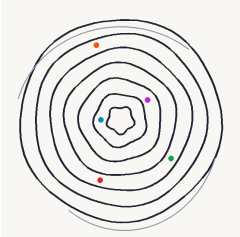
Topology: five example questions



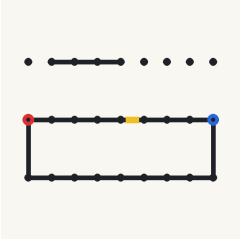
1 Given to agent: "Which colored link lies on a loop?"
Answer: blue



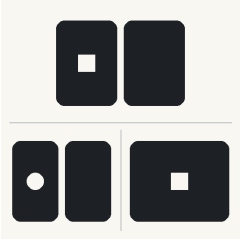
2 Given to agent: "Which maze connects green to red: left, center, or right?"
Answer: left



3 Given to agent: "Which colored dot is enclosed by the same number of black loops as green?"
Answer: red



4 Given to agent: "If the highlighted bridge is removed, do the two markers remain connected?"†
Answer: yes



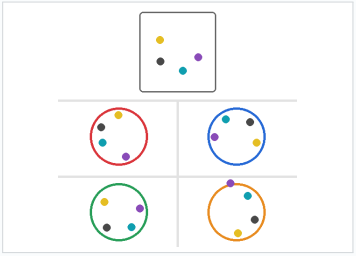
5 Given to agent: "Which candidate has the same numbers of connected components and holes as the reference?"†
Answer: left

Figure A5: Topology: the five example questions, each with one stored image. †: the stored image uses a reworded, more detailed question.

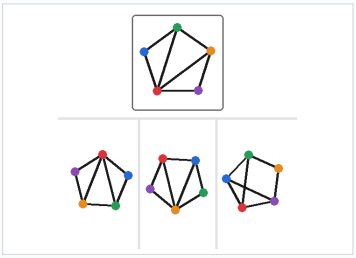
E.2.3 CORRESPONDENCE

Match parts or attributes across panels using the relation specified in the question.

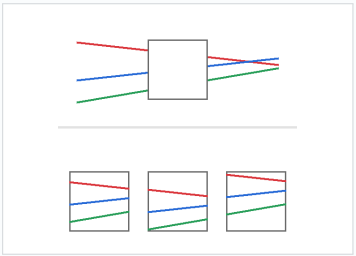
Correspondence: five example questions



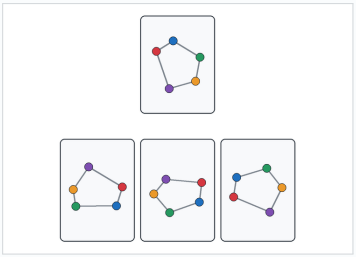
1 Given to agent: "What halo color belongs to the object matching the visual reference?"†
Answer: blue



2 Given to agent: "Which candidate preserves the same colored-part adjacency graph as the reference?"†
Answer: left



3 Given to agent: "Which candidate patch exactly completes the missing window so that every contour continues across the seam?"†
Answer: left



4 Given to agent: "Which candidate is the same multipart object as the reference after rotation?"†
Answer: right



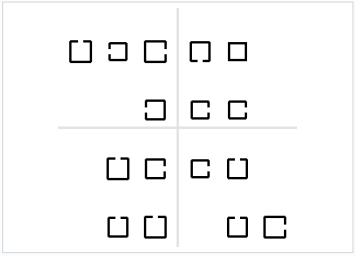
5 Given to agent: "Which colored fraction has a numerator matching the green denominator?"
Answer: cyan

Figure A6: Correspondence: the five example questions, each with one stored image. †: the stored image uses a reworded, more detailed question.

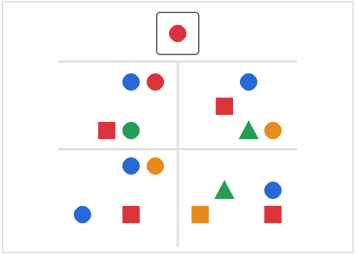
E.2.4 SEARCH

Find the unique item that satisfies the stated visual constraints.

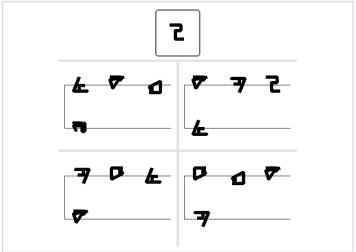
Search: five example questions



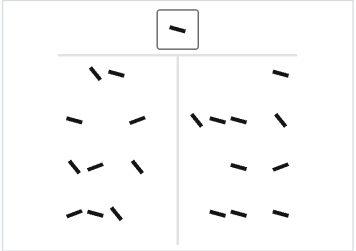
1 Given to agent: "Which quadrant contains the only completely closed contour?"†
Answer: B



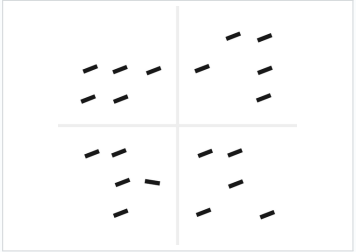
2 Given to agent: "Which region contains the object matching both attributes of the visual reference?"†
Answer: A



3 Given to agent: "Which sector contains the exact reference contour embedded in the clutter?"†
Answer: B



4 Given to agent: "Which half contains more elements matching the orientation shown in the reference box?"†
Answer: right



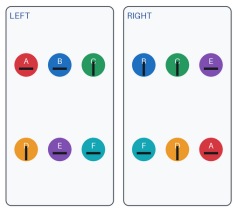
5 Given to agent: "Which quadrant contains the uniquely oriented dark bar?"†
Answer: C

Figure A7: Search: the five example questions, each with one stored image. †: the stored image uses a reworded, more detailed question.

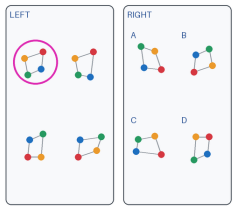
E.2.5 STATE TRACKING

Track identity, position, or state across a sequence of displayed steps.

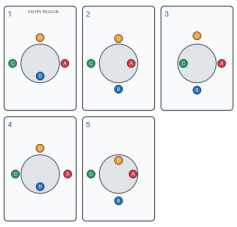
State tracking: five example questions



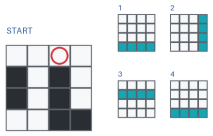
1 Given to agent: "Which object underwent the single declared visual change between the two panels?"†
Answer: B



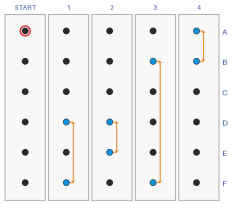
2 Given to agent: "Where does the initially ringed object appear in the final panel?"†
Answer: B



3 Given to agent: "Which labeled object makes the earliest outside-to-inside transition into the shaded region across the ordered panels?"†
Answer: A



4 Given to agent: "What is the final state of the ringed cell after the shown row and column toggles?"†
Answer: off



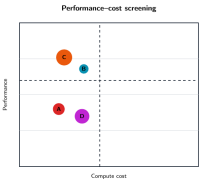
5 Given to agent: "Starting from the ringed slot, where does the token finish after the visible swaps?"†
Answer: B

Figure A8: State tracking: the five example questions, each with one stored image. †: the stored image uses a reworded, more detailed question.

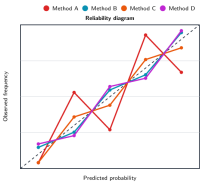
E.2.6 MEASUREMENT

Read visual quantities and compare them using the stated criteria.

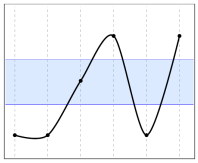
Measurement: five example questions



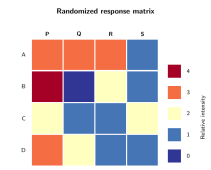
1 Given to agent: "Among points beyond both thresholds, which bubble is largest?"†
Answer: C



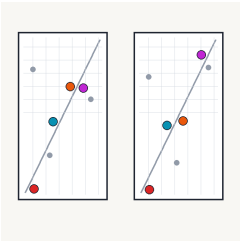
2 Given to agent: "Which model departs furthest from the dashed diagonal?"
Answer: A



3 Given to agent: "At how many checkpoints is the black trace fully inside the blue band?"
Answer: 1



4 Given to agent: "Which row is hottest in P but coldest in Q?"
Answer: B



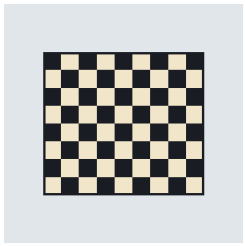
5 Given to agent: "Which colored sample moves from below the diagonal to above it?"
Answer: magenta

Figure A9: Measurement: the five example questions, each with one stored image. †: the stored image uses a reworded, more detailed question.

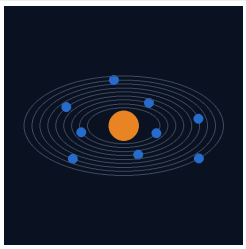
E.2.7 PRIOR CONFLICT

Report the features visible in the image, even when they differ from familiar expectations.

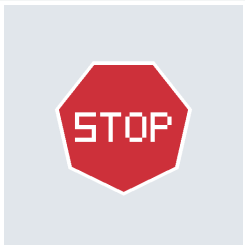
Prior conflict: five example questions



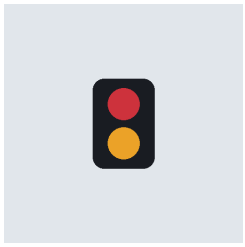
1 Given to agent: "How many columns are in the checkerboard?"
Answer: 9



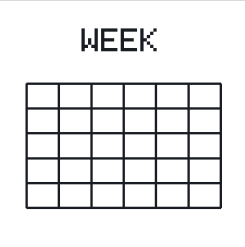
2 Given to agent: "How many blue planets are visible?"
Answer: 9



3 Given to agent: "How many sides does the STOP sign have?"
Answer: 7



4 Given to agent: "How many lamps are in the traffic signal?"
Answer: 2



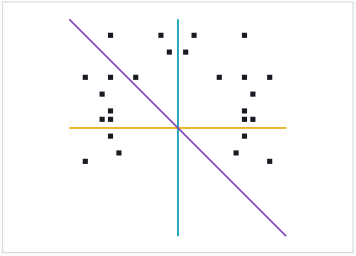
5 Given to agent: "How many columns are in the WEEK grid?"
Answer: 6

Figure A10: Prior conflict: the five example questions, each with one stored image.

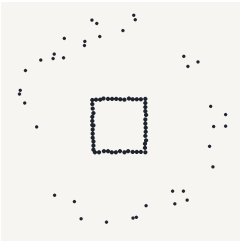
E.2.8 GLOBAL STRUCTURE

Combine spatially distributed evidence or negative space to identify the requested structure.

Global structure: five example questions



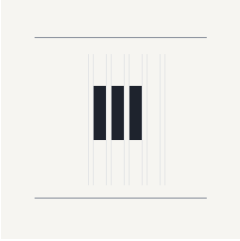
1 Given to agent: "Which colored line is a symmetry axis?"
Answer: cyan



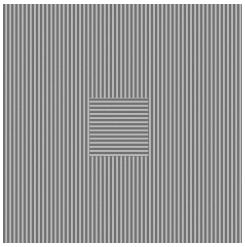
2 Given to agent: "Which symbol do the connected dots outline?"
Answer: square



3 Given to agent: "Which shape is formed by the empty region?"
Answer: diamond



4 Given to agent: "Which symbol continues behind the bars?"
Answer: square



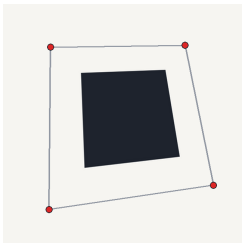
5 Given to agent: "Which shape is hidden in the texture?"
Answer: square

Figure A11: Global structure: the five example questions, each with one stored image.

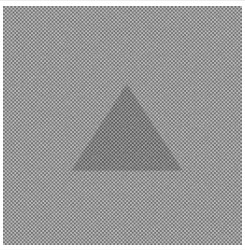
E.2.9 DECLARED TRANSFORM

Apply an explicitly stated rearrangement to determine the resulting visual structure.

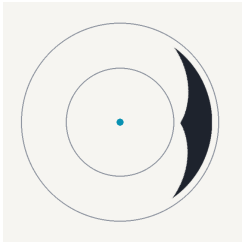
Declared transform: five example questions



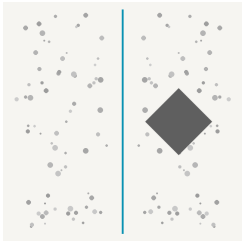
1 Given to agent: “Which symbol appears when the marked quadrilateral is viewed front-on?”
Answer: square



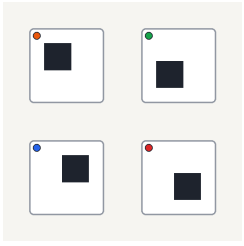
2 Given to agent: “Which symbol appears after blurring?”
Answer: triangle



3 Given to agent: “Which symbol appears when the ring is unwrapped?”
Answer: triangle



4 Given to agent: “Which symbol appears in the left-right difference?”
Answer: diamond



5 Given to agent: “Which symbol do the marked panels form?”
Answer: square

Figure A12: Declared transform: the five example questions, each with one stored image.

E.3 Model configurations

Profile-steered and spatially-steered generation use the same profiles, implementation demonstrations, world protocol, and sequential controller. Spatially-steered generation additionally includes requests about the spatial pattern of relevant image regions. The table below lists the coding-agent configurations used in both experiments. The model-feedback-steered experiment uses different campaign inputs and feedback, described in Sec. I.1.

Table A3: Exact coding-agent configurations used in profile-steered and spatially-steered generation. Two CLI versions are shown only where they differed between the experiments.

Configuration	Provider model identifier	Harness and CLI version
Sol (high)	`gpt-5.6-sol`	Codex `0.146.1` / `0.147.0`
Sol (max)	`gpt-5.6-sol`	Codex `0.147.0`
Luna	`gpt-5.6-luna`	Codex `0.147.0`
Opus 5	`claude-opus-5`	Claude Code `2.1.224`
DS-V4-Flash	`deepseek/deepseek-v4-flash-0731`	OpenCode `1.18.11`

E.4 Budgets and episode-hour accounting

Each coding-agent configuration receives a budget of six hours of cumulative agent-session time for each discovery profile in Secs. 4.1 and 4.2 and for each feedback example set in Sec. 4.6. This is not a six-hour wall-clock limit on the entire campaign. Agent sessions in Secs. 4.1 and 4.2 are capped at 20 minutes. Before each episode, the controller sets the session allowance to the smallest of the per-session cap, the unused campaign agent-time budget, and the remaining campaign wall time after reserving time for verification. It records this allowance in the agent's task instructions and configures Harbor (Harbor Framework Team, 2026) to enforce the same timeout. Episode-hours measure time spent in agent sessions, including provider waiting time, but exclude final verification and later analysis. Realized counts and rates appear in Tab. A11.

Coding-agent token usage and cost. Tab. A4 reports token usage and API costs for all three generation experiments, summed over episodes. Fresh counts input not read from cache, including Opus 5 cache writes, and Output includes reasoning tokens. DS-V4-Flash costs are recorded charges from its metered API. Sol, Luna, and Opus 5 ran under subscriptions, so we estimate their costs at public API prices as of September 25, 2026, as Fresh $\times$ cache-write rate + Cached $\times$ cache-read rate + Output $\times$ output rate.

Write/read/output rates in USD per million tokens are 5.00/0.40/20.00 for Sol, 0.25/0.02/1.20 for Luna, and 6.25/0.50/25.00 for Opus 5. Fresh tokens not written to cache would cost 4.00, 0.20, and 5.00 per million instead. Opus 5 sessions that hit the time limit reported only partial output, so its spatially-steered estimate is a lower bound.

Table A4: Coding-agent token usage (millions) and cost (USD), summed over episodes. Italicized costs with an asterisk are estimates.

	Profile-steered				Spatially-steered				Model-feedback-steered			
Configuration	Fresh	Cached	Output	Cost ($)	Fresh	Cached	Output	Cost ($)	Fresh	Cached	Output	Cost ($)
Sol (high)	31.7	582.4	6.0	*511.46**	35.2	744.3	6.3	*599.72**	8.0	129.5	1.6	*123.80**
Sol (max)	28.1	597.4	6.6	*511.46**	32.5	709.7	6.9	*584.38**	7.8	150.7	1.7	*133.28**
Luna	47.9	853.7	7.5	*38.05**	43.0	1,013.6	7.6	*40.14**	8.0	229.4	1.8	*8.75**
Opus 5	20.7	537.6	11.3	*680.68**	21.8	670.3	8.7	*688.90**	5.8	186.7	2.0	*179.60**
DS-V4-Flash	35.5	860.8	9.8	31.56	29.9	374.8	5.8	16.31	10.9	126.7	1.4	2.99

E.5 Post-generation replay procedure

The 200-scene replay is a later check of a generated world, separate from its initial final verification. We regenerate 200 instances using a fixed replay random seed and compare the forward and inverse answers on scenes not excluded by the declared ambiguity rule. The agreement and answer-class requirements are described in Sec. D.4. We record each world as passing replay, failing with a

preserved failure record, or missing a replay record. Missing records are reported separately rather than counted as measured failures. Experiment-specific counts appear with the generation results.

Scenes excluded by the ambiguity rule. Each replay record marks the scenes excluded by the world's declared ambiguity rule. Tab. A5 summarizes the excluded fraction of the 200 replay scenes per replay-verified world. Most worlds exclude nothing, and worlds excluding more than half of their scenes are rare in every experiment. Worlds generated by Opus 5 exclude more scenes than those of other configurations: in profile-steered generation, their median excluded fraction is 12.5% (mean 12.1%), whereas the other four configurations have a median of zero and means from 0.5% to 7.0%. These counts cover only scenes that a world emits and then excludes, because scenes that a sampler rejects before emitting them cannot be observed here.

Excluded scenes are not used as evaluation instances. The five evaluation instances per world are drawn only from non-excluded replay rows; we confirmed that none of the 6,490 profile-steered and 4,370 spatially-steered evaluation instances is an excluded scene. The training pipeline reads the same filtered replay manifests, which contain no excluded rows in any profile-steered replay record.

Table A5: Fraction of the 200 replay scenes excluded by each world's declared ambiguity rule, over replay-verified worlds. Columns >x% count worlds whose excluded fraction exceeds x%.

Experiment	Worlds	Median	Mean	P90	Max	>10%	>25%	>50%
Profile-steered	1,301	0.0	4.1	16.5	62.0	223	43	4
Spatially-steered	875	0.0	4.9	18.0	82.5	176	46	5
Model-feedback-steered	264	0.0	2.2	8.9	34.5	24	6	0

F How the Inverse Programs Solve Questions

This section reports a controlled image-substitution test (Sec. F.1), the repairs that inverse programs prompted during generation (Sec. F.2), a source analysis of 225 programs (Sec. F.3), and worked examples (Sec. F.4). Unlike the executed tests, the source analysis relies on model-assisted judgments.

F.1 Controlled image-substitution test

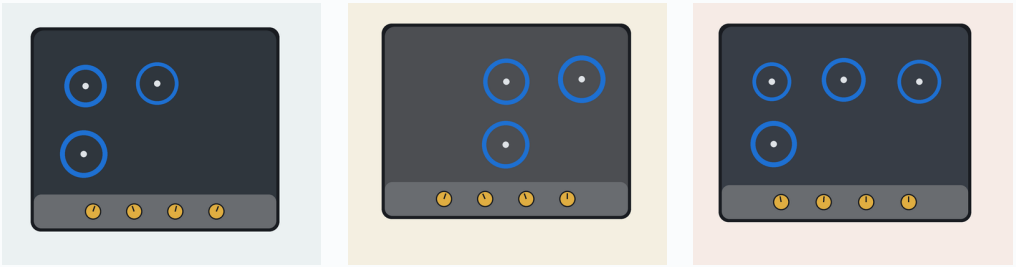
We test whether the inverse and forward program answer comparison correctly detects answer-changing rendering errors. For each selected world, we replace the intended image with an image from a different scene while leaving the intended scene specification and forward answer unchanged.

We select 100 replay-verified profile-steered worlds, 20 from each coding-agent configuration, covering all nine profiles in Tab. A2 For each world we run three trials. Each trial takes an intended scene A and tests three substituted images: a different scene B with the same generated answer (but with a different image than A), a scene C with a different visible answer, and the original rendered image of A. All three use the question text of A. The forward program and prompt always receive A.

Every check behaved as intended in all 300 trials. When the image was swapped for one with a different answer, the checks that ignore the image still passed and the inverse answer correctly disagreed with the forward answer. When the image was swapped for one with the same answer, or kept as the same image, the inverse and forward answers agreed.

Stovetop Burner Count *(counting decision)*

The first two images both visibly support the answer three, whereas the faulted image supports four. The forward program still reads the intended scene in all three cases. The inverse answer agrees with it for the answer-preserving substitution and disagrees for the answer-changing substitution.

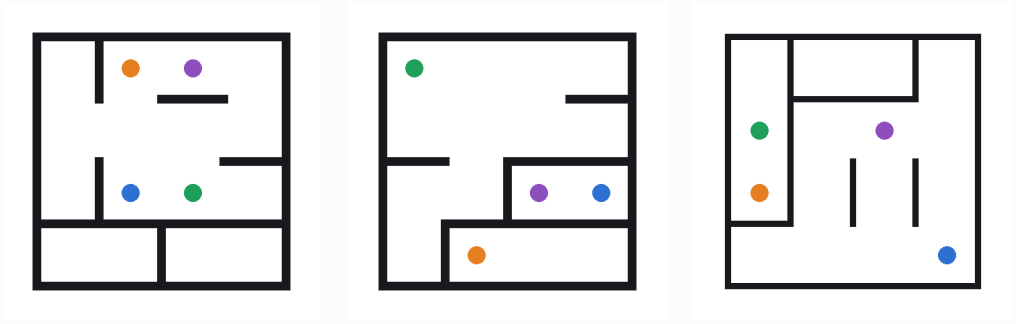


(a) Intended render: answer 3. (b) Different render, same answer 3. (c) Faulted render: visible answer 4.

Figure A13: A controlled renderer change for *Stovetop Burner Count*. The forward program keeps the intended scene's answer, while a deterministic fault swaps in another valid image. The inverse answer agrees with the forward answer in (b) but disagrees in (c), because the new image shows a different answer.

Marker Region Partition *(topology)*

Marker Region Partition asks whether four markers occupy exactly two connected components of white space. The intended scene and the substituted control differ in wall geometry and marker placement but both answer *no*. The faulted scene changes the visible connectivity so that the answer becomes *yes*; the forward program still declares *no*, and only the inverse answer changes (Fig. A14).



(a) Intended render: answer no. (b) Different render: answer no. (c) Faulted render: answer yes.

Figure A14: Topology substitution for *Marker Region Partition*. The inverse answer remains *no* despite the substantial pixel change from (a) to (b), then changes to *yes* in (c) after the wall arrangement changes the number of connected marker regions.

F.2 Repairs prompted by the inverse program during generation

While the controlled test above measures sensitivity, this analysis looks at what the inverse program did in the actual campaigns. We searched the recorded tool outputs, not the agents' own explanations, of every agent session in profile-steered and spatially-steered generation for a total of 2,605 sessions across the five coding-agent configurations. No session transcript was missing.

Definitions. A session contains an *inverse failure* if the inverse program, run on an ordinary generated scene, either returned an answer different from the forward program (an answer disagreement) or raised an error or abstained. We excluded four kinds of events that are not failures on ordinary scenes. These are deliberate tests on blank, corrupted, mutated, or masked images, import and syntax errors, failures of other checks, and agents re-reading reports from earlier episodes. A failure counts as *repaired* if the agent then edited the world's code and a later check in the same session passed. We classify each repair by the files edited between the failure and the passing check. A hand check of 30 randomly selected flagged sessions, ten from each of the three agent harnesses' log formats, found all 30 to be genuine. Recall was not measured, so the counts are lower bounds.

Table A6: Inverse failures and repairs by coding-agent configuration, pooled over profile-steered and spatially-steered generation. *Disagreement*: sessions in which the inverse program returned a different answer from the forward program; *any failure* also counts errors and abstentions. *Repaired* counts accepted worlds whose session contains a failure followed by a repair; *All* counts all accepted worlds; *Share* is Repaired/All.

	Agent sessions			Accepted worlds		
Configuration	All	Disagreement	Any failure	Repaired	All	Share (%)
Sol (high)	751	29	248	246	743	33.1
Sol (max)	517	14	158	144	490	29.4
Luna	429	57	194	141	344	41.0
Opus 5	451	41	66	44	374	11.8
DS-V4-Flash	457	99	138	74	283	26.1
Total	2,605	240	804	649	2,234	29.1

Table A7: What the agent changed in each repair, classified by the world files edited between the first inverse failure and the passing check. *Other* combines a question-text or answer-mapping edit with renderer or sampler edits.

Configuration	Repairs	Inverse only	Inverse + renderer/sampler	Renderer/sampler only	Other
Sol (high)	248	137	63	42	6
Sol (max)	155	88	47	16	4
Luna	188	90	67	28	3
Opus 5	65	23	22	15	5
DS-V4-Flash	105	48	32	20	5
Total	761	386	231	121	23

Results. Of the 2,605 sessions, 804 (30.9%) contain an inverse failure and 240 (9.2%) an explicit answer disagreement (Tab. A6). Of the 804 sessions with a failure, 761 (94.7%) were followed by a repair. By experiment, repaired sessions account for 370 of 1,331 accepted profile-steered worlds (27.8%) and 279 of 903 accepted spatially-steered worlds (30.9%). Nearly half of the 761 repairs (375, 49%) edited the renderer or scene sampler, alone or together with other files; the rest changed only the inverse program (Tab. A7). Fig. A15 shows three further cases, from Sol (high) and DS-V4-Flash, in which the image itself was wrong or hard to read.

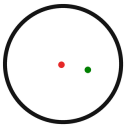
Before repair	After repair	
		**(a)** Sol (high), "How many separate white holes are fully enclosed inside the blue capital B?" Before: forward 3, inverse 2; check **fails**. The middle hole leaks into the background at the waist of the letter. Repair: the renderer closes the wall. After: 3 and 3; check **passes**.
		**(b)** DS-V4-Flash, "Which colored dot is enclosed by more of the closed loops?" Before: forward red (four loops against three), inverse green; check **fails**. Rings were drawn smallest first, so each larger ring's white fill erased the inner rings (51 of 200 test scenes). Repair: draw rings largest first. After: red and red; check **passes**.
		(c) DS-V4-Flash, "How many snowballs make up this snowman?" Before: forward 4, inverse 3; check **fails**. Balls were stacked upward without a height check, so the head fell mostly above the canvas. Repair: resample sizes until the stack fits. After (same seed, new scene): 4 and 4; check **passes**.

Figure A15: Three further cases, from two coding-agent configurations, in which the inverse program failed during generation because the image itself was wrong or unreadable, and the agent repaired the renderer or sampler. Each row shows the same seed before and after the repair; images are regenerated from the agents' own code. Question texts are abbreviated.

Scope. A repair follows a failure in time, but this does not prove that the failure alone caused it. Many failures reflect an inverse program too strict or fragile for a valid image rather than a wrong image, and the agent's fix may then change only the inverse program. Because renderer files sometimes also compute the expected answer, an edit to them is not always a pure drawing change. Tool output was truncated in many Luna sessions, so its counts likely undercount failures and repairs more than those of other configurations.

F.3 Source analysis of inverse programs

To see how inverse programs answer questions from images, we asked Luna to read the code of 225 worlds. Each analysis session received one world's question text, answer options, scene specification, generation code, renderer, forward and inverse programs, three stored instances, and run metadata. It traced how the image and the two answers were produced and filled a fixed structured schema. Luna ran at maximum reasoning effort and read the code without executing it. It also generated 45 of the 225 worlds.

The worlds are replay-verified worlds from profile-steered generation, five for each of the 45 pairings of coding-agent configuration and profile. We first chose two per pairing (90 worlds) and later added three more per pairing (135 worlds), preferring code that shared the fewest identifiers with worlds already chosen for the same pairing (maximum Jaccard distance). This widens the range of implementations, so the counts below describe this sample rather than all 1,301 worlds. For the first 90 worlds, we also ran each inverse program on its three stored images: all 270 answers matched the saved records.

Most results below are Luna's judgments from reading code, not measured failure rates. The exceptions are the code-size metrics in Sec. F.3.2, which are computed directly from the source.

F.3.1 What the inverse program can access

The inverse program must answer from the image alone (App. D). In all 225 worlds, the inverse program's entry point takes only an image, cannot access the scene specification or recorded answer, and imports no forward-program code (Tab. A8). The analysis judged the two implementations clearly separate in 204 worlds and partly separate in 21. None of the 21 reads the scene specification or the recorded answer; instead, their inverse programs re-implement the question's public rules, such as the stated transform, the answer vocabulary, glyph templates, or the decision procedure. Both groups often copy public constants such as color palettes (199 of the 204 clearly separate programs do so).

Table A8: What the 225 inverse programs can access. All rows come from the analysis model's reading of the source; the last two rows are qualitative judgments.

Check	Worlds
Entry point takes only an image	225 / 225
No access to the scene specification	225 / 225
No access to the recorded answer	225 / 225
No import of forward-program code	225 / 225
Answer evidence judged visible in the image	225 / 225
Implementations judged clearly separate	204 / 225

F.3.2 How large the inverse programs are

We compare each inverse program with the file that holds the world's sampler, renderer, and forward program, using metrics computed from the Python syntax tree without running the code (Fig. A16). The inverse program is usually the smaller of the two, with a median of 94 non-blank lines against 159, 5 functions against 11, and about two-thirds as many syntax nodes. It is larger in only 17 of 225 worlds by line count. However, it contains more branches in 129 worlds (median 10 against 8), which reflects the checks it makes before committing to an answer.

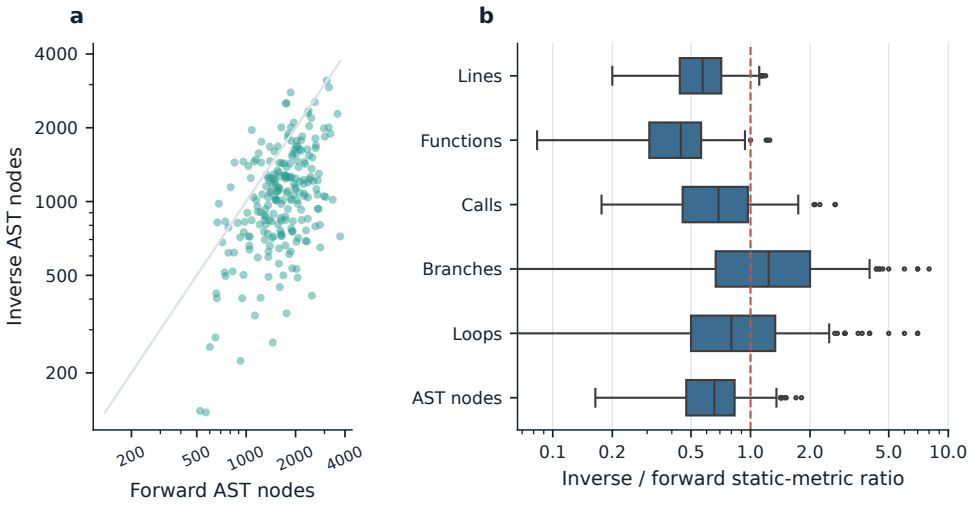


Figure A16: Code size of the inverse program compared with the file holding the sampler, renderer, and forward program ($N = 225$). (a) Syntax-tree node counts. The diagonal marks equal size. (b) Inverse-to-forward ratios. the dashed line marks parity. Metrics come from parsing the source, not from running it.

F.3.3 Steps and algorithms

Every inverse program first detects visible elements such as colors, shapes, lines, or text. Most then apply a question-specific rule (218 of 225), choose among the answer options (199), and check for missing or ambiguous evidence (210). 132 also organize their measurements into a grid, graph, ordering, or grouping (Fig. A17).

At a finer level, the analysis assigned each program to categories from a list of 30 algorithm types, 1,826 assignments in total, each linked to a supporting function in the source. A typical program uses eight categories (mean 8.12; range 3–17). The most common are answer checks and selection, color thresholding, region statistics, and connected-component labeling (Fig. A18).

No inverse program imports OpenCV, scikit-image, or a learned vision model; the programs write these operations by hand, mostly with NumPy and the Python standard library. Mapped to their closest OpenCV equivalents, 88% of programs contain a color-range mask (`cv2.inRange`), 82% a connected-component pass, 58% image moments, and 41% contour extraction (Fig. A19).

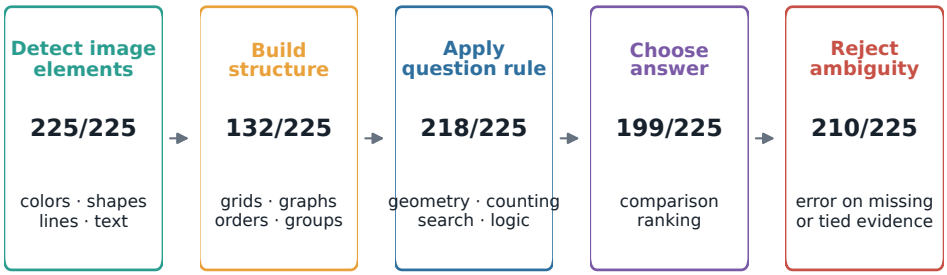


Figure A17: Common steps in the 225 inverse programs. A world may use several steps, combine them, or answer directly from measurements.

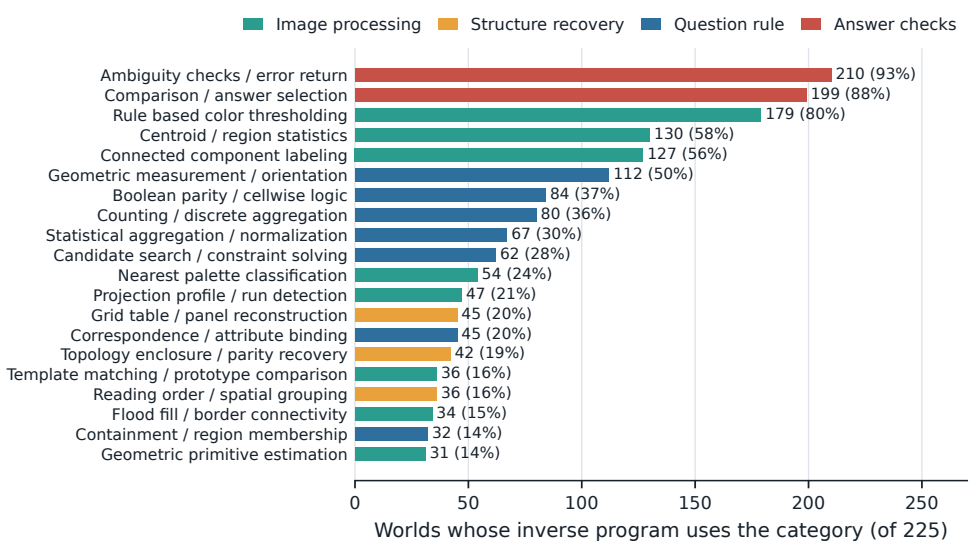


Figure A18: The 20 most frequent algorithm categories. Each bar counts worlds using the category; one inverse program may use several.

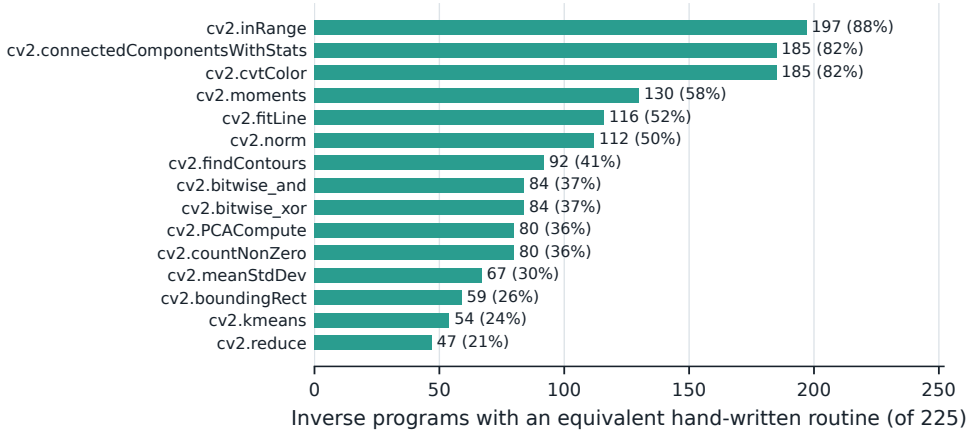


Figure A19: Closest OpenCV equivalents of the hand-written inverse routines, as identified by the analysis ($N = 225$). No program imports OpenCV. Each bar counts programs that implement an equivalent operation themselves.

F.3.4 Differences across profiles

Inverse programs differ with the kind of question rather than following one fixed parser (Fig. A20). Topology recovery is concentrated in the topology profile, constraint search in the search profile, attribute matching in the correspondence profile, counting in the state-tracking profile, and grid reconstruction in the declared-transform profile. Each profile contributes 25 worlds, so the rows are directly comparable, although they are not estimates for the full collection.

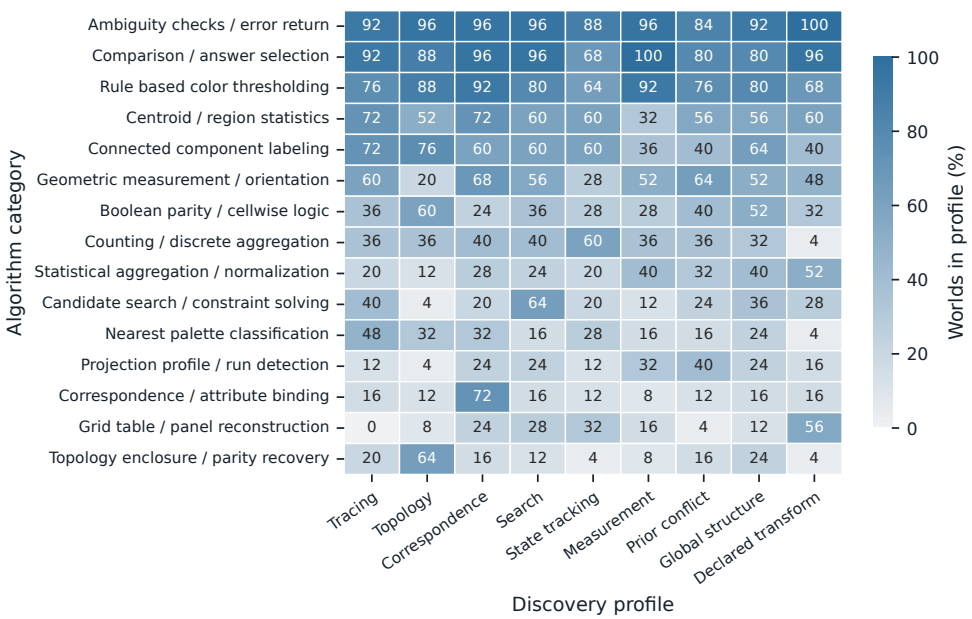


Figure A20: Inverse-algorithm use by profile. Each cell is the percentage of that profile's 25 worlds using the category. A world may use several categories, so rows need not sum to 100%.

F.3.5 Agreement with the forward rule

The analysis also judged whether each inverse program keeps the information and the rule that the answer needs. It judged that 204 programs discard scene detail the answer does not need while keeping what it does, 14 could discard needed information, and 7 do not reduce the scene before deciding. For the decision rule, it judged 202 forward–inverse pairs as matching (36 identical and 166 equivalent in a different representation), 21 as approximate, and 2 as partial. The schema names these labels but does not define *approximate* or *partial* further. These are single judgments from reading code rather than measured errors, and all 225 worlds also passed the executed agreement checks during verification and replay.

F.4 Worked examples

The following ten examples show how the forward and inverse programs solve the same question using different inputs. Each includes the original question, stored images and answers, and descriptions of scene generation and both programs. We also describe approximations and failure cases. The examples span all nine profiles and come from the initial 90-world sample, whose inverse answers were checked on three stored images per world. For six of the examples, a second figure runs the archived inverse program on the first stored image and shows its intermediate results step by step.

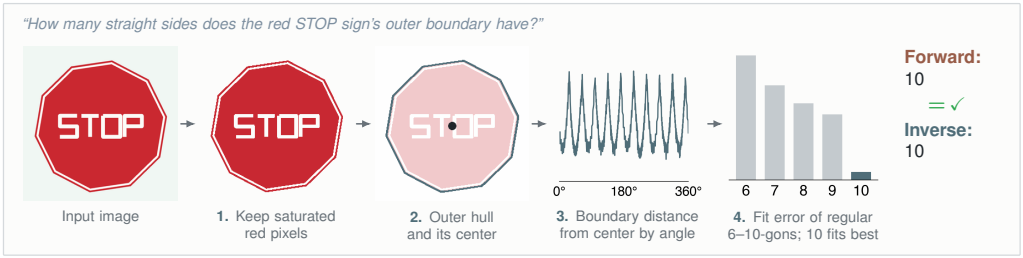


Figure A21: Inverse-program steps on the first stored image above, repeated from Fig. A32 in the main text.

Compulsory Route Checkpoint *(tracing)*

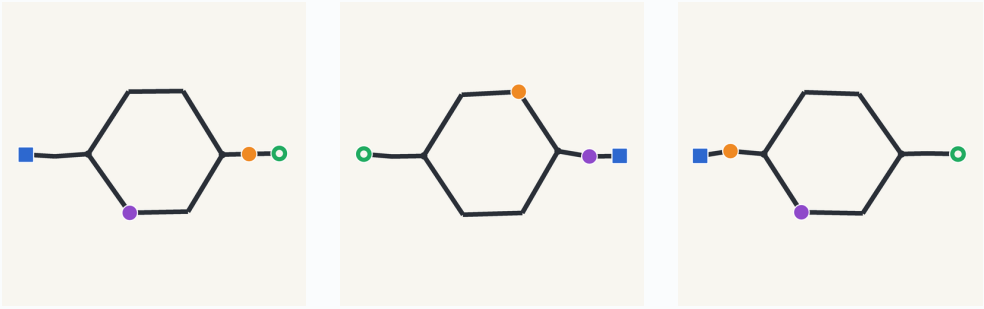


Figure A22: Three transformations and checkpoint assignments for the same public rule. Stored outputs from left to right are orange, purple, and orange.

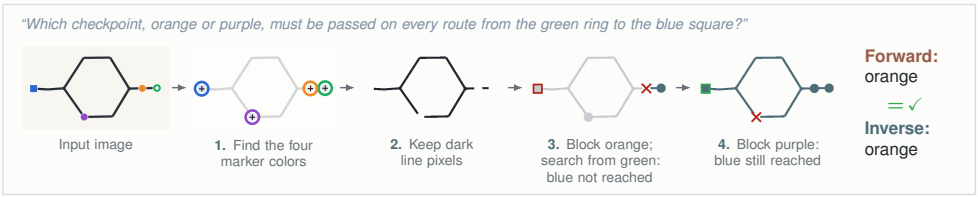


Figure A23: Inverse-program steps on the first stored image above, drawn from the values its own functions return.

Question. "Follow the dark network from the green ring to the blue square, taking any branch at a junction. Which checkpoint, orange or purple, lies on every route?"

Scene generation. The scene contains a graph with ten nodes and ten edges. Its paths split and rejoin between a start and a destination. One colored checkpoint is placed on one of the two nodes that every route must cross; the other is placed on one of four branch nodes. The sampler varies node positions by up to five pixels, rotates the graph in quarter turns, and may reflect it. These changes preserve connectivity and the answer.

Renderer and forward program. The renderer draws the graph at four-times resolution, including dark edges and junction hubs, colored checkpoint disks, a green start ring, and a blue rounded destination square, and then downsamples to a 512×512 image. The forward program operates on the exact edge list. For each checkpoint it deletes that node, builds the remaining undirected adjacency map, and runs breadth-first search (BFS) from start to destination. The sole deletion that makes the destination unreachable determines the answer.

Inverse program. The inverse program tests each checkpoint by removing it from the visible network and checking whether the start can still reach the destination. It detects the four marker colors by RGB distance, requires at least 80 pixels per marker, and estimates their centers by averaging pixel positions. Network pixels must have all three channels below 105 and a channel spread below 35.
A marker covers part of the dark line, so thresholding alone would break the network at every marker. The program fills a radius-16 disk around each marker to reconnect the line, except at the checkpoint being tested. It then uses breadth-first search through horizontal and vertical pixel neighbors. It returns the color of the sole checkpoint whose removal disconnects the start and destination. Missing markers, blocked endpoints, or no unique answer raise `ValueError`.

Comparison with the forward program. Both programs test whether removing a checkpoint disconnects the route. The forward program uses the stored graph. The inverse program uses marker positions and connected pixels. It does not recover the original node names, edge list, pose transform, or random seed.

Region Hole Selection *(topology)*

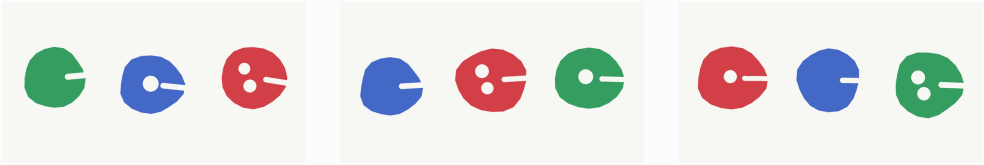


Figure A24: The colors, positions, boundary shapes, and hole assignments all change. Stored outputs from left to right are blue, green, and red.

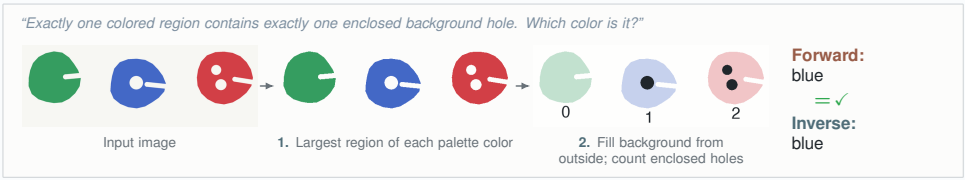


Figure A25: Inverse-program steps on the first stored image above, drawn from the values its own functions return.

Question. "Exactly one colored region contains exactly one enclosed background hole. Which color is it? Open indentations and the outside background do not count."

Scene generation. The sampler assigns zero, one, and two holes to three colored regions in random order. Each region has a wavy, 20-vertex polygon boundary; its position, width, height, and boundary shape vary. Circular cutouts form the holes. Each region also has an open notch connected to the outside background. The question requires counting enclosed holes while ignoring these notches.

Renderer and forward program. The renderer paints the polygons, erases the circular cavities and open notch channels, and downsamples a three-times supersampled rendering to 720×380. The forward answer does not inspect the image: it selects the unique region whose stored cavity list has length one. A separate clearance calculation excludes scenes with less than six pixels of approximate cavity clearance under the ambiguity rule.

Inverse program. The inverse program counts enclosed background areas within each colored region. It assigns pixels to the nearest red, green, blue, or background color and keeps the largest eight-connected component for each region. It crops each region with a two-pixel margin, then fills background pixels reachable from the crop boundary. This fill reaches the outside and open notches but cannot enter an enclosed hole.
The program counts the remaining enclosed components with at least 120 pixels and returns the unique color with one hole. Missing components or more than one possible answer cause an error.

Comparison with the forward program. The forward program counts stored circular cutouts; the inverse program counts enclosed background components. Recovering the original polygon vertices or the exact circles is unnecessary for this decision.

Shape–Pip Binding *(correspondence)*

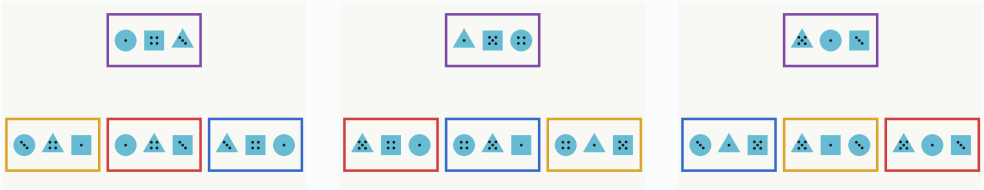


Figure A26: The shape order, rotations, pip assignments, and correct border color vary. Stored outputs from left to right are blue, gold, and red.

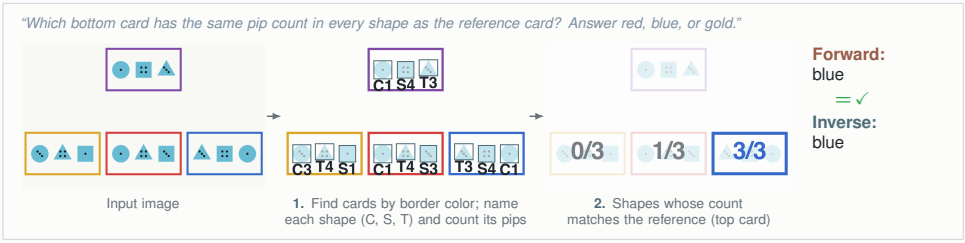


Figure A27: Inverse-program steps on the first stored image above, drawn from the values its own functions return.

Question. "The top card is the reference. Which border color marks the lower card where the circle, triangle, and square preserve their matching pip counts?"

Scene generation. The sampler assigns distinct dot counts from one through five to a circle, triangle, and square. One lower card preserves all three counts. The other two rearrange them so that at most one shape retains its original count. The correct card's position and border color vary. Each card shuffles its shapes independently. Triangles rotate by 0, 120, or 240 degrees and squares by 0 or 90 degrees. The answer depends on the dots within each shape, regardless of its position.

Renderer and forward program. The renderer draws four 300×170 cards on a 960×590 canvas, using fixed dot patterns within the shapes. The forward program compares the stored dot count for each shape with the reference. Exactly one card must match all three counts, and each other card must match at most one.

Inverse program. The inverse program identifies each shape and counts its dots. It finds the purple reference border and the three option borders by color, requiring at least 100 pixels per border. Within each card, it extracts three connected shape regions of at least 700 pixels and dot regions of at least 30 pixels, using horizontal and vertical pixel connections.
The fraction of each shape's bounding box filled by the shape determines its identity: above 0.88 is a square, above 0.65 is a circle, and the remainder is a triangle. A dot is assigned to a shape when its center falls strictly inside that shape's bounding box. The program compares the three shape–count pairs with the reference. The selected card must match all three, and every other card must match at most one.

Failure checks. Missing borders, the wrong number of shapes, duplicate shape identities, out-of-range pip counts, or an incomplete/nonunique correspondence all raise `ValueError`. The inverse program reconstructs the shape-to-count binding but does not recover the sampled display order or rotations as scene variables.

Contour Match in Clutter *(search)*

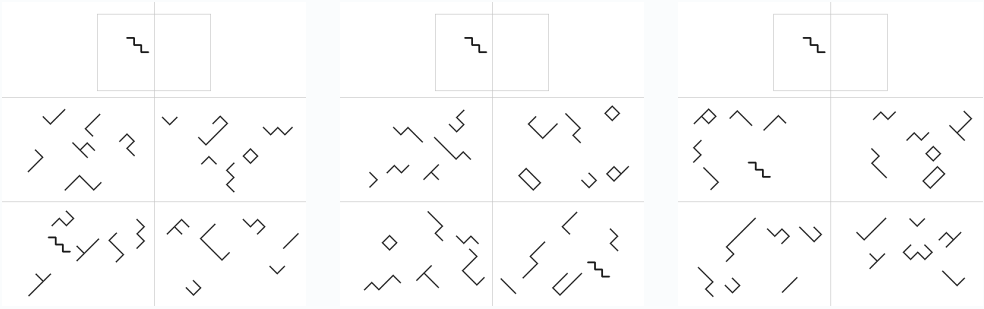


Figure A28: The target sector, target translation, and 23 distractor fragments vary. Stored outputs from left to right are bottom-left, bottom-right, and top-left.

Question. "Which of four sectors contains an exact copy of the dark reference line shown in the gray box above the sector grid?"

Scene generation. The reference is a fixed six-vertex, five-segment axis-aligned staircase. One of four sectors is sampled uniformly and receives a translated exact copy. Each sector contains six fragments, so the remaining 23 are distractors made from four to six randomly signed diagonal steps. Their locations are sampled subject to a target bounding-box clearance. Target position, distractor shape, and fragment order vary, but equality of the normalized target vertex sequence to the reference determines the answer.

Renderer and forward program. The renderer draws all polylines into a Boolean mask, overlays the gray reference frame and sector dividers, and emits a 512×512 image. The forward program compares normalized vertex lists exactly. It also rasterizes non-target fragments to compute a margin equal to one minus the best distractor coverage, but the sector label comes from exact vector equality.

Inverse program. The inverse program crops each contour to its bounding box and compares it with the cropped reference. It identifies dark pixels by a channel sum below 150, which excludes the lighter gray frame and dividers. The reference and the four sectors are located at fixed image coordinates.
Within each sector, the program groups dark pixels into eight-connected components. It aligns each component's crop with the reference at the top left and measures how much of the reference it covers. The highest component score becomes the sector's score. Cropping allows the comparison to ignore the contours' absolute positions.

Approximation and failure behavior. The inverse program compares raster coverage, whereas the forward program compares vertex sequences exactly. It breaks ties by sector order and requires no minimum gap between the best and second-best scores. An empty reference or all-zero scores can therefore return top-left rather than an abstention. The three stored examples have a large coverage gap, and their replayed answers match the records. This does not test the failure cases above. This example was classified as an approximate inverse program. The corruption check in Sec. D.4 tests the submitted checking script, not abstention by the inverse function alone.

Capacity Pour Replay *(state tracking)*

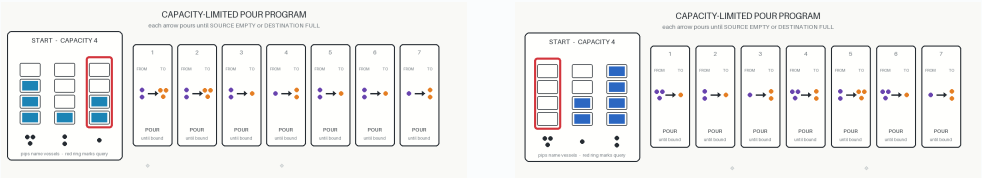


Figure A29: Initial levels, display order, query identity, and seven directed pours vary. Stored outputs from left to right are 0 and 2.

Question. "Starting from the shown levels, execute seven capacity-limited pours from left to right. What is the final level of the red-ringed vessel?"

Scene generation. The scene contains three vessels, each holding at most four units of liquid. Their initial levels sum to six. It also specifies the queried vessel and seven ordered source–destination pairs. The vessels can appear in different left-to-right positions; one, two, or three dots identify them across these rearrangements. Colors, canvas offsets, card spacing, and decorations vary. The sampler requires that reversing the pours changes the answer, at least one pour moves no liquid, and at least three pours move liquid. It samples scenes that produce the requested final answer class.

Renderer and forward program. For operation (s, t), the forward transition moves $\min(x_s, 4 - x_t)$ units, updates both vessel levels, and records whether the pour was a no-op, emptied the source, filled the target, or reached both boundaries. The renderer places four-cell gauges and pip identities in the initial panel, then seven numbered cards with purple source pips, orange destination pips, and arrows on a 1320×500 canvas. The forward program replays the exact operation list and returns the final level of the queried vessel.

Inverse program. The inverse program reads the initial levels and the ordered pouring instructions, then simulates the pours. It first extracts small round, four-connected pip components beneath the gauges, groups them by horizontal position, and requires three groups with one, two, and three dots. These counts identify the vessels independently of their display positions. Saturated rectangular gauge components provide the initial levels, which must remain within capacity and sum to six. A tall sparse red component identifies the query slot. In the card region, purple and orange pip components are grouped left-to-right into seven paired source/destination instructions. After reading the vessel identities and instructions, the inverse program applies the same pouring rule as the forward program.

Failure checks. The program must identify the vessels across panels and execute the pours in order. It raises `PixelParseError` for merged or missing dot groups, invalid levels, misaligned query rings, missing or incorrectly paired cards, and simulations that violate vessel capacity or total liquid volume.

Boxplot Dual Asymmetry *(measurement)*

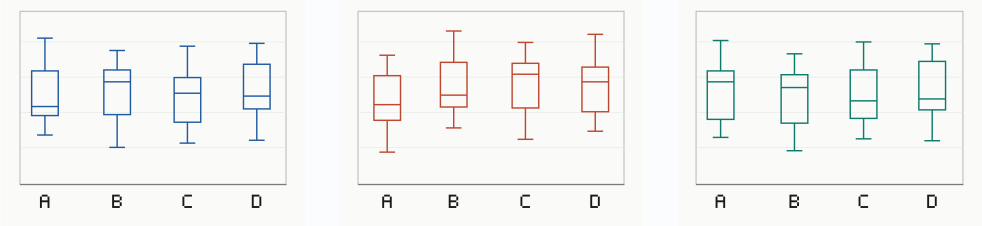


Figure A30: The label satisfying both conditions, geometry, and ink color vary. Stored outputs from left to right are A, B, and C.

Question. "Which sample has its median in the lower half of its box and an upper whisker longer than its lower whisker?"

Scene generation. Each of the four boxplots stores five vertical positions: the upper cap, box top, median, box bottom, and lower cap. The sampler creates one boxplot for each combination of the question's two conditions: both true, only the first true, only the second true, or neither true. It randomly assigns these cases to labels. Box height, total whisker length, position, colors, and the size of the differences vary. Every eleventh random seed places a median or whisker difference near the decision threshold.

Renderer and forward program. The forward program measures how far the median lies below the box midpoint and subtracts the lower-whisker length from the upper-whisker length. Both values must be positive for a boxplot to satisfy the question. Its score is the smaller of the two values. The renderer draws the four boxplots and gray guides on a 640×480 canvas. The ambiguity rule excludes scenes with a median or whisker comparison within six pixels of its threshold, or a gap of at most six pixels between the two highest scores.

Inverse program. The inverse program finds the five horizontal lines in each boxplot and measures their vertical positions. It separates colored ink from gray guides and black labels using a channel spread of at least 55, maximum channel value at most 225, and minimum value at most 170. It groups occupied columns across gaps of at most two pixels to locate four boxplots in A–D order.
Within each boxplot, rows containing at least 18 colored pixels identify the horizontal lines. Adjacent rows are merged, and their mean positions give the five measurements. The program then computes the median displacement, whisker-length difference, and score in the same way as the forward program.

Decision and limitation. Exactly one boxplot must satisfy both conditions: its score must be positive and every other score negative. Otherwise the inverse program raises `ValueError`. This check does not apply the forward program's six-pixel ambiguity threshold. It checks which side of each threshold the measured values fall on, without requiring the same distance from that threshold.

STOP-Sign Side Count *(prior conflict)*



Figure A31: Side count, rotation, size, translation, red tone, and border width vary. Stored outputs from left to right are 10, 6, and 7.

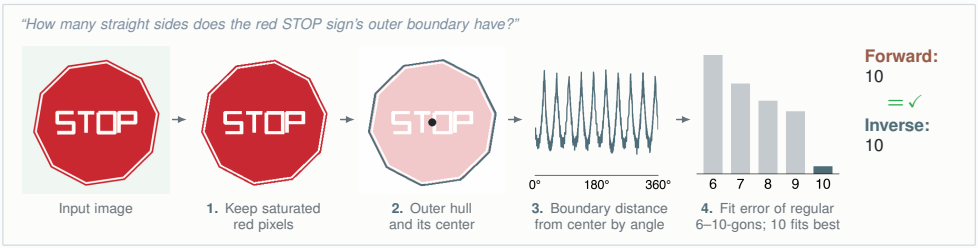


Figure A32: Inverse-program steps on the first stored image above, drawn from the values its own functions return.

Question. "How many straight sides does the red STOP sign's outer boundary have? Ignore the letters and count the visible boundary."

Scene generation. The sampled side count ranges from six through ten. Center, rotation, projected area, red palette, pale background, and white inset-border width also vary. The polygon radius is derived from the sampled target area and the regular-polygon area formula; consequently, area is deliberately held in an overlapping range across side counts and cannot reliably predict the side count. The word STOP is drawn with simple white strokes and does not affect the answer.

Renderer and forward program. The renderer constructs equally spaced vertices for a regular polygon, draws the red fill, inset border, and lettering at four-times resolution, and downsamples to 420×420. The forward program returns the stored side count directly. The intended challenge is therefore visible-boundary measurement in conflict with the familiar expectation that a stop sign has eight sides.

Inverse program. The inverse program compares the red boundary with regular polygons having six through ten sides. It selects red pixels using $R > 125$, $R - G > 70$, and $R - B > 55$, requires at least 22,000 such pixels, and rejects shapes within 20 pixels of the image boundary. It constructs a convex hull from the outermost red pixels in each row and column and estimates its center.
It then measures the boundary's distance from the center in 720 angular bins. At least 75% of the bins must contain measurements; missing values are interpolated around the boundary. For each possible side count, it tests 181 rotations and fits the scale. The root-mean-square difference between the measured and fitted boundaries determines the best match. The white border and letters are excluded by the red-pixel test.

Decision and abstention. The best residual must be at most 1.25, beat the runner-up by at least 1.5, and be at least three times smaller than the runner-up. Otherwise the inverse program raises `OracleAbstention`. The inverse program fits the visible boundary and returns 10, 6, and 7 for the three shown scenes.

Oriented-Dash Topology *(global structure)*

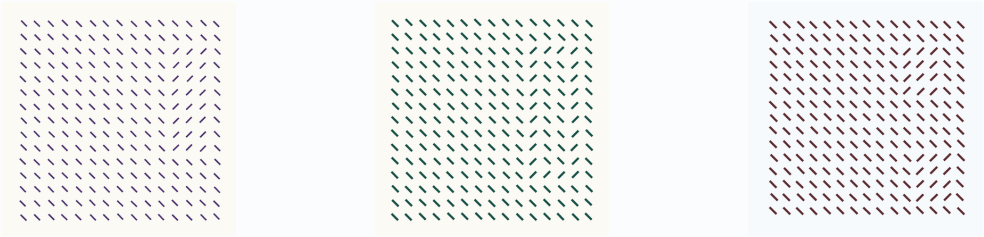


Figure A33: The local dash field encodes each of the three global topology classes. Stored outputs from left to right are one island, one ring, and two islands.

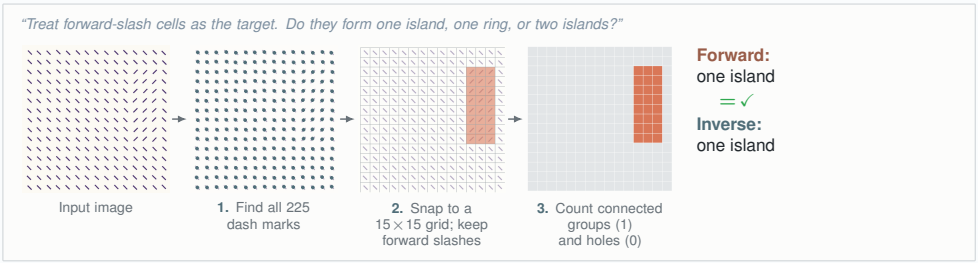


Figure A34: Inverse-program steps on the first stored image above, drawn from the values its own functions return.

Question. "Treat forward-slash cells as the target and connect only horizontal or vertical neighbors. Do they form one island without a hole, one ring, or two islands?"

Scene generation. The scene places 24 target cells on a 15×15 grid. The three base masks are a filled rectangle, a perimeter ring, and two separated 3×4 rectangles. A random quarter turn, optional reflection, and translation place the mask in the larger lattice. Spacing varies from 24 to 26 pixels, dash length from 15 to 18, width from three to four, and every one of the 225 cell centers receives independent jitter; palette and origin vary as well.

Renderer and forward program. At four-times resolution, target cells are rendered as forward slashes and all other cells as backslashes, then downsampled to 440×440. On the exact cell set, the forward program counts four-neighbor target components and background regions enclosed inside a padded bounding box. It maps $(1, 0)$ to one island, $(1, 1)$ to a ring, and $(2, 0)$ to two islands. Spacing minus dash length defines a pixel-clearance margin; unsupported topology or margin below five pixels is excluded by the ambiguity rule.

Inverse program. The inverse program reads each dash as a forward slash or backslash, places it in a grid, and counts connected regions and holes. It estimates the background color from the median of the border pixels and selects pixels more than 28 RGB units away. Groups of at least 12 pixels connected horizontally or vertically are treated as dashes; at least 100 groups must be detected.
For each dash, the covariance of its pixel coordinates gives its orientation: negative xy covariance indicates a forward slash. The program rejects strokes whose spread or orientation ratio is too small. It groups dash centers into rows and columns using gaps between their coordinates. The number of rows and columns must agree and lie between 9 and 25, with exactly one dash assigned to every cell. This recovers the grid without the sampled spacing or origin.

Decision and failure behavior. The inverse program then counts components and holes in the recovered forward-slash matrix, applying the same decision rule as the forward program. Ambiguous orientations, inseparable axes, duplicate or missing cells, fewer than eight targets, or unsupported topology all raise `ValueError`. Continuous jitter, color, and spacing are not recovered once the lattice is reconstructed.

Edge-Coded Mosaic Reassembly *(declared transform)*

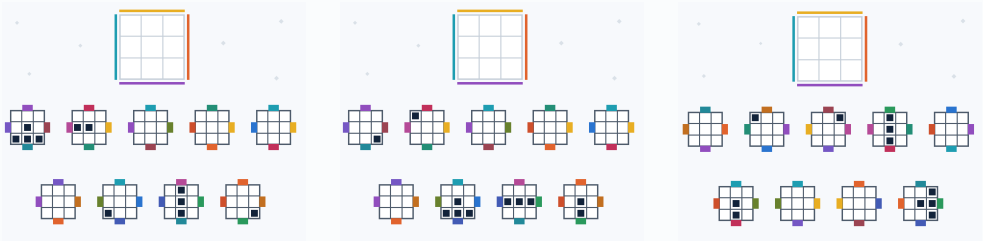


Figure A35: Card permutation, rotation, connector assignment, translation, and arrow vary. Stored outputs from left to right are down, left, and right.

Question. "Rotate and arrange nine cards so outer colors face their named sides and adjacent inner bars match. Which arrow appears after the patches are joined?"

Scene generation. The construction begins with one of four 9×9 binary arrow templates and splits it into nine 3×3 patches. The outer boundary uses fixed side codes: gold at top, orange at right, violet at bottom, and cyan at left. Twelve sampled internal connector colors are shared by each true neighboring edge pair. The nine cards are independently permuted and quarter-turned; a global translation, draw order, and six neutral distractor diamonds affect only presentation. Reflection is never allowed.

Renderer and forward program. The renderer rotates each patch and its four edge codes together and scatters the cards below a neutral destination board in an 840×660 image. The forward program returns the original arrow direction and computes its Hamming distance from the other arrow templates. It never performs the reassembly that the public question asks the viewer to perform.

Inverse program. The inverse program reads the nine cards, then searches for an arrangement that satisfies the edge-color rules. It detects the frame color and groups horizontally or vertically connected pixels. A card frame must contain at least 650 pixels and have width and height between 90 and 102 pixels; exactly nine frames must be found.
For each card, the program reads the nine cell centers and the most frequent color in a 7×7 patch beyond each edge. An edge color must occupy at least 41 of the 49 sampled pixels and cannot be a background, paper, frame, or ink color. The program tries each card in four rotations, placing cards row by row. Outer edges must have the specified boundary colors, inner edges cannot use those colors, and adjacent edges must match. Backtracking stops exploring an arrangement as soon as one of these rules fails. Exactly one complete arrangement must satisfy all the rules.

Decision and abstention. The program joins the nine patches into a 9×9 grid and compares it with the four arrow templates. The best template may differ in at most one cell and must have at least eight fewer mismatches than the next-best match. Missing frames, unclear edge colors or cell values, no unique arrangement, or an insufficient arrow match raise `OracleAbstention`. The forward program reads the stored arrow direction; the inverse program must reconstruct the arrangement to recover it.

One Per Ring *(search)*

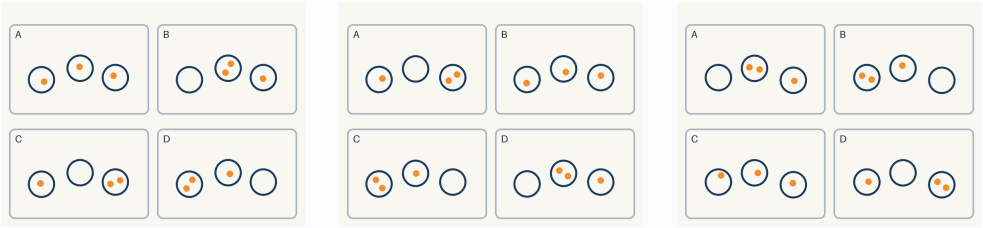


Figure A36: The winning label, ring jitter, and dot placements vary across instances. Stored outputs from left to right are A, B, and C.

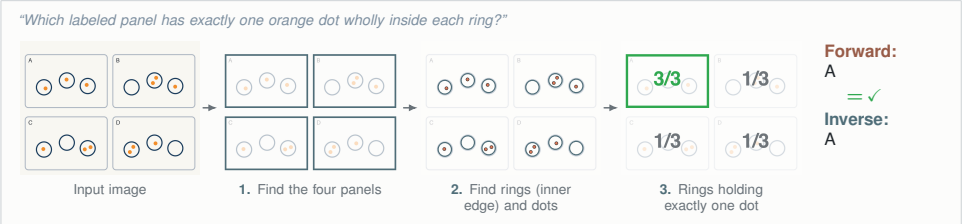


Figure A37: Inverse-program steps on the first stored image above, drawn from the values its own functions return.

Question. "Which labeled panel has exactly one orange dot wholly inside each of its three navy rings? A touching dot does not count."

Scene generation. Four panels share three jittered ring centers with radius 34. The target selected by the random seed receives ring-count vector $[1, 1, 1]$; the distractors receive three of the six permutations of $[2, 1, 0]$. Thus every panel contains exactly three dots, ruling out total-dot count as a shortcut. Dots have radius eight. Count-two pairs are placed oppositely at radial distance 13; count-one dots usually lie one to ten pixels from the ring center, while selected random seeds place one at distance 18 to create a tighter containment margin. Dot order and distractor assignment are presentation details that do not affect the answer.

Renderer and forward program. The forward program assigns a dot to a ring only when the entire dot is strictly inside and it has a unique owner. It scores each panel by the number of rings that contain exactly one dot and returns the unique score-three label. The renderer draws rounded frames, labels, navy ring outlines, and orange dots at three-times resolution and downsamples to 760×560. Nonpositive winner gaps or minimum geometric clearance below four pixels are excluded by the ambiguity rule.

Inverse program. The inverse program detects the rings and dots, tests which dots fit inside each ring, and selects the panel with one dot per ring. It classifies every pixel by nearest distance to five fixed prototypes: background, border, ring, dot, and text. Eight-connected border components must yield exactly four panels with area above 1,200, width above 250, and height above 160; centroid sorting establishes A–D reading order. After cropping five pixels inside each frame, ring components of area at least 450 and dot components of area at least 80 provide centroids and bounding boxes. Radii are estimated from bounding-box dimensions. A dot belongs to a ring only when $d + r_{\text{dot}} < r_{\text{ring}} - 4$. This requires more than four pixels of clearance using the estimated centers and radii. It is the inverse program's pixel-based test of "wholly inside".

Decision and failure behavior. Every panel must contain exactly three recovered rings and three dots, each dot must have exactly one owner, and the best exact-one score must be a unique three. Otherwise the inverse program raises `ValueError`. It recovers component-level membership and reading-order labels, not exact subpixel coordinates, list order, or the forward clearance value.

G Collection Growth and Generation Behavior

This section reports generation, replay, and repair outcomes for Secs. 4.1 and 4.2, spatial-pattern measurements, and repetition and similarity analyses. Full generation counts and accuracy tables appear in App. H.

G.1 Profile-steered generation

We examine growth, verification outcomes, and repair behavior across the 45 model–profile campaigns in Sec. 4.1.

G.1.1 Growth over episode time

Fig. A38 shows when accepted worlds accumulate within each campaign. We divide elapsed episode time by the campaign's total episode time and cumulative accepted count by its final accepted count, then average the normalized curves across the nine profiles for each configuration. The curves show continued additions over campaign time, but normalization removes differences in final accepted-world counts. Absolute throughput appears in Fig. A39.

For an absolute-time comparison, we assign each acceptance to the end of its episode and divide each campaign at three accumulated episode-hours. Rates pool counts and elapsed time across the nine profiles per configuration; the later window ends at each campaign's actual duration. First-window and later-window rates are 7.37 and 7.33 for Sol (high), 4.33 and 5.16 for Sol (max), 3.41 and 4.17 for Luna, 4.30 and 3.83 for Opus 5, and 4.30 and 5.27 for DS-V4-Flash, in accepted worlds per episode-hour. These are descriptive comparisons within the observed six-hour campaigns.

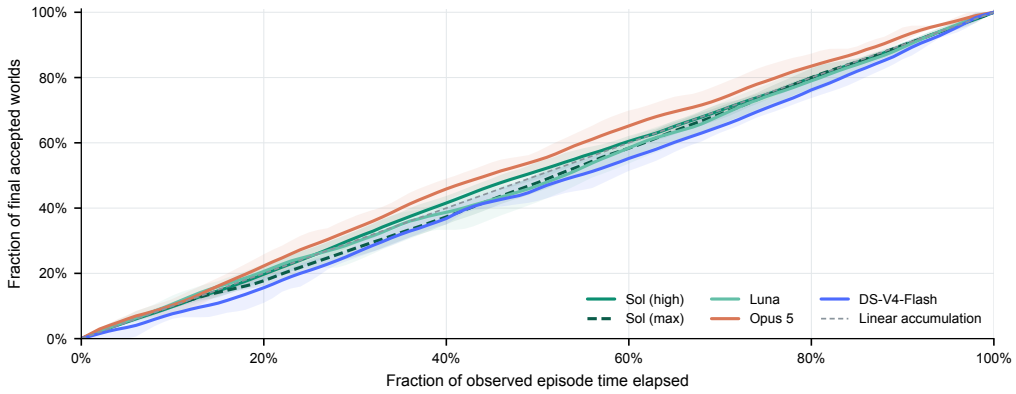


Figure A38: Fraction of final accepted worlds versus fraction of observed episode time in profile-steered generation. Lines average the nine normalized profile campaigns for each configuration; shaded bands span their 20th–80th percentiles.

G.1.2 Submission outcomes and replay

The 45 combinations consumed 269.12 episode-hours and completed 1,426 episodes. Of these, 1,414 produced valid submissions, 1,385 passed the final checks, and 1,331 were accepted. Of the 54 final-verification passes not accepted, 18 failed a retention criterion. The other 36 satisfied the retention criteria but were recorded as budget-incomplete because the agent session ended without the required submission-finalization record. Every configuration produced at least one accepted world in every profile. Fig. A39 shows each model–profile combination separately: every cell reports accepted worlds per episode-hour and gives the accepted count in parentheses. This is not a controlled model comparison because the coding harness, provider, and reasoning setting also differ.

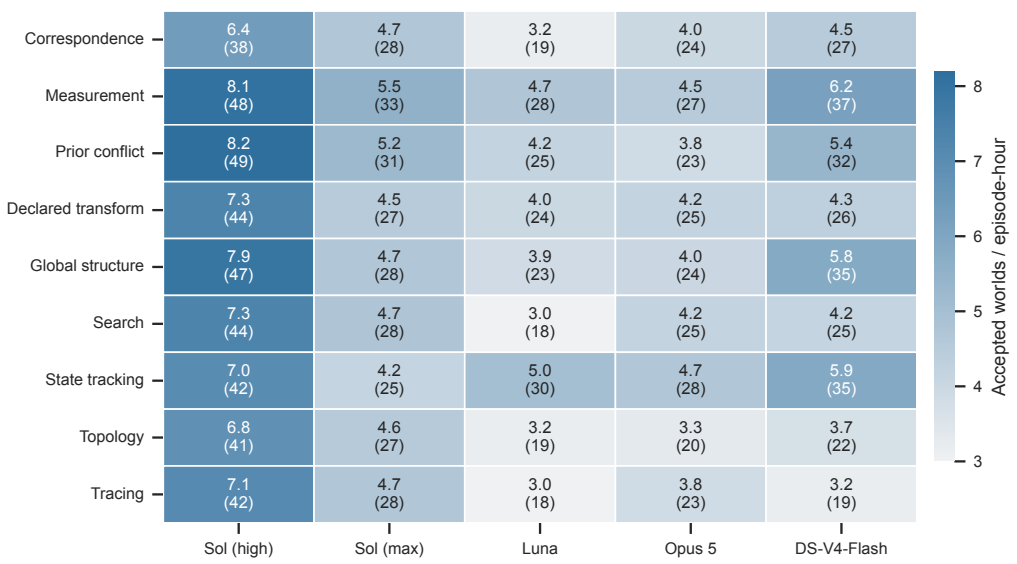


Figure A39: Profile-steered generation yield. Each cell reports worlds accepted during generation per episode-hour; parentheses give the accepted count. Episode time includes provider waiting but excludes final verification and later replay.

Replay outcomes Final verification is final for its generation episode, but acceptance during generation is not the same as passing the larger replay reported here. Of the 1,331 accepted worlds, 1,301 (97.75%) passed the later 200-scene replay using the fixed replay random seed. The 30-world gap comprises 19 recorded replay failures and 11 accepted worlds that are missing from the replay records, with no recorded failure. In 13 of the 19 failures the inverse program was involved: 7 answer disagreements with the forward program and 6 inverse errors. The other six are three sampler failures, two scene-count errors, and one input/output error. Fig. A40 reports both stages rather than treating every world accepted during generation as replay verified.

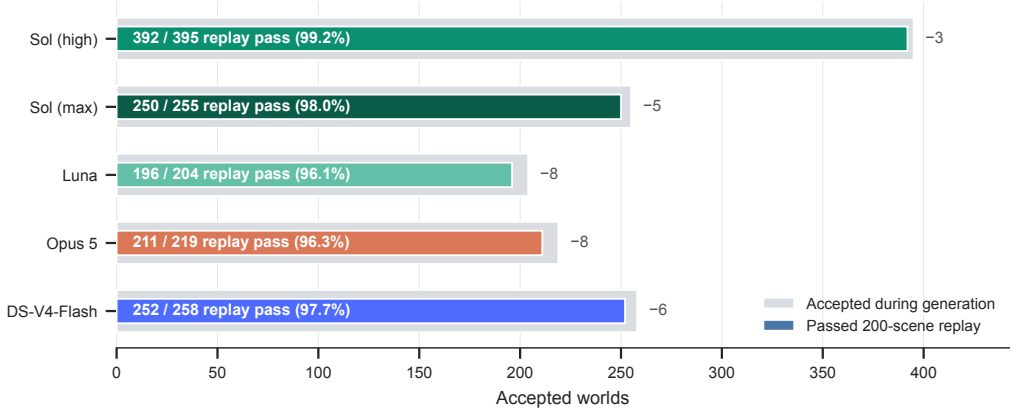


Figure A40: Generation-time acceptance and later 200-scene replay verification. Gray bars show accepted worlds and colored overlays show replay-verified worlds; right-hand labels report the corresponding gaps.

G.1.3 Self-check repairs and final-check failures

Self-check repairs High acceptance does not mean that every world passed on the first try. The run logs identify 681 accepted worlds that first failed the self-checks and were then repaired within the same episode. Fig. A41 separates first-pass acceptance from acceptance after repair for each configuration. Repairs after inverse-program failures, pooled over both experiments, are classified by the files they changed in Tab. A7.

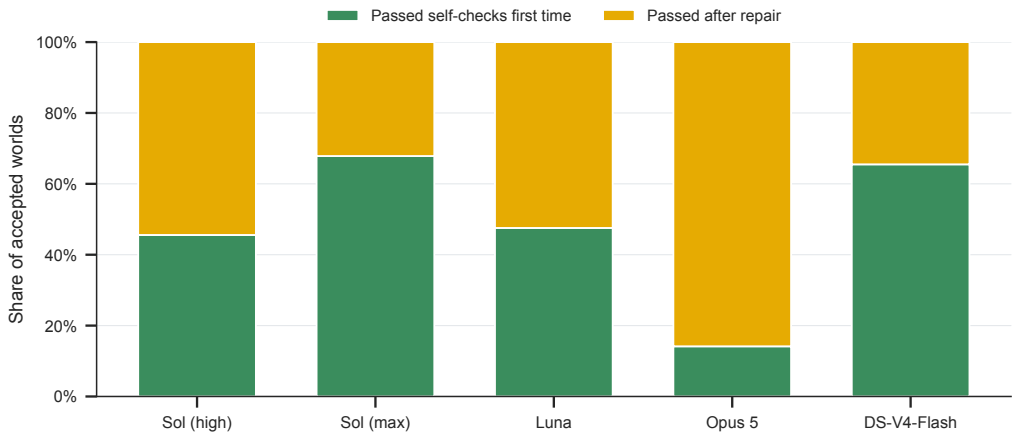


Figure A41: Repair paths among worlds accepted during generation. Green denotes acceptance on the first completed self-check pass; yellow denotes acceptance after a self-check failure and repair.

Failures found by final checks Across 1,414 episodes that reached final verification, the verifier recorded 74 failed checks. The most common concerned the required interface, stored evidence, execution, scene changes that should leave the image unchanged, and coverage of at least two answer classes. Fig. A42 reports failures per 100 checked episodes and prints the raw count. The 74 failures occurred across 29 episodes; one world may fail several checks, so the cells cannot be summed as rejected worlds. The submitting agent was not resumed and could not repair the world. The agent in the next episode could see the rejected submission and its report, but not the protected comparison records. What each agent sees is described in Sec. D.1.

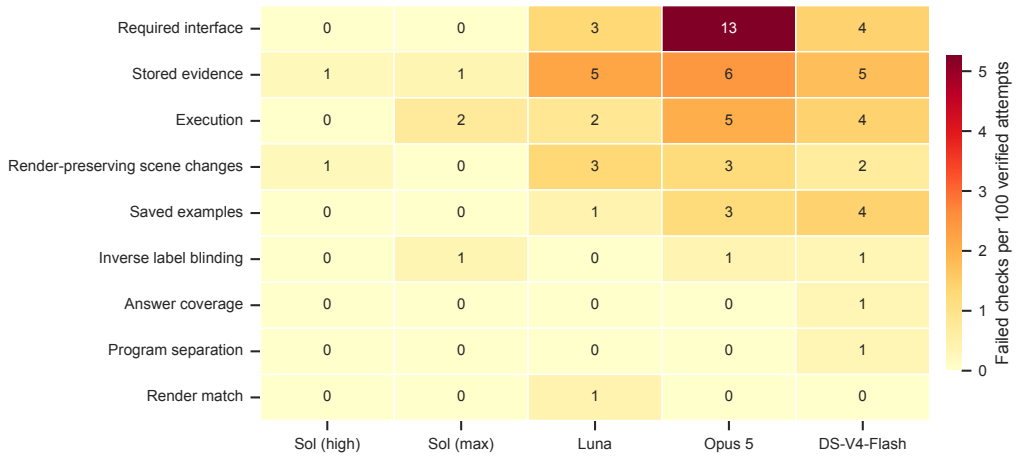


Figure A42: Failures found by final checks. Color reports failed checks per 100 checked episodes and cell text gives the raw failure count. The 74 failed checks occurred across 29 episodes and may co-occur within a world. The submitting agent was not resumed after these checks.

G.2 Spatially-steered generation

This experiment asks the agent to generate questions with a requested spatial pattern of the answer's evidence. Verification determines whether a world is accepted. A separate masking analysis measures its spatial pattern. The controller keeps a smaller category record to guide later requests. We report generation and replay outcomes, agreement with requests, and category coverage separately.

G.2.1 Pattern measurement and request selection

Before each episode, a deterministic rule selects one of the nine spatial patterns from the controller-maintained category record. The score favors categories represented by fewer stored worlds, cate-

gories with more previous target matches or fewer observations, and categories not requested in the preceding episode; it penalizes repeated misses. The record holds at most three worlds per category. The coding agent sees the selected request but does not assign the measured category.

After submission, a separate perturbation analysis applies the fixed rules in Tab. A9. An error caused by a deliberately corrupted image is recorded as perturbation evidence; an inverse-program error on an unmodified verification image still fails final verification.

Table A9: Fixed spatial-pattern measurement and category-record rules.

Component	Fixed rule
Fresh scenes	Four to six distinct scenes per submitted world.
Probe lattice	8×8 positions; each region extends one quarter-cell beyond its nominal boundary on every side.
Replacement	Median color of the image's outer three-pixel border.
Affected position	The inverse answer changes from the unmodified image, or the inverse program raises an error on the corrupted image.
Scale	Maximum normalized span below 0.30 is focal; 0.30 to below 0.65 is regional; otherwise distributed.
Shape	At least two four-connected components is multipart. Otherwise, elongation at least 2.75 or fill at most 0.55 with nonzero span is pathlike; all remaining cases are compact.
Complete measurement	At least two thirds of scenes expose an affected region.
Stable category	At least two thirds of all analyzed scenes receive the aggregate category.
Category record	At most three worlds per category. A fixed weighted similarity over the declared mechanism fields and summary text must be below 0.68.
Behavior separation	A normalized 149-component vector must have root-mean-square distance at least 0.06 from every stored world and 0.10 from stored worlds in the same category.
Full category	Keep the three-world subset with the largest minimum pairwise behavior distance.

For each image, the analysis measures the positions affected by masking and assigns a scale and shape. It assigns the world's category by applying the same rules to the median geometry measurements across sampled images. A measurement is complete if masking affects at least two thirds of the images. A category is stable if at least two thirds of all images receive the world-level category; images with no affected position do not count as matches. Thus, masking can affect enough images for a complete measurement without producing a stable category.

To avoid storing worlds with very similar masking responses, the controller compares a 149-component vector for each world. It contains 64 centered answer-change rates, 64 centered error rates, and 21 summaries of pattern geometry, category stability, and image structure. The distance thresholds in Tab. A9 govern this category record. Pattern measurement is separate from acceptance: a category mismatch does not reject an otherwise accepted world, and a rejected submission may still have a complete measurement. All thresholds were fixed before the experiment.

G.2.2 Generation and replay evidence

The five configurations completed all nine profiles, producing 1,179 episodes and 903 generation-time acceptances. Replay regenerated each accepted world on 200 fresh scenes using random seed 42. It verified 875 worlds. Of the 28-world gap, 17 have recorded replay failures and 11 Luna worlds are missing from the replay records, with no recorded failure. In 15 of the 17 failures the inverse program was involved: 4 answer disagreements and 11 inverse errors or abstentions. The other two are a sampler failure and a failed ablation check. These are separate from the 11 missing worlds in the profile-steered experiment; no shared cause is established. We report omissions separately rather than counting them as failed worlds (Fig. A43).

DS-V4-Flash accepted only 25 worlds. 142 of its episodes reached the 20-minute limit without an accepted world. Of these, 63 produced no valid submission, 62 submitted a world that failed the final checks, and 17 passed the final checks and retention criteria but ended without the required submission-finalization record. The logs separate these outcomes but do not isolate whether provider

latency, harness overhead, or agent iteration caused the timeout. Median episode duration rises from 625.23 seconds in profile-steered generation to 1,204.16 seconds under spatial steering, while the median provider-turn count falls from 46 to 32.5.

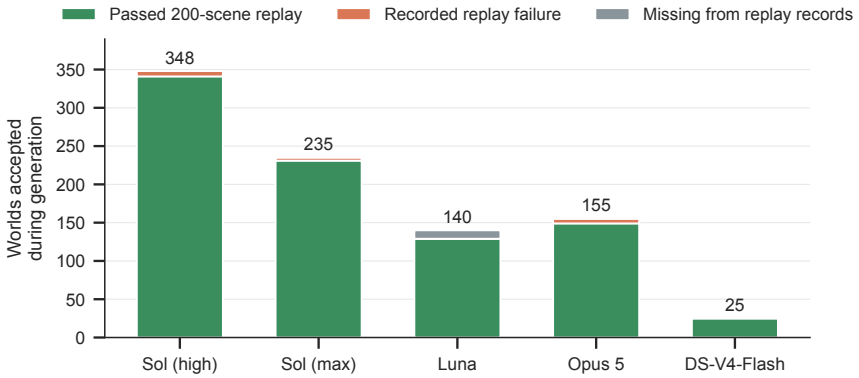


Figure A43: Reconciliation of generation-time accepted worlds with later 200-scene replay. Recorded replay failures and worlds missing from the replay records are reported separately from successful replay verification.

G.2.3 Requested and observed spatial patterns

We report request matching for two populations: replay-verified worlds and all episodes with a complete pattern measurement (Tab. A10). In each case, the rate is the number matching the request divided by the number with complete measurements. Measurements were made after submission, not repeated on the 200 replay scenes. Replay determines inclusion in the first population.

Of 875 replay-verified worlds, 874 have complete pattern measurements and 835 match the requested category (95.5%). The remaining 39 measured worlds have a different category. Rates range from 79.2% for Opus 5 to 99.6% for Sol (max). The one incomplete measurement belongs to a DS-V4-Flash topology world: masking exposed an affected region in only three of six scenes, below the required two-thirds threshold. Its stored category is shown in the grid of all replay-verified worlds, but this world is excluded from the match rate and the coverage comparison requiring complete measurements. A missing or incomplete measurement is not counted as a category mismatch.

Table A10: Requested-category fidelity by generation outcome and replay status. Measured denotes a complete pattern measurement; Missing includes absent and incomplete measurements. Match rates divide Matches by Measured. Accepted and Not accepted partition All episodes; Replay-verified is a subset of Accepted.

Population	Total	Measured	Missing	Matches	Match (%)
All episodes	1,179	981	198	918	93.6
Accepted	903	902	1	860	95.3
Not accepted	276	79	197	58	73.4
Replay-verified	875	874	1	835	95.5
Replay-verified worlds by coding-agent configuration					
Sol (high)	341	341	0	336	98.5
Sol (max)	231	231	0	230	99.6
Luna	129	129	0	128	99.2
Opus 5	149	149	0	118	79.2
DS-V4-Flash	25	24	1	23	95.8

Across all episodes with complete pattern measurements, 918 of 981 match the request (93.6%). This includes 860 of 902 accepted worlds (95.3%) and 58 of 79 submissions that were not accepted (73.4%).

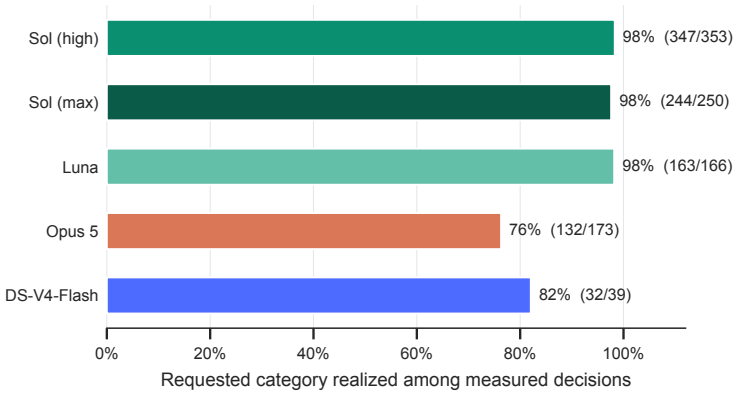


Figure A44: Agreement between requested and measured categories by configuration, over all episodes with a complete post-submission pattern measurement, including submissions that were not accepted. The population is not restricted to replay-verified worlds.

The other 198 episodes lack a complete measurement: 64 have no recorded measurement status, 130 are marked missing, and four are incomplete. Only one produced an accepted world, the topology example described above.

Of the 276 episodes without acceptance, 272 ended at the session limit without an accepted world (no valid submission, a failed final check, or no finalization record), and four had other outcomes. Complete measurements were available for 78 of these 272 episodes and one submission that passed verification but failed retention. A pattern measurement can therefore be available even when generation does not produce an accepted world.

Of 1,179 episodes, 835 produced an accepted world that passed replay and had a complete pattern measurement matching the request (70.8%). Across 269.21 episode-hours, this is 3.10 worlds per episode-hour, including time spent on unsuccessful episodes. The 95.5% match rate above instead uses only replay-verified worlds with complete measurements.

The remaining analyses use all 981 measured episodes, not the replay-verified subset. Fig. A44 reports match rates and the number of measured episodes for each configuration: the three OpenAI configurations exceed 97%, whereas Opus 5 and DS-V4-Flash reach 76.3% and 82.1%.

For a descriptive comparison, we permute the requested category among all 981 measured episodes within each model–profile campaign. Across 10,000 permutations, the mean match rate is 14.2%, the central 95% interval is $[12.0, 16.4]\%$, and no draw matches or exceeds the observed 93.6% (Monte Carlo $p < 10^{-4}$ with random seed 20260902). The within-campaign restriction preserves configuration- and profile-specific category frequencies. Because the controller adapts later requests to its campaign record, this permutation is a reference distribution rather than a randomized treatment test.

Fig. A45 shows where mismatches occur. The final column contains episodes for which no complete category was measured; these episodes are excluded from the reported match rate. The concentration along the diagonal shows agreement between requested and measured categories. The figure includes episodes regardless of their verification or replay outcome.

G.2.4 Category coverage and comparison with unsteered generation

We compare spatial patterns within each coding-agent configuration and profile. The comparison starts from all replay-verified worlds and applies the pattern-measurement criteria below. It does not use the smaller controller record, which stores at most three worlds per category.

Together, the 875 replay-verified worlds cover every category (Fig. A46). Counts range from 78 focal–pathlike worlds to 121 focal–compact worlds. This does not mean that every model–profile combination contains every category.

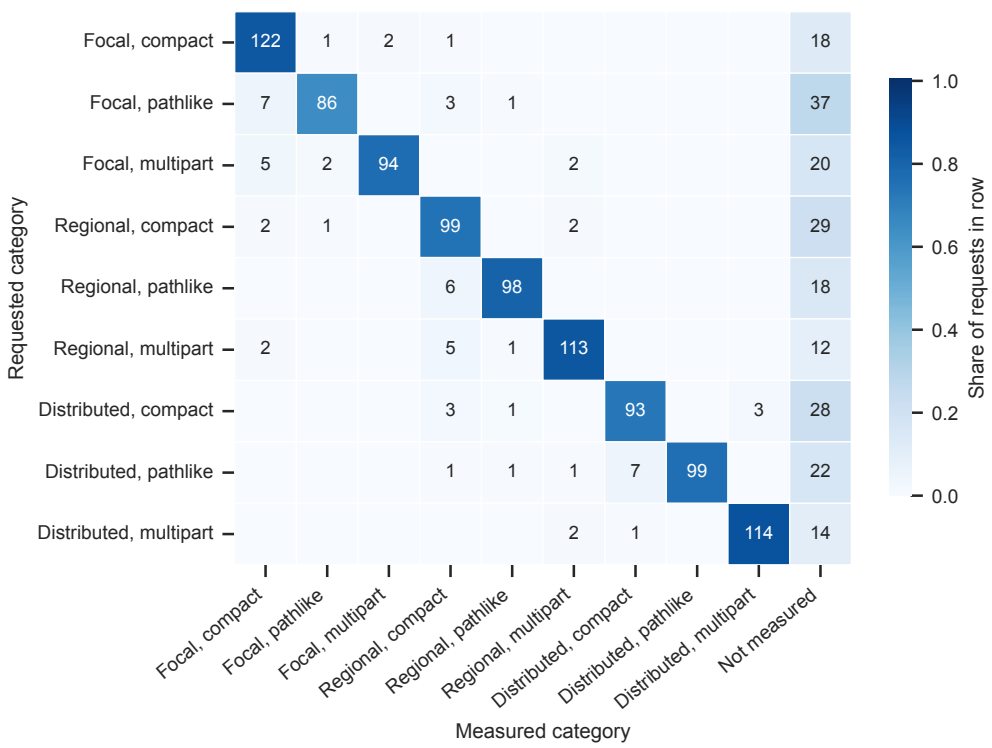


Figure A45: Requested versus measured spatial pattern. Rows are requested categories and columns are measurement outcomes. Each category names its scale, then its shape. Numbers count all generation episodes, including submissions not accepted, and color is normalized within each requested category.

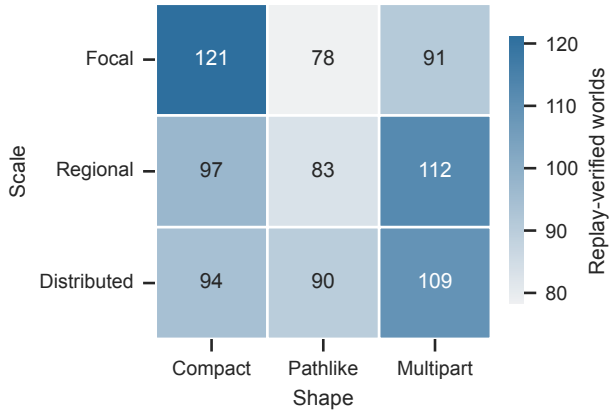


Figure A46: Spatial patterns assigned to all 875 replay-verified worlds, including one with an incomplete measurement that is excluded from the comparison below. Each cell crosses scale with shape.

Comparison within each configuration and profile The two experiments use the same five coding-agent configurations and nine profiles, giving 45 model–profile combinations with six-hour budgets. The spatially-steered experiment measured the spatial pattern during generation. The profile-steered experiment ran earlier, so we apply the same fixed measurement to its saved replay images. The main analysis requires a completed pattern measurement and a valid category. Completion requires an affected region in at least two thirds of tested scenes; it does not require the same category in two thirds of scenes. These requirements are met for 73.9% of profile-steered replay worlds and 99.9% of spatially-steered replay worlds.

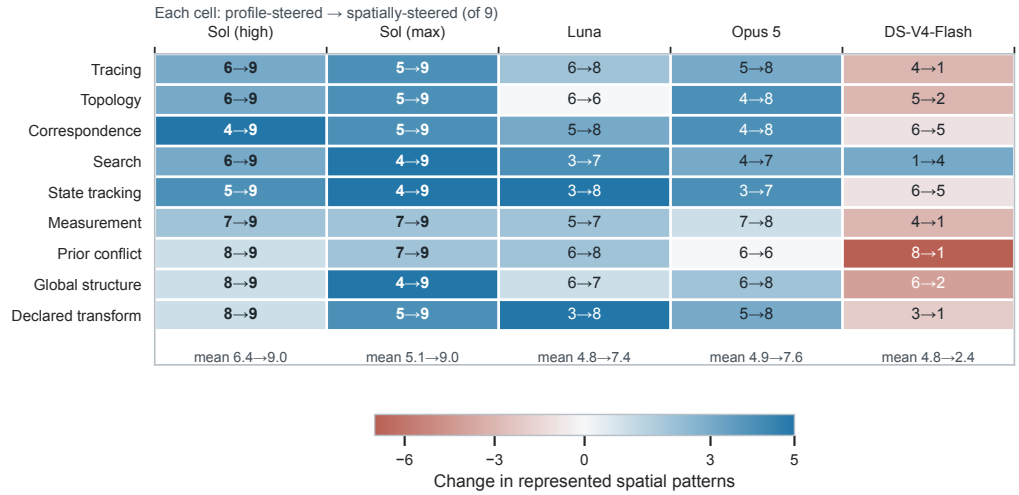


Figure A47: Spatial patterns represented in every matched model–profile combination. Each cell gives profile-steered → spatially-steered categories (of nine), while color encodes the change. Bold endpoints indicate that all nine categories are represented. The means summarize the nine profile rows rather than pooling worlds. The DS-V4-Flash decrease coincides with its spatially-steered campaign, in which 142 episodes reached the session limit without an accepted world

.

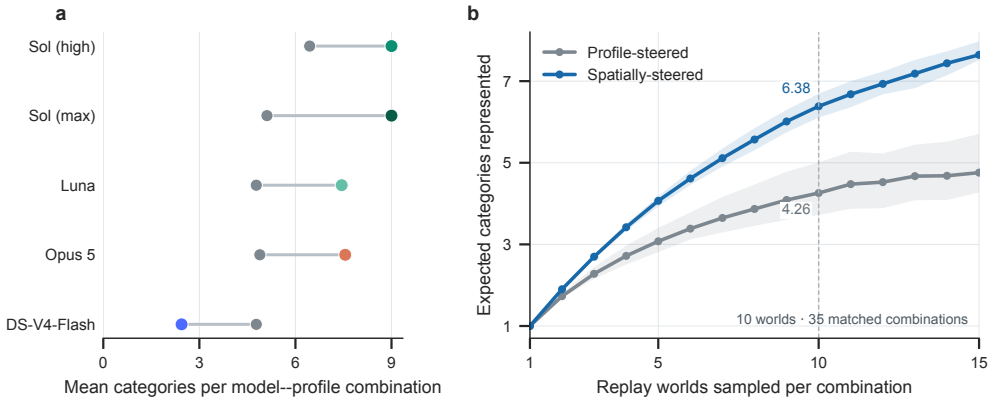
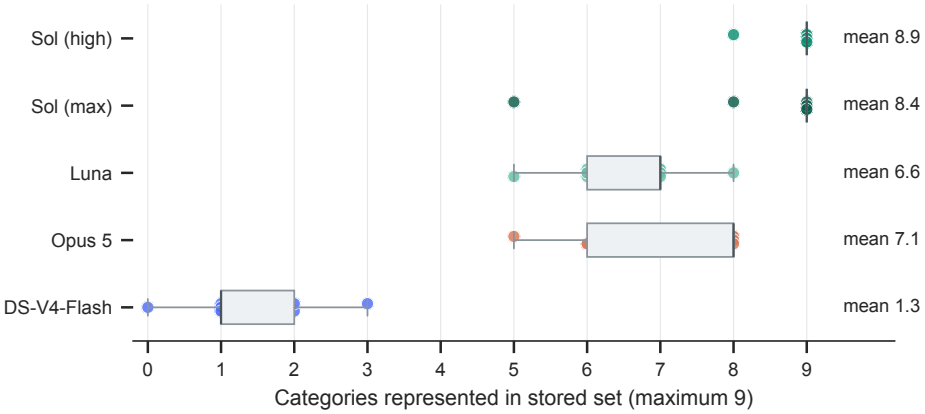


Figure A48: Matched comparison of represented spatial patterns. **(a)** Mean categories across the nine profiles for each coding-agent configuration. Gray points denote profile-steered generation and colored points denote spatially-steered generation. **(b)** Expected categories after sampling equal numbers of replay worlds from each eligible matched combination; bands show the interquartile range across combinations.

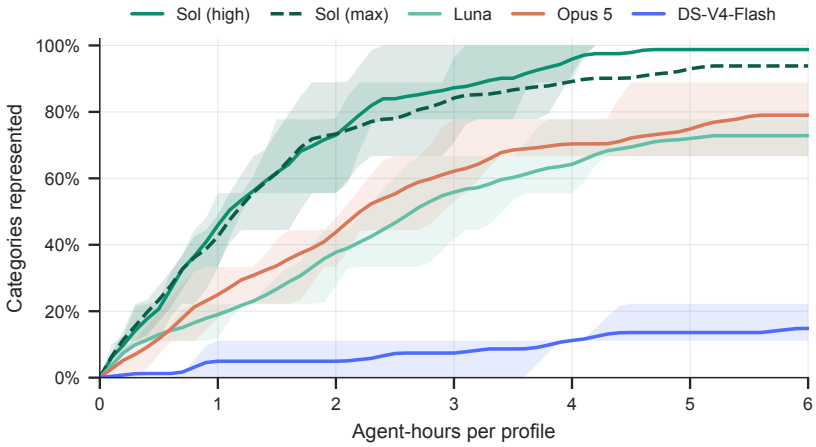
Under the main rule, the mean number of represented categories changes from 5.20 to 7.09 of nine. The changes in Fig. A47 are positive in 35 model–profile combinations, zero in two, and negative in eight. The negative results are concentrated in DS-V4-Flash, whose accepted yield also falls from 258 to 25 worlds across the two experiments. The overall average therefore does not describe every configuration.

Different numbers of accepted worlds could explain part of the difference. We therefore compute the expected number of categories after sampling the same number of replay worlds without replacement from each model–profile combination. At ten worlds, 35 matched combinations have enough data: the expected number is 4.26 categories for profile-steered generation and 6.38 for spatially-steered generation, a paired difference of 2.12 categories. The full curve appears in Fig. A48b. Because the experiments were not randomized or executed concurrently, both the full-set and equal-count comparisons remain descriptive.

Controller category records The controller-maintained category record reaches all nine categories in 15 of 45 model–profile combinations, but the result varies sharply by configuration. The mean number of represented categories is 8.9 for Sol (high), 8.4 for Sol (max), 6.6 for Luna, 7.1 for Opus 5, and 1.3 for DS-V4-Flash. The corresponding trajectories in Fig. A49 show that the two Sol configurations cover most categories early, whereas DS-V4-Flash covers few.



(a) Final controller-maintained category record.



(b) Categories represented over episode time.

Figure A49: Spatial patterns represented over time. Points or bands summarize the nine profile runs for each configuration. The controller's category record is smaller than the full replay-verified set used in the matched comparison.

G.3 Repetition and cross-world similarity

We check whether a world generates the same image repeatedly and whether different worlds share images or question texts. We also retrieve similar worlds for inspection. Similarity scores identify possible duplicates, but confirming semantic equivalence requires reviewing the questions and images.

G.3.1 Profile-steered generation

Within-world repetition Replay also shows how often a generator produces the exact same image. Across all 260,200 decoded RGB scenes, 9,577 occurrences (3.68%) repeat an earlier pixel-identical scene within the same world. The repetitions are concentrated: 167 of 1,301 worlds contain any repeat, while the median world for every configuration contains 200 distinct scenes. Fig. A50 reports both the repeated-scene rate and the fraction of affected worlds. Most repetition therefore comes from a small subset of generators.

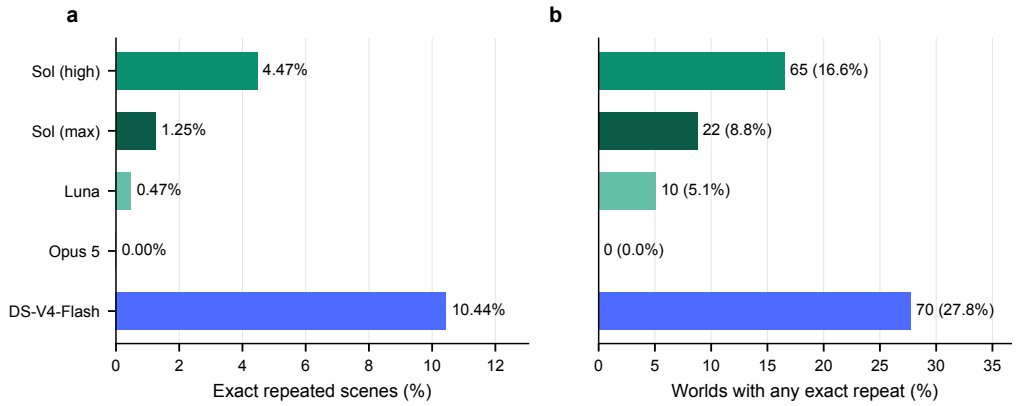


Figure A50: Exact decoded-pixel repetition in the 200-scene replay set. Panel (a) counts repeated scene occurrences beyond the first identical RGB render; panel (b) reports worlds containing at least one such repeat.

Cross-world reuse and retrieval Exact reuse across worlds is much narrower: 102 shared exact images come from only two world pairs, and no normalized question text is reused exactly. We also compare worlds using fixed DINOv2 (Oquab et al., 2024) image representations and MiniLM (Wang et al., 2020b) text representations. Among all 845,650 unordered pairs, 16 exceed 0.90 cosine similarity for both image and question, and two exceed 0.95. Fig. A51 shows the nearest-neighbor similarity distributions for images, question texts, and their combined scores.

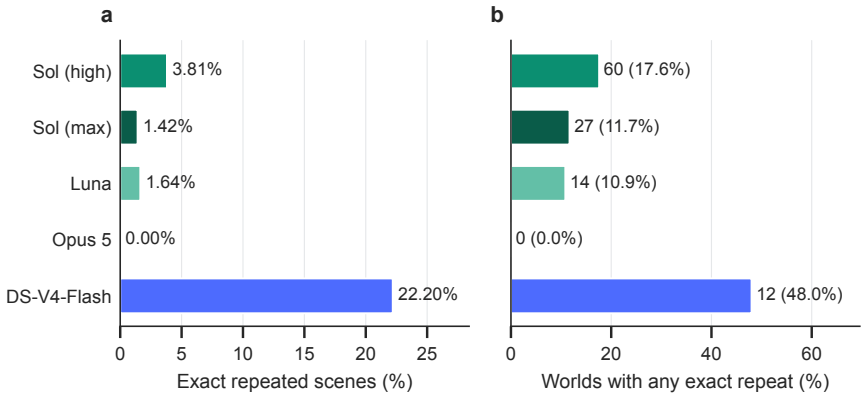


Figure A52: Exact decoded-pixel repetition within executable worlds. Panel (a) counts repeated scenes. Panel (b) reports worlds containing at least one repeat.

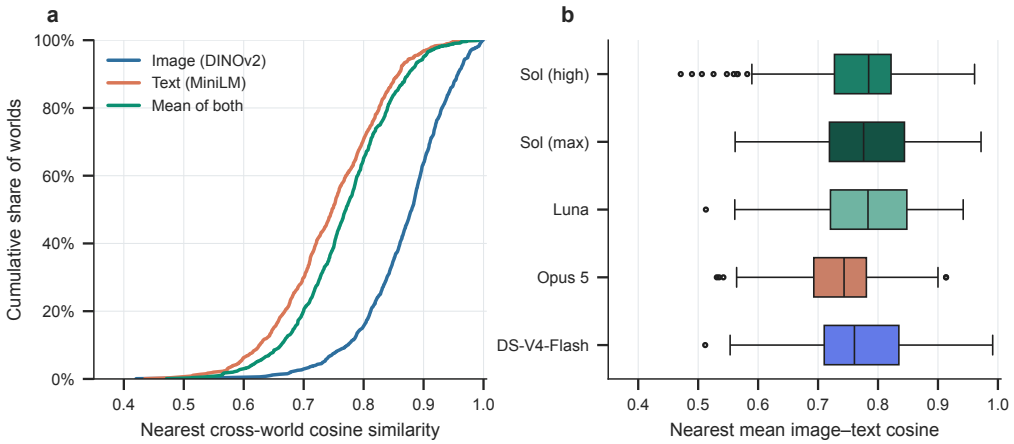


Figure A51: Cross-world nearest-neighbor retrieval. Panel (a) compares each world's nearest image, text, and mean image–text cosine similarity; panel (b) stratifies the nearest mean score by configuration. The scores are used only to rank pairs for human review and are not probabilities that two worlds are duplicates.

G.3.2 Spatially-steered generation

The replay check covers 175,000 scenes from 875 executable worlds. Within a world, 4,786 scenes repeat another scene's decoded RGB pixels, or 2.73% of all scenes. Repetition is uneven: it is 22.2% for DS-V4-Flash and zero for Opus 5 (Fig. A52). No decoded-pixel signature and no normalized question text is shared across two different worlds.

We also compare worlds using fixed DINOv2 image and MiniLM text representations. Among all 382,375 cross-world pairs, no pair exceeds 0.95 cosine similarity for both image and question, and one exceeds 0.90 for both. These thresholds only select pairs for review and do not decide whether two worlds are duplicates (Fig. A53).

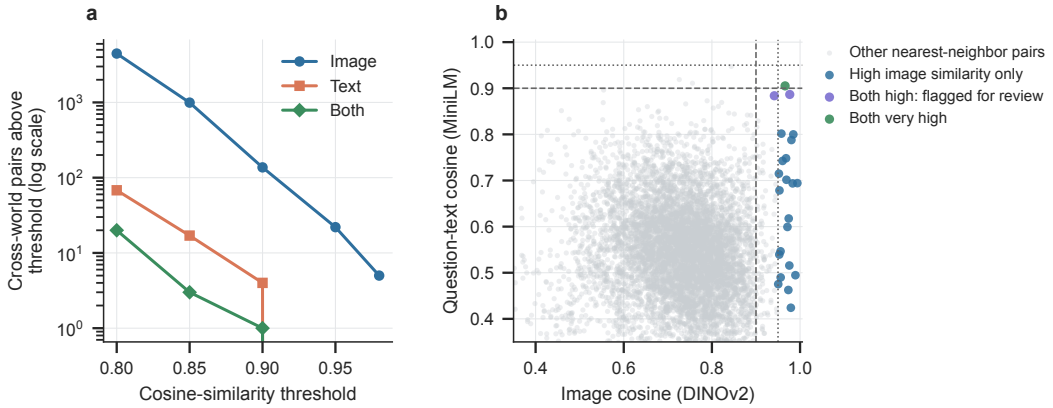


Figure A53: Cross-world image and question-text comparison. The similarity scores only identify pairs for review and are not probabilities that two worlds are duplicates.

H GENERATION COUNTS AND MODEL ACCURACY

We report generation counts, describe the evaluation sets and scoring rules, then report results by discovery profile and generating configuration. Finally, we identify the worlds with complete frontier evaluations eligible for the feedback example sets used in model-feedback-steered generation.

H.1 GENERATION SUMMARY ACROSS BOTH EXPERIMENTS

Tab. A11 gives counts and rates for both generation experiments; spatially-steered results are expanded in Sec. G.2.

Table A11: Episodes, accepted worlds, and replay results by coding-agent configuration in profile-steered and spatially-steered generation. Accepted worlds/hour uses episode-hours. Replay counts worlds that passed the later 200-scene replay.

	Profile-steered				Spatially-steered			
Configuration	Episodes	Accepted	Replay	Accepted/h	Episodes	Accepted	Replay	Accepted/h
Sol (high)	396	395	392	7.35	355	348	341	6.47
Sol (max)	263	255	250	4.75	254	235	231	4.36
Luna	230	204	196	3.79	199	140	129	2.60
Opus 5	248	219	211	4.06	203	155	149	2.88
DS-V4-Flash	289	258	252	4.78	168	25	25	0.46
Overall	1,426	1,331	1,301	4.95	1,179	903	875	3.35

H.2 RESULTS BY DISCOVERY PROFILE

Tab. A12 breaks down the evaluation in Tab. 1 by discovery profile. Declared transforms have the lowest accuracy for each frontier evaluator in both evaluation sets. Among this profile's instances scored by all three frontier evaluators, at least two are wrong on 18.4% in profile-steered and 18.8% in spatially-steered generation.

Table A12: Evaluator accuracy by discovery profile. Worlds counts evaluated worlds, with five instances each; model columns report accuracy (%) over scored responses.

Profile-steered

Profile	Worlds	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Tracing	125	97.4	95.0	97.3	54.4	60.3	51.2
Topology	129	98.1	95.8	97.7	63.4	64.5	58.6
Correspondence	133	99.1	96.7	98.6	49.3	51.0	52.0
Search	138	98.4	93.9	96.2	53.5	60.6	54.5
State tracking	156	98.2	97.9	99.7	39.5	37.7	33.3
Measurement	169	99.2	99.4	98.6	67.5	71.6	63.4
Prior conflict	157	97.1	96.2	98.2	78.3	85.0	80.6
Global structure	148	96.2	93.6	95.1	63.6	71.1	62.6
Declared transform	143	81.2	73.8	80.2	40.0	44.5	39.4
Overall	1,298	96.1	93.7	95.9	56.9	61.0	55.4

Spatially-steered

Profile	Worlds	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Tracing	94	97.4	93.6	98.6	48.5	54.3	55.3
Topology	91	95.8	96.3	99.3	60.9	68.6	62.2
Correspondence	103	93.8	85.4	95.0	39.4	41.4	39.8
Search	89	95.5	84.7	92.1	51.9	54.8	52.8
State tracking	114	91.2	84.7	93.8	32.6	32.8	34.0
Measurement	109	97.8	94.5	95.3	59.6	64.0	59.4
Prior conflict	89	94.4	90.5	95.1	65.8	70.6	71.5
Global structure	82	94.6	86.3	91.2	55.6	63.2	60.2
Declared transform	103	83.5	64.5	80.5	34.2	31.5	32.4
Overall	874	93.7	86.5	93.6	49.1	52.5	51.1

H.3 RESULTS BY GENERATING CONFIGURATION

Tab. A13 separates the same evaluation by the coding-agent configuration that generated each world. It is a descriptive breakdown of complete configurations, not an estimate of a model-only effect. Tabs. A14 and A15 split profile-steered generation further, by profile and generating configuration, and Tabs. A16 and A17 do the same for spatially-steered generation.

Table A13: Accuracy by generating configuration and evaluator. Values are percentages over scored responses. The last column reports instances on which at least two frontier evaluators are incorrect, using only instances with scored responses from all three. Worlds is the number in the corresponding evaluation export.

Configuration	Worlds	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B	≥ 2 incorrect
Profile-steered								
Sol (high)	392	96.4	93.2	95.8	55.6	60.1	53.7	3.1
Sol (max)	250	96.7	93.8	97.0	48.3	53.2	48.7	2.3
Luna	196	95.3	94.5	95.9	65.4	68.3	62.8	3.7
Opus 5	211	96.0	92.8	94.4	43.6	45.3	40.4	3.6
DS-V4-Flash	249	95.9	94.7	96.3	72.4	78.0	71.6	3.4
Spatially-steered								
Sol (high)	341	94.1	85.6	94.3	51.1	54.8	54.5	5.6
Sol (max)	231	92.9	87.9	94.2	50.8	53.3	49.8	5.1
Luna	129	93.0	86.7	91.2	56.0	60.0	58.1	6.0
Opus 5	149	94.0	84.7	92.6	31.5	35.2	34.4	5.4
DS-V4-Flash	24	95.8	96.7	98.3	77.5	80.0	80.8	1.7

H.4 Generation and accuracy for every profile–configuration combination

The following tables report every combination of the nine profiles and five generating configurations in each experiment. Episodes counts generation attempts; Accepted counts worlds retained after verification. Evaluated counts worlds in the evaluation export, each with five instances. Accuracy is the percentage correct among scored responses for each evaluator; missing responses are excluded. A dash indicates no scored responses. The archived data retain correct and scored counts for every cell.

Table A14: Profile-steered generation: episodes, accepted worlds and evaluator accuracy by profile and generating configuration. Episodes, Accepted, and Evaluated are counts; the six model columns are accuracies (%) over scored responses.

Coding agent	Episodes	Accepted	Evaluated	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Tracing									
Sol (high)	42	42	42	97.6	95.7	98.0	57.6	63.3	50.0
Sol (max)	29	28	28	95.7	93.6	94.5	35.7	42.9	40.7
Luna	21	18	16	98.8	100.0	98.7	72.5	76.2	63.8
Opus 5	27	23	22	98.2	91.8	97.9	51.8	48.2	45.5
DS-V4-Flash	26	19	17	97.6	95.3	97.6	63.5	82.4	67.1
Topology									
Sol (high)	41	41	41	97.1	94.6	95.2	54.6	57.6	55.6
Sol (max)	28	27	27	98.5	98.5	100.0	59.3	53.3	51.1
Luna	23	19	19	98.9	94.7	96.7	71.6	74.7	66.3
Opus 5	25	20	20	97.0	91.0	97.7	52.0	52.0	40.0
DS-V4-Flash	25	22	22	100.0	100.0	100.0	88.2	93.6	83.6
Correspondence									
Sol (high)	38	38	38	98.9	97.4	100.0	46.8	47.4	46.3
Sol (max)	29	28	28	97.9	94.2	98.4	39.3	45.0	41.4
Luna	24	19	19	100.0	95.8	97.8	66.3	64.2	75.8
Opus 5	28	24	22	99.1	100.0	99.0	35.5	41.8	33.6
DS-V4-Flash	29	27	26	100.0	96.2	96.8	63.1	60.8	70.0
Search									
Sol (high)	45	44	44	98.2	93.2	95.3	46.4	59.1	51.8
Sol (max)	29	28	27	100.0	96.3	100.0	48.9	56.3	52.6
Luna	24	18	18	100.0	95.6	97.6	64.4	66.7	54.4
Opus 5	28	25	25	94.4	87.9	89.7	38.4	42.4	40.8
DS-V4-Flash	34	25	24	100.0	97.5	99.1	79.2	82.5	75.8
State tracking									
Sol (high)	42	42	42	96.7	98.1	100.0	41.9	38.6	32.9
Sol (max)	27	25	24	98.3	94.9	100.0	30.8	27.5	25.8
Luna	31	30	30	100.0	100.0	100.0	36.7	30.0	31.3
Opus 5	29	28	27	98.5	97.8	99.2	25.2	23.7	20.0
DS-V4-Flash	35	35	33	98.2	98.2	99.4	57.0	62.4	52.1

Table A15: Profile-steered generation: episodes, accepted worlds and evaluator accuracy by profile and generating configuration (continued). Episodes, Accepted, and Evaluated are counts; the six model columns are accuracies (%) over scored responses.

Coding agent	Episodes	Accepted	Evaluated	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Measurement									
Sol (high)	48	48	47	99.1	99.1	100.0	70.2	71.5	63.8
Sol (max)	33	33	33	100.0	99.4	98.0	55.2	65.5	49.1
Luna	29	28	26	98.5	100.0	96.9	79.2	86.9	76.2
Opus 5	28	27	26	99.2	100.0	97.3	45.4	43.1	36.9
DS-V4-Flash	38	37	37	98.9	98.9	99.5	82.2	86.5	85.4
Prior conflict									
Sol (high)	49	49	48	95.8	95.0	98.3	82.9	86.2	83.3
Sol (max)	31	31	30	98.7	99.3	100.0	82.0	89.3	84.0
Luna	27	25	24	94.2	95.8	95.0	81.7	85.8	82.5
Opus 5	26	23	23	100.0	96.5	98.2	73.0	78.3	73.0
DS-V4-Flash	33	32	32	97.5	94.9	98.8	69.4	83.1	77.5
Global structure									
Sol (high)	47	47	46	95.2	93.0	91.6	61.7	73.0	61.7
Sol (max)	29	28	26	93.8	91.5	97.4	45.4	59.2	56.2
Luna	24	23	20	98.0	93.0	99.0	64.0	70.0	60.0
Opus 5	28	24	22	97.3	95.5	95.0	52.7	55.5	52.7
DS-V4-Flash	36	35	34	97.6	95.3	95.8	87.1	88.2	76.5
Declared transform									
Sol (high)	44	44	44	89.1	72.7	83.1	32.7	37.3	31.8
Sol (max)	28	27	27	86.7	74.6	83.6	31.9	31.1	31.9
Luna	27	24	24	72.3	75.8	81.8	61.7	70.8	62.5
Opus 5	29	25	24	80.8	74.2	75.0	24.2	29.2	25.8
DS-V4-Flash	33	26	24	70.0	72.5	74.5	56.7	61.7	52.5

Table A16: Spatially-steered generation: episodes, accepted worlds and evaluator accuracy by profile and generating configuration. Episodes, Accepted, and Evaluated are counts; the six model columns are accuracies (%) over scored responses.

Coding agent	Episodes	Accepted	Evaluated	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Tracing									
Sol (high)	36	36	36	98.3	95.6	98.2	52.8	55.6	61.1
Sol (max)	28	27	26	96.2	94.6	99.2	50.8	50.0	54.6
Luna	22	15	14	95.7	94.3	97.1	50.0	71.4	58.6
Opus 5	22	17	17	98.8	86.7	100.0	31.8	41.2	38.8
DS-V4-Flash	19	1	1	100.0	100.0	100.0	100.0	100.0	100.0
Topology									
Sol (high)	36	35	35	97.1	98.3	98.8	57.1	68.6	61.7
Sol (max)	27	25	25	93.6	96.0	99.2	63.2	66.4	65.6
Luna	22	14	14	97.1	98.6	100.0	75.7	87.1	72.9
Opus 5	21	14	14	94.3	88.6	100.0	44.3	47.1	38.6
DS-V4-Flash	19	3	3	100.0	100.0	100.0	93.3	100.0	100.0
Correspondence									
Sol (high)	33	32	31	92.3	76.1	94.7	35.5	37.4	38.1
Sol (max)	28	26	26	91.5	89.2	94.9	33.1	38.5	29.2
Luna	22	17	17	95.3	80.0	91.5	55.3	51.8	57.6
Opus 5	24	23	23	95.7	93.9	96.8	25.2	27.0	25.2
DS-V4-Flash	19	6	6	100.0	100.0	100.0	96.7	100.0	100.0
Search									
Sol (high)	37	37	37	94.6	81.1	89.6	54.6	58.4	59.5
Sol (max)	28	26	25	96.0	88.8	96.4	50.4	53.6	48.0
Luna	20	10	10	100.0	80.0	90.7	54.0	50.0	44.0
Opus 5	21	12	12	96.7	90.0	92.6	33.3	38.3	38.3
DS-V4-Flash	19	5	5	88.0	88.0	91.7	80.0	84.0	80.0
State tracking									
Sol (high)	46	46	46	93.9	90.4	96.1	35.2	33.9	34.8
Sol (max)	26	24	24	89.2	85.0	96.3	35.8	34.2	35.0
Luna	23	19	19	86.3	80.0	84.1	30.5	32.6	35.8
Opus 5	25	22	21	90.5	73.3	93.3	21.0	25.7	25.7
DS-V4-Flash	19	5	4	100.0	100.0	100.0	55.0	50.0	55.0

Table A17: Spatially-steered generation: episodes, accepted worlds and evaluator accuracy by profile and generating configuration (continued). Episodes, Accepted, and Evaluated are counts; the six model columns are accuracies (%) over scored responses.

Coding agent	Episodes	Accepted	Evaluated	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Measurement									
Sol (high)	45	45	43	100.0	95.8	98.6	66.0	69.3	65.6
Sol (max)	33	31	31	95.5	94.8	95.3	60.6	65.8	59.4
Luna	24	18	15	96.0	94.7	93.0	66.7	80.0	65.3
Opus 5	24	21	19	97.9	90.5	88.6	37.9	36.8	41.1
DS-V4-Flash	18	1	1	100.0	100.0	100.0	60.0	60.0	60.0
Prior conflict									
Sol (high)	40	38	38	92.6	90.0	94.1	71.6	76.8	77.4
Sol (max)	27	25	25	96.8	90.4	94.3	70.4	75.2	68.8
Luna	22	16	14	95.7	91.3	95.6	60.0	61.4	75.7
Opus 5	21	13	11	96.4	90.9	100.0	45.5	52.7	54.5
DS-V4-Flash	18	1	1	60.0	100.0	100.0	40.0	40.0	40.0
Global structure									
Sol (high)	40	38	34	95.3	85.8	92.4	58.2	69.4	63.5
Sol (max)	30	26	24	93.3	84.2	90.8	53.3	59.2	61.7
Luna	21	12	11	94.5	94.5	92.2	67.3	63.6	65.5
Opus 5	21	12	11	94.5	81.8	84.8	36.4	47.3	34.5
DS-V4-Flash	18	2	2	100.0	100.0	100.0	80.0	90.0	100.0
Declared transform									
Sol (high)	42	41	41	83.4	57.1	84.0	30.2	27.8	32.7
Sol (max)	27	25	25	83.2	66.1	78.1	37.6	34.4	24.0
Luna	23	19	15	81.3	71.2	76.7	54.7	50.7	53.3
Opus 5	24	21	21	84.8	71.4	78.3	23.8	21.9	27.6
DS-V4-Flash	19	1	1	100.0	80.0	100.0	20.0	20.0	20.0

H.5 Evaluation coverage and eligibility for model-feedback-steered generation

All three frontier evaluators return scored responses for the same 6,078 profile-steered instances and 3,936 spatially-steered instances. At least two of the three are incorrect on 192/6,078 (3.2%) and 212/3,936 (5.4%), respectively. All six evaluators are incorrect on 32/6,078 (0.5%) and 24/3,936 (0.6%) instances with scored responses from all three frontier evaluators. Results by profile, generating configuration, and their cross-product are retained in the released evaluation record. Because the two generation experiments were separate campaigns, these comparisons describe their evaluation sets and do not identify a causal effect of spatial steering.

To be eligible as a feedback example, a world must have scored responses from all three frontier evaluators on all five instances (15 responses). The eligibility pool contains 1,547 such worlds. Of these, 70 (4.5%) are hard for frontier models: at least two frontier evaluators get two or more of the five instances wrong. The other 625 of the 2,172 exported worlds have incomplete frontier evaluations and are excluded from this rate. Selection of the two feedback example sets from this pool is described in Sec. I.1.

H.6 Evaluation cost

Frontier evaluation of the 6,490 profile-steered and 4,370 spatially-steered instances cost $425.74 through OpenRouter (Tab. A18). Costs include responses that were refused or could not be parsed. Reasoning tokens account for over 96% of completion tokens for every evaluator.

Table A18: Frontier-evaluator token usage (millions, prompt plus completion) and OpenRouter cost (USD), over 6,490 profile-steered and 4,370 spatially-steered instances per evaluator.

	Profile-steered		Spatially-steered	
Evaluator	Tokens	Cost	Tokens	Cost
Sol (high)	8.1	47.44	6.2	39.74
Opus 5	11.2	156.34	7.6	105.50
Gemini 3.7 Flash	20.0	42.80	14.7	33.93
Total	39.3	246.57	28.5	179.17

I MODEL-FEEDBACK-STEERED GENERATION

Model-feedback-steered generation supplies earlier question implementations and feedback from the three frontier evaluators to guide generation. We describe these inputs and report the generated worlds, their evaluation results, and the subset hard for frontier models. Generated examples appear in Sec. B.3.

I.1 FEEDBACK EXAMPLE SETS AND EVALUATOR-ERROR SUMMARIES

The evaluation used to select feedback examples is reported in App. H.

The feedback examples were selected from 70 replay-verified worlds that were hard for frontier models: 35 from profile-steered generation and 35 from spatially-steered generation. Each had all 15 frontier responses, with at least two evaluators answering at most three of five instances correctly. A deterministic selection procedure formed two disjoint sets of ten worlds. Each set contains five worlds from each experiment and two from each coding-agent configuration. The sets share no exact question or image matches and no near-duplicate clusters.

Each campaign receives the source and tests for its ten feedback examples, the three implementation demonstrations, and a card describing the feedback example set. Initial feedback supplies two detailed sample records per world, including the question, answer choices, gold answer, evaluator predictions, and correctness. Aggregate evaluator accuracies are computed from five instances per world. Initial feedback contains neither recorded confidence values nor public rationales. When a new world passes the final checks, the controller renders ten deterministic scenes and selects five unique images for the three evaluators. All three evaluators are called through OpenRouter (`openai/gpt-5.6-sol`, `anthropic/claude-opus-5`, and `google/gemini-3.7-flash`) with high reasoning and at most 65,536 completion tokens. Later agents may receive predictions, correctness, confidence, and concise public rationales from completed evaluations. The feedback channel does not deliver evaluator images or private provider reasoning. Later episodes can, however, receive gallery images from the most recent rejected submission. A world with fewer than 15 responses remains in the experiment record but is excluded from hardness rates and evaluator accuracy.

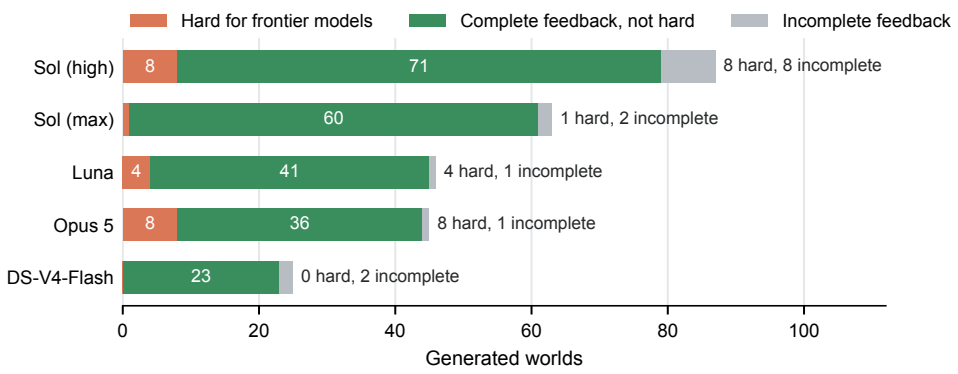
Feedback example set A also includes a retained 32-record manifest for one feedback example, containing questions, answers, scene specifications, and random seeds. Its referenced images are absent. The manifest was available to agents. These feedback-example records are distinct from final-verification scenes for new submissions.

The delivered instructions contain two inconsistencies. Initial feedback prose states that gold answers remain controller-side, but the JSON supplies them. The initial prompt also refers to five conceptual examples, although ten feedback examples were supplied. The delivered artifacts are preserved unchanged; their effect on generation was not measured.

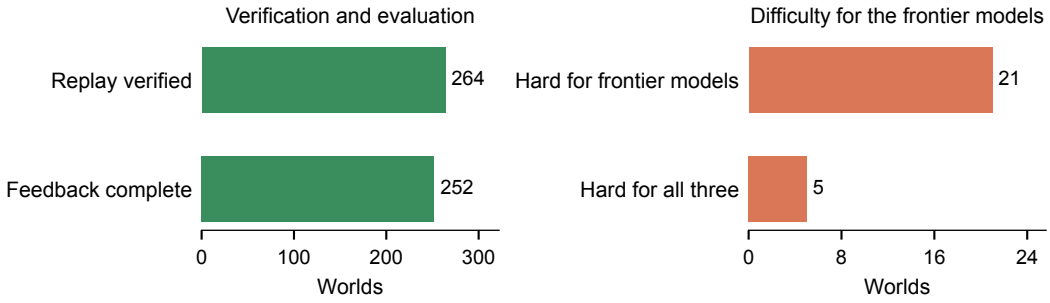
All ten campaigns use the same difficulty-feedback policy. Both Opus 5 campaigns and the DS-V4-Flash campaign using feedback example set B additionally validate the required fields in feedback records; the other seven campaigns do not include this step. The final checks, frontier evaluators, five-instance rule, and hardness threshold did not change. Comparisons across coding-agent configurations and feedback example sets are therefore descriptive.

I.2 GENERATION OUTCOMES, REPLAY, AND EXCLUDED RECORDS

The ten campaigns generated 266 worlds that passed the final checks in 295 episodes (Tab. A19). Complete frontier evaluations are available for 252 worlds; the 45 responses from 14 incomplete frontier evaluations are retained but excluded from difficulty and accuracy estimates. Replay verified 264 of the 266 worlds, including all 21 worlds hard for frontier models. Of the two worlds that failed replay, the DS-V4-Flash world has an incomplete frontier evaluation and is therefore already excluded from difficulty and accuracy estimates. The Opus 5 world has a complete frontier evaluation (14 of 15 responses correct, not hard for frontier models) and remains in the 252-world evaluation, including the open-weight and fine-tuning evaluations.



(a) Generated-world outcomes by configuration.



(b) Final-check, evaluator, and human-review states.

Figure A54: Outcomes of model-feedback-steered generation. Complete frontier evaluations, replay verification, hardness labels, and human-review coverage are counted separately. Inclusion in the human-review sample does not mean a world was judged valid.

Table A19: Model-feedback-steered generation outcomes by campaign. Complete denotes all 15 evaluator responses. Replay is reported over all worlds that passed the final checks. Set is the feedback example set (A or B). Field checks indicates whether the campaign validates the required fields in feedback records.

Configuration	Set	Episodes	Worlds	Complete	Hard	Replay	Field checks
Sol (high)	A	45	44	37	7	44/44	No
Sol (high)	B	43	43	42	1	43/43	No
Sol (max)	A	32	31	29	1	31/31	No
Sol (max)	B	32	32	32	0	32/32	No
Luna	A	27	24	24	4	24/24	No
Luna	B	24	22	21	0	22/22	No
Opus 5	A	25	22	21	4	21/22	Yes
Opus 5	B	27	23	23	4	23/23	Yes
DS-V4-Flash	A	21	15	14	0	15/15	No
DS-V4-Flash	B	19	10	9	0	9/10	Yes
Total		295	266	252	21	264/266	

I.3 Frontier-model and post-run open-weight evaluation

Tab. A20 reports both evaluator groups by generating configuration. The frontier models supply feedback and determine which worlds count as hard. The open-weight models are evaluated only after generation.

Table A20: Model-feedback-steered generation by coding-agent configuration. Passed counts worlds passing final checks; Replay counts worlds passing the later 200-scene replay. Hard counts worlds hard for frontier models among the n worlds with all 15 frontier-model responses: at least two evaluators answer at most three of five instances correctly. In-loop accuracy is for the frontier models that provide generation feedback. Post-run accuracy is for open-weight models evaluated after generation; they provide no feedback and do not determine which worlds count as hard. Accuracies (%) use five instances per world. Post-run scores count unparseable responses as incorrect.

	Generation outcomes					In-loop (%)			Post-run (%)		
Configuration	Passed	n	Replay	Hard	Hard (%)	Sol (high)	Opus 5	Gemini 3.7 Flash	Gemma 4-12B	Gemma 4-31B	Qwen 3.8-27B
Sol (high)	87	79	87	8	10.1	96.5	88.4	91.6	28.1	28.1	30.6
Sol (max)	63	61	63	1	1.6	97.4	93.4	97.0	29.2	26.2	23.0
Luna	46	45	46	4	8.9	97.8	93.8	90.7	42.2	40.4	37.3
Opus 5	45	44	44	8	18.2	90.9	87.3	88.6	20.9	25.5	21.4
DS-V4-Flash	25	23	24	0	0.0	100.0	98.3	100.0	51.3	42.6	46.1
Overall	266	252	264	21	8.3	96.3	91.3	93.0	31.7	30.7	29.8

The post-generation open-weight evaluation covers all 252 worlds with complete frontier evaluations, five instances each. Gemma 4 12B, Gemma 4 31B, and Qwen 3.8 27B answer 400, 387, and 375 of the 1,260 instances correctly. Gemma 4 12B has seven unparseable responses, all on Sol (high) worlds, which are scored as incorrect. The other two models have none. Per configuration, the accuracies are computed over five times the n worlds in Tab. A20: 395, 305, 225, 220, and 115 instances for Sol (high), Sol (max), Luna, Opus 5, and DS-V4-Flash. These evaluations supply no feedback during generation and play no part in deciding which worlds are hard for frontier models.

I.4 Worlds hard for frontier models and evaluator overlap

Of the 252 worlds with all 15 frontier responses, 21 are hard for frontier models. All three evaluators score at most 3/5 on five of these worlds; exactly two evaluators do so on the other 16.

The evaluators' failures overlap only partly (Figs. A55 and A56). Among the 21 worlds hard for frontier models, the most common failure pair is Opus 5 with Gemini 3.7 Flash (11 worlds). Accuracy on this selected subset is 70.5% for Sol (high), 42.9% for Opus 5, and 49.5% for Gemini 3.7 Flash. Because these same evaluators define the subset, the values characterize the generated worlds and are not held-out estimates of general model accuracy.

Selected generated questions and frontier-model scores appear in Sec. B.3.

I.5 Differences across coding-agent configurations, feedback example sets, and declared mechanisms

Campaigns using feedback example set A produced 16 worlds hard for frontier models among 125 worlds with complete evaluator responses, compared with 5 among 127 for set B (Fig. A57a). The intervals overlap, and each coding-agent configuration was run only once with each set. These rates describe campaign outputs. With only one run per coding-agent configuration and feedback set, we cannot determine how much of the difference is due to the choice of feedback examples. The mechanism labels in Fig. A57b are declarations by the coding agents. Topological connectivity accounts for 12 worlds hard for frontier models among 37 completely evaluated worlds, but these labels are neither exclusive nor assigned by an independent annotator.

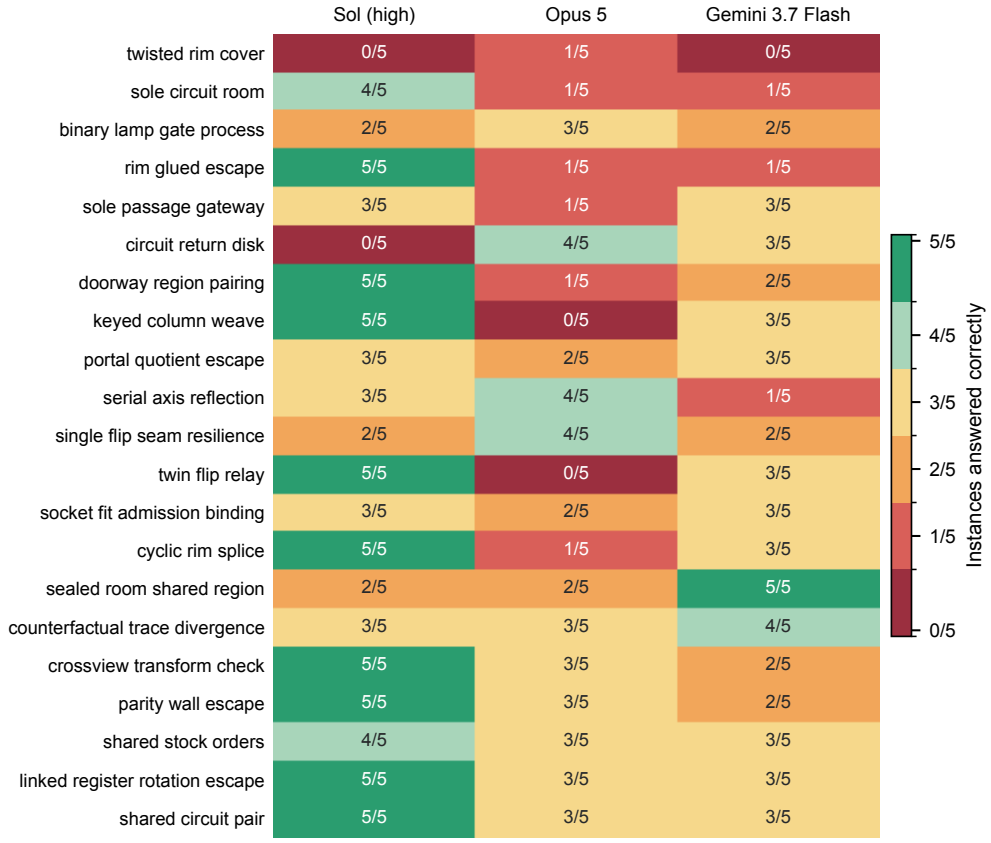
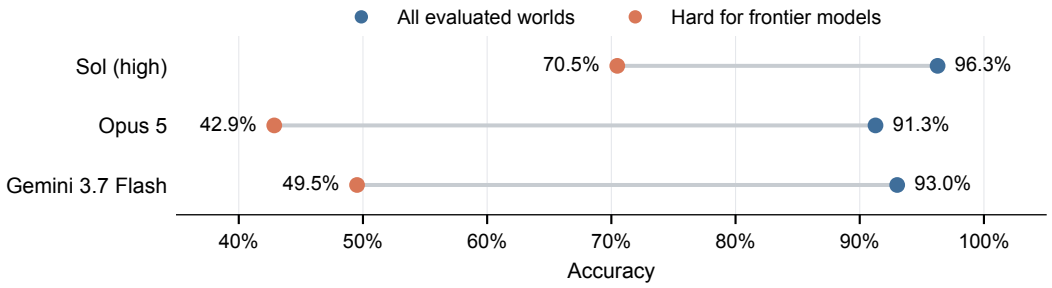


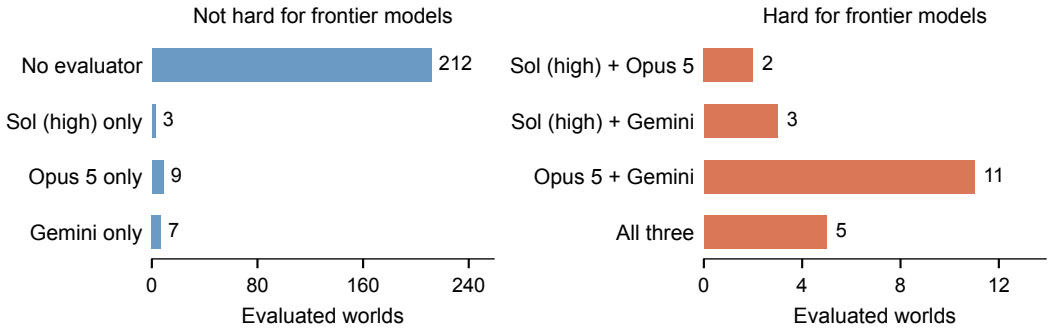
Figure A55: Per-evaluator outcomes for all 21 worlds hard for frontier models. Each cell reports correct answers out of five fixed instances. Cells at 3/5 or below count as an evaluator failure under this criterion.

I.6 Evaluator cost

The three-model evaluations cost $135.45 through OpenRouter, excluding coding-agent subscriptions and usage charges. Fig. A58 reports evaluator expenditure alongside campaign throughput.

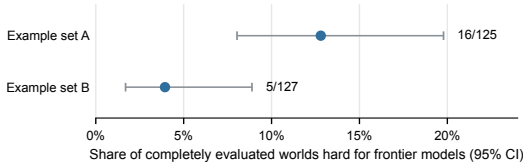


(a) Worlds with complete frontier evaluations and the subset hard for frontier models.

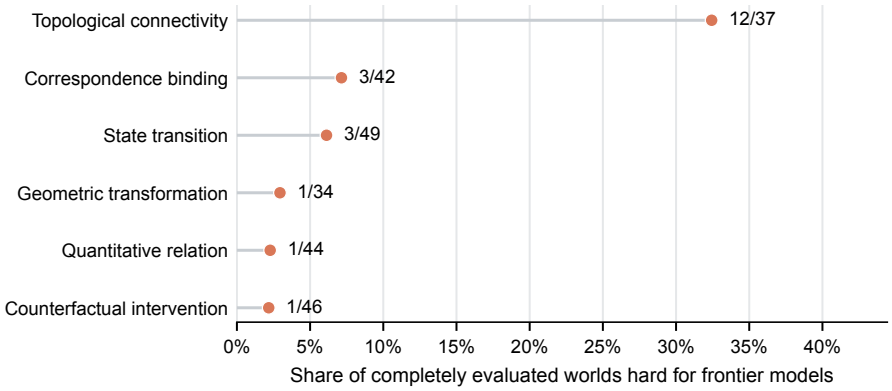


(b) Mutually exclusive evaluator failure sets.

Figure A56: Evaluator outcomes in model-feedback-steered generation. The subset hard for frontier models is selected by the same frontier models shown here and is not an independent benchmark.

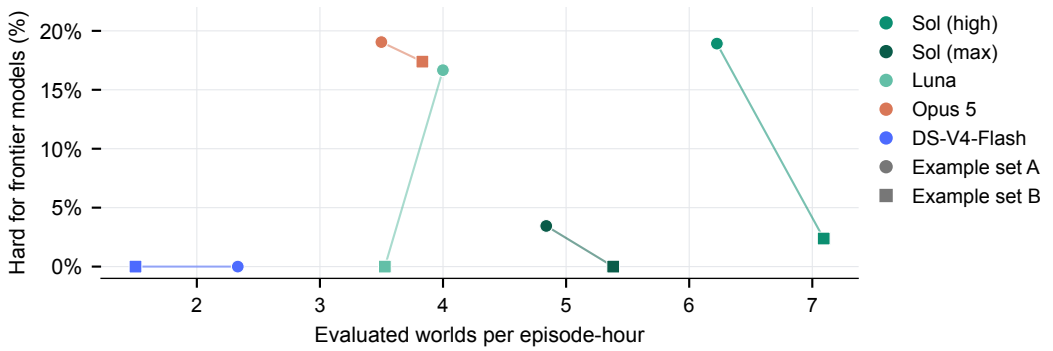


(a) Share of worlds hard for frontier models, by feedback example set.

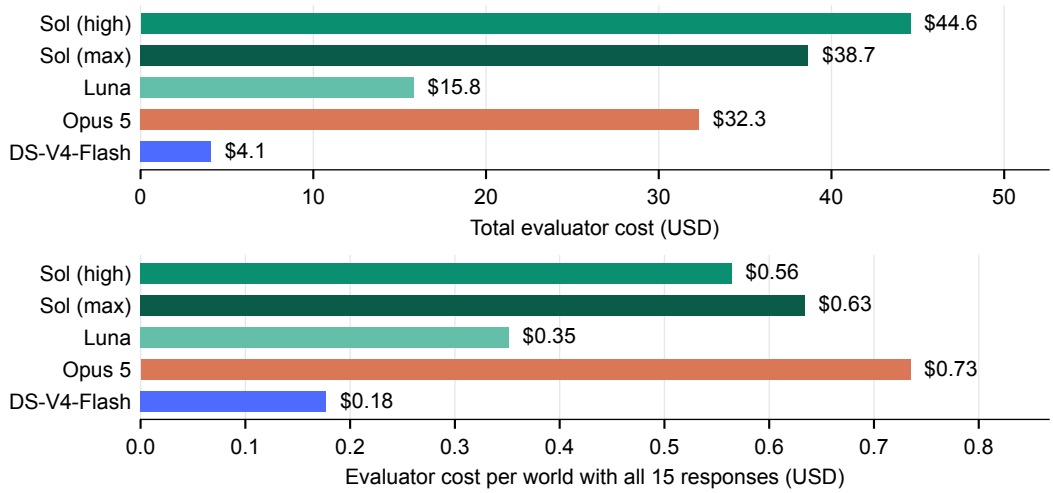


(b) Descriptive rates by declared mechanism.

Figure A57: Variation across the two feedback example sets and the mechanism declared for each generated world. Error bars in (a) are 95% Wilson intervals.



(a) Yield of worlds hard for frontier models, and evaluator throughput.



(b) Evaluator spend and spend per complete frontier evaluation.

Figure A58: Campaign throughput and evaluator cost. Circles denote feedback example set A and squares denote feedback example set B. Lines join the two campaigns of one coding-agent configuration only as a visual aid; comparisons spanning differences in feedback-record validation are descriptive. Coding-agent subscription costs are excluded from (b).

J HUMAN REVIEW

J.1 REVIEW PROCEDURE AND SAMPLING

Two reviewers independently assessed the same 800 instances: 400 from profile-steered generation, 200 from spatially-steered generation, and 200 from model-feedback-steered generation. They recorded an answer and could flag an instance with a free-text explanation. Our reason classification groups concerns about the recorded answer and concerns about ambiguity under the label *invalid*.

Within each generation experiment, instances were sampled at random within strata defined by the generating coding-agent configuration and, where applicable, the discovery profile. The review tool showed each image and question, recorded the reviewer's answer, and allowed a free-text reason for any flag. We then mapped each reason string to one of three outcomes: a concern about the recorded answer, a concern about ambiguity or clarity, or no concern. Both concern types count as *invalid*. An ambiguous sample did not mean that the reviewer could not solve it, but rather that the problem could have been phrased more clearly. This is shown by the reviewers scoring 96% correct on average. The first reviewer flagged 37 instances and the second 43, of which 13 overlap.

J.2 REVIEW OUTCOMES AND CLASSIFICATION

Under our reason classification, 67 of 800 instances (8.4%) were classified as invalid by at least one reviewer: 30 of 400 (7.5%) from profile-steered generation, 17 of 200 (8.5%) from spatially-steered generation, and 20 of 200 (10.0%) from model-feedback-steered generation. Tab. A21 counts instances flagged by neither, one, or both reviewers. Tabs. A22 to A24 break the invalid rate down by generating model, discovery profile, and generation experiment, and Tab. A25 reports agreement between the two reviewers.

Table A21: Human-review outcomes under our reason classification. Neither, One, and Both count items classified as invalid by zero, one, or two reviewers, respectively.

Experiment	Instances	Neither	One	Both
Profile-steered	400	370	25	5
Spatially-steered	200	183	13	4
Model-feedback-steered	200	180	16	4
Total	800	733	54	13

The reviewers' classifications agree on 746 of 800 items (93.25%). They both identify a concern on 13 of the 67 items identified by at least one reviewer; the remaining 54 are disagreements. Across the 1,600 reviewer–item judgments, 80 are classified as invalid.

Table A22: Human-review invalid rate by generating configuration, pooling all three generation experiments. Judgments are counted per reviewer–item pair (each of the two reviewers' readings of an item counted separately).

Model	Luna	Sol (high)	Sol (max)	Opus 5	DS-V4-Flash	Total
Rate (%)	6.2	5.7	4.7	3.8	3.8	5.0

The proportion of reviewer–item judgments classified as invalid ranges from 3.8% for Opus 5 and DS-V4-Flash to 6.2% for Luna. These pooled rates describe the reviewed sample. Without subgroup counts and uncertainty estimates, they do not establish differences between coding-agent configurations.

Table A23: Human-review invalid rate by discovery profile, for profile-steered and spatially-steered generation only. Model-feedback-steered generation does not use the nine discovery profiles and is excluded here.

Profile	Rate (%)
Declared transform	14.3
Correspondence	4.5
Search	4.3
State tracking	4.3
Global structure	4.0
Measurement	3.3
Tracing	2.7
Topology	1.5
Prior conflict	0.9
Total	4.7

Among the reviewed profile-steered and spatially-steered instances, the declared-transform profile has the highest reported judgment-level invalid rate (14.3%). The remaining profiles range from 0.9% to 4.5%. These rates pool two experiments and both reviewers' judgments.

Table A24: Human-review invalid rate by generation experiment, counting each reviewer–item judgment separately (versus Tab. A21, which counts each item once).

Experiment	Invalid rate (%)
Profile-steered	4.38
Spatially-steered	5.25
Model-feedback-steered	6.00
Overall	5.00

The judgment-level rates are 4.38%, 5.25%, and 6.00% for profile-steered, spatially-steered, and model-feedback-steered generation, respectively. These correspond to 35 of 800, 21 of 400, and 24 of 400 judgments. The review does not isolate why these rates differ.

Table A25: Agreement between the two reviewers on the invalid labels under the reason classification. The coefficients use different adjustments for chance agreement; the raw item counts are reported in Tab. A21.

	Cohen's kappa	PABAK	Gwet's AC1
Overall	0.29	0.865	0.925

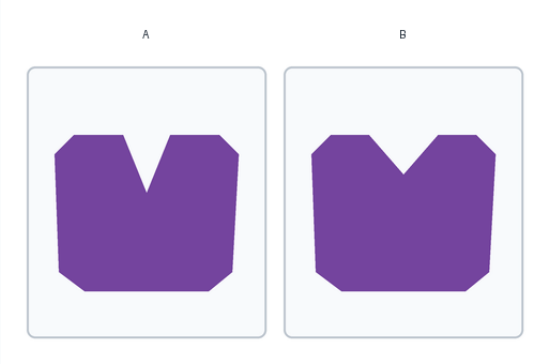
Raw agreement is 93.25%, largely because neither reviewer flagged 733 of 800 instances. Agreement on flagged instances is lower: both reviewers flagged 13 of the 67 instances flagged by at least one reviewer. Cohen's kappa is 0.29, PABAK is 0.865, and Gwet's AC1 is 0.925. These coefficients treat chance agreement differently and can diverge when one label is rare (Feinstein & Cicchetti, 1990; Cicchetti & Feinstein, 1990; Byrt et al., 1993; Gwet, 2008). They measure agreement on the classified labels, not whether an instance is valid or whether the reviewers gave the same answer.

J.3 ILLUSTRATIVE CONCERNS

The following reviewer-selected examples illustrate unclear instructions and competing interpretations. We preserve the question, recorded answer, and each reviewer's response rather than assigning a new answer. These examples are not a random sample of failures.

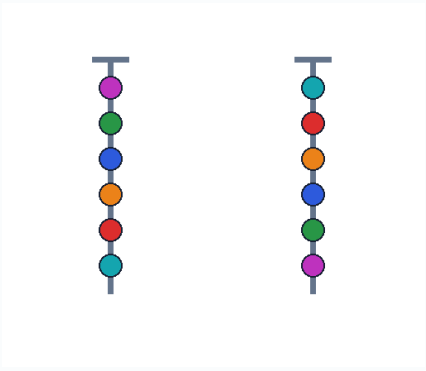
Human-review examples: instructions and interpretation

(a) An unspecified comparison
Generated by: Luna. **Experiment:** Profile-steered generation.



Question: "The two filled outlines have controlled area and size. Using only the visible top-notch depth, choose panel A or B."
Recorded answer: A.
Review: Both reviewers answered A. One flagged "no clear instruction given"; the other did not flag the item. The question does not specify whether to choose the deeper or shallower notch.

(b) Competing names for the resulting shape
Generated by: Sol (high). **Experiment:** Profile-steered generation.



Question: "Mentally draw a straight segment between every two dots that share a color. Is the resulting stitch a ladder, three Xs, a starburst, or a braid?"
Recorded answer: starburst.
Review: One reviewer answered starburst but flagged "multiple valid right answers". The other answered three Xs and also flagged the item. Their responses point to an ambiguity in how to name the constructed shape.

K Training on Generated Questions

We describe the training data and optimization, then report transfer to model-feedback-steered worlds and external benchmarks. Additional evaluations cover held-out profile-steered worlds, answer-option order, direct training on model-feedback-steered worlds, and transfer to profiles excluded from training.

K.1 Training data and optimization

The main training data contains only profile-steered worlds. The fixed split assigns 748 of 793 worlds to training and 45 to testing. The training pool contains 27,762 instances, with 555 reserved for validation; the test set contains 1,691 instances. Transfer evaluation uses 1,260 instances from 252 model-feedback-steered worlds with different world identifiers.

The 793 worlds are a subset of the 1,301 replay-verified profile-steered worlds, spanning all five coding-agent configurations and nine profiles. The retained share differs by configuration: 83 of 392 Sol (high) worlds, 147 of 250 Sol (max), 172 of 196 Luna, 159 of 211 Opus 5, and 232 of 252 DS-V4-Flash worlds. Each retained world contributes between 11 and 100 instances with a median of 40. Each test split holds out one world from each of the 45 model–profile campaigns. Worlds were retained by a stratified example subsample: for each (campaign, prompt-family) group, at most 20 examples were kept (11 for the DS-V4-Flash source, which has many more distinct prompt-family variants per campaign than the other four). Worlds left with no examples after the cap was applied were excluded.

Training uses `transformers.Trainer` and PEFT LoRA adapters on the language model's attention and feed-forward projections; the vision tower is not directly adapted. The main recipe uses rank 16, scaling 32, dropout 0.05, bfloat16, one epoch, learning rate 10^{-4}, cosine decay, 3% warm-up, batch size two, and eight accumulation steps. The loss covers only the assistant answer and turn-end token. Thinking mode is disabled for Qwen 3.8 27B. Runs with sixteen accumulation steps are identified separately.

Training runs on NVIDIA A100-80GB GPUs, using two GPUs for Gemma 4 31B and Qwen 3.8 27B and one for Gemma 4 12B. Some leave-one-profile-out runs used A100-40GB or H200 GPUs instead. The software stack is PyTorch 2.13.0, Transformers 5.14.1, PEFT 0.20.0, and CUDA 13.0. Fine-tuning on profile-steered worlds took approximately 3.5 hours for Gemma 4 12B, 8.3 hours for Gemma 4 31B, and 17.8 hours for Qwen 3.8 27B, or 55.6 GPU-hours in total. Training every model behind the main, four-seed, answer-option-order, and leave-one-profile-out results took 946.7 GPU-hours.

K.2 Transfer to model-feedback-steered questions

We test whether training on profile-steered worlds improves performance on worlds produced by model-feedback-steered generation. Using the training recipe above, we evaluate four independently fine-tuned checkpoints per model on 1,260 instances from 252 model-feedback-steered worlds. All three models improve on average across the four checkpoints (Tab. A26).

Table A26: Transfer from profile-steered training worlds to 1,260 instances from 252 model-feedback-steered worlds. Base is the unchanged checkpoint's accuracy. SFT is the mean ± standard deviation across four independently fine-tuned checkpoints. Δ is the mean SFT−Base change.

Model	Base (%)	SFT (%)	Δ pp
Gemma 4 12B	31.7	$34.5_{\pm 1.5}$	+2.8
Gemma 4 31B	30.7	$37.8_{\pm 0.5}$	+7.1
Qwen 3.8 27B	29.8	$35.8_{\pm 0.3}$	+6.0
Average	30.7	36.0	+5.3

K.3 External benchmark transfer

The 16 external benchmarks are VLMsAreBlind (Rahmanzadehgervi et al., 2024), PuzzleVQA (Chia et al., 2024), VLMsAreBiased (Vo et al., 2026), AlgoPuzzleVQA (Ghosal et al., 2025), PGM (Barrett et al., 2018), V* (Wu & Xie, 2024), NaturalBench (Li et al., 2024), VSR (Liu et al., 2023), CV-Bench (Tong et al., 2024), GQA (Hudson & Manning, 2019), BLINK (Fu et al., 2024), A-OKVQA (Schwenk et al., 2022), DocVQA (Mathew et al., 2021), CLEVR (Johnson et al., 2017), MathVista (Lu et al., 2024), RAVEN (Zhang et al., 2019). Tab. A27 reports external-benchmark results across four training splits. All three models have positive mean changes on VLMsAreBlind, PuzzleVQA, VLMsAreBiased, AlgoPuzzleVQA, PGM, and NaturalBench. Transfer varies by benchmark and model. RAVEN has negative mean changes for all three models, but the standard deviations across splits are larger than the magnitudes of those changes. We use four training splits to measure how consistent each change is.

Table A27: External-benchmark accuracy before and after SFT, across four training splits. Base is the unchanged checkpoint's accuracy. SFT is the mean ± standard deviation across four independently fine-tuned checkpoints. Δ is the mean SFT−Base change (its standard deviation equals SFT's, since Base is fixed across splits). Avg. Δ is the mean change across all three models. DocVQA uses ANLS ×100. BLINK covers IQ_Test only; RAVEN covers center_single only. V* scores only use instances for which both the base and the corresponding SFT checkpoint return a parseable answer.

	Gemma 4 12B			Gemma 4 31B			Qwen 3.8 27B			
Benchmark	Base	SFT	Δ	Base	SFT	Δ	Base	SFT	Δ	Avg. Δ
VLMsAreBlind	46.4	77.3±0.7	+30.8	74.6	82.9±1.2	+8.3	91.3	94.2±1.1	+2.9	+14.0
PuzzleVQA	58.0	63.5±2.3	+5.5	68.1	71.8±2.9	+3.7	58.6	68.7±2.2	+10.0	+6.4
VLMsAreBiased	26.7	31.2±1.7	+4.5	23.6	28.6±3.1	+5.0	43.1	47.5±2.4	+4.4	+4.6
AlgoPuzzleVQA	35.7	37.3±0.7	+1.6	38.8	43.3±1.4	+4.5	36.9	40.1±1.7	+3.2	+3.1
PGM	13.2	14.0±1.4	+0.8	14.3	15.8±0.6	+1.5	15.8	17.7±1.0	+1.9	+1.4
V*	56.2	55.6±0.9	−0.6	75.1	72.1±1.3	−3.1	81.7	89.6±1.2	+7.8	+1.4
NaturalBench	78.6	79.8±0.5	+1.1	82.9	83.5±0.2	+0.6	81.4	83.1±0.5	+1.8	+1.2
VSR	80.0	79.8±1.6	−0.2	84.0	84.6±0.3	+0.6	85.8	87.5±0.5	+1.7	+0.7
CV-Bench	83.2	83.0±0.9	−0.2	85.6	86.3±0.5	+0.7	87.1	87.7±0.3	+0.6	+0.4
GQA	56.3	56.4±1.0	+0.1	58.0	56.8±1.1	−1.2	62.8	64.3±0.2	+1.5	+0.1
BLINK	29.3	27.7±1.2	−1.7	27.3	27.0±1.3	−0.3	26.0	28.3±1.3	+2.3	+0.1
A-OKVQA	84.1	81.4±0.9	−2.7	89.3	89.8±0.3	+0.4	87.6	88.7±0.1	+1.1	−0.4
DocVQA	81.8	80.1±0.6	−1.7	92.3	92.7±0.6	+0.3	96.0	96.2±0.4	+0.2	−0.4
CLEVR	60.0	54.7±1.0	−5.3	68.4	69.4±3.7	+1.0	95.0	97.5±0.3	+2.5	−0.6
MathVista	54.6	51.7±2.4	−2.9	55.8	52.0±6.3	−3.8	53.3	57.9±1.2	+4.6	−0.7
RAVEN	14.8	14.1±5.6	−0.8	22.6	20.8±3.1	−1.8	48.5	47.3±4.4	−1.2	−1.2
Average	53.7	55.5	+1.8	60.0	61.1	+1.0	65.7	68.5	+2.8	+1.9

K.4 Evaluation on held-out profile-steered worlds

This evaluation tests new worlds from the same generation experiment used for training. We report four independently trained checkpoints per model, one per random split, using the main training recipe above (Sec. 4.7). The fixed split contains 748 training worlds and 45 held-out worlds with 1,691 test instances. Each of the other three splits holds out a different set of 45 worlds, with 1,682, 1,700, and 1,669 test instances; no world is held out in all four splits, and any two splits share at most four test worlds. The mean ± standard deviation therefore reflects variation in both the training data and the test worlds. For Gemma 4 12B, the four checkpoints mix eight- and sixteen-step gradient accumulation. Mean accuracy rises after fine-tuning for all three models (Tab. A28).

Table A28: Accuracy on held-out profile-steered worlds before and after SFT, mean ± standard deviation across four independently trained checkpoints per model. Each checkpoint is evaluated on its own held-out split of 45 worlds (1,669–1,700 instances), disjoint from that checkpoint's training worlds.

Model	Base (%)	SFT (%)	Δ pp
Gemma 4 12B	57.8±1.0	69.2±3.7	+11.5
Gemma 4 31B	64.4±1.7	74.6±3.8	+10.2
Qwen 3.8 27B	59.8±2.2	70.2±4.7	+10.4
Average	60.7	71.3	+10.7

K.4.1 Answer-option order

The coding agents that generate these worlds do not place the correct answer uniformly across candidate positions. Since training and test worlds come from the same generation process, a model could inflate its measured accuracy by learning a positional bias instead of the underlying visual task. To rule this out, we retrain and re-evaluate each seed-42 checkpoint with every example's answer, in both the training and test data, reassigned to a balanced-random position, holding every other hyperparameter fixed. If the original gain depended on exploiting answer position, this fully rebalanced setup should score lower than the original checkpoint. Instead, accuracy changes from 71.0% to 71.5% for Gemma 4 12B, from 74.4% to 75.7% for Gemma 4 31B, and from 71.8% to 72.5% for Qwen 3.8 27B – a small, non-negative change in every case. The SFT gain is therefore not attributable to answer-position leakage. This comparison is a separate evaluation, not part of the four-split means in Tab. A28.

K.5 Direct training on model-feedback-steered questions

In a separate experiment, we train directly on replay-verified worlds from model-feedback-steered generation. The training pool contains 63,857 instances from 214 worlds, capped at 32,560 instances. Evaluation uses 250 instances from 50 different worlds, all excluded from training. All three models use the main LoRA recipe with eight-step gradient accumulation. Accuracy increases by 0.8 percentage points for Gemma 4 12B, 19.2 for Gemma 4 31B, and 12.8 for Qwen 3.8 27B (Tab. A29).

Table A29: Direct adaptation on model-feedback-steered worlds, evaluated on 250 instances from 50 held-out worlds.

Model	Base (%)	SFT (%)	Δ pp
Gemma 4 12B	28.8	29.6	+0.8
Gemma 4 31B	27.2	46.4	+19.2
Qwen 3.8 27B	25.6	38.4	+12.8
Average	27.2	38.1	+10.9

K.6 Transfer to excluded profiles

For each model, we train nine separate adapters, each excluding all worlds from one discovery profile, and evaluate on the instances from the excluded profile. The training sets contain roughly 660–720 worlds and 24,500–26,800 instances, depending on the profile. These runs use the main LoRA recipe. For Gemma 4 12B, whose eight-step runs collapsed on several profiles (Tab. A31), we use sixteen accumulation steps instead. Gemma 4 31B and Qwen 3.8 27B use the standard eight-step recipe throughout. Each cell reports a single run. Accuracy improves in 26 of the 27 model–profile combinations (Tab. A30); the exception is Gemma 4 12B on measurement, which decreases by 1.5 percentage points. Averaged over profiles, the gain is +6.8 pp for Gemma 4 12B, +7.3 pp for Gemma 4 31B, and +11.4 pp for Qwen 3.8 27B.

Table A30: Accuracy before and after SFT when an entire profile is excluded from training and the checkpoint is evaluated on that profile's own held-out worlds. One checkpoint per profile per model (single seed, not averaged across seeds). Avg. Δ is the mean change across all three models.

	Gemma 4 12B			Gemma 4 31B			Qwen 3.8 27B			
Excluded profile	Base	SFT	Δ	Base	SFT	Δ	Base	SFT	Δ	Avg. Δ
Correspondence	51.6	60.3	+8.7	54.9	67.9	+13.0	54.3	66.9	+12.6	+11.4
Measurement	68.2	66.7	−1.5	74.7	80.7	+6.0	65.2	79.0	+13.8	+6.1
Prior conflict	77.9	84.3	+6.4	84.9	86.6	+1.7	80.7	87.8	+7.1	+5.1
Declared transform	51.8	54.9	+3.1	58.6	61.3	+2.7	51.2	55.2	+4.0	+3.3
Search	60.5	69.3	+8.8	65.1	74.8	+9.7	61.6	72.0	+10.5	+9.7
Global structure	64.5	68.6	+4.1	69.7	70.9	+1.2	63.6	72.3	+8.6	+4.6
State tracking	36.2	45.5	+9.3	40.7	49.8	+9.1	33.5	47.9	+14.4	+10.9
Topology	63.0	74.8	+11.8	66.7	77.3	+10.6	60.3	74.9	+14.6	+12.3
Tracing	56.3	67.0	+10.7	62.9	74.6	+11.7	53.9	70.5	+16.7	+13.0
Average	58.9	65.7	+6.8	64.2	71.5	+7.3	58.3	69.6	+11.4	+8.5

K.6.1 TRAINING-RUN COMPARISON

Tab. A31 reports Gemma 4 12B's eight- and sixteen-step accumulation runs for the excluded-profile study (the other two models use only the eight-step recipe throughout), including the available retries with different random seeds. Results vary across recipes and runs. For example, the eight-step measurement runs score 0.0% and 75.6%, while the sixteen-step run scores 66.7%. We report the sixteen-step run for every profile in Tab. A30. The table retains the other runs, including those with higher accuracy, so the variation is visible.

Table A31: Runs for the leave-one-profile-out study, by number of gradient-accumulation steps. Reported marks the run shown in Tab. A30: the sixteen-step run for every profile, regardless of which run scores higher. Three profiles also have a repeat run with eight-step gradient accumulation and a different random seed.

Excluded profile	8 steps (%)	16 steps (%)	8 steps, repeat (%)	Reported
Correspondence	61.9	60.3	–	16 steps
Measurement	0.0	66.7	75.6	16 steps
Prior conflict	40.6	84.3	41.3	16 steps
Declared transform	34.4	54.9	–	16 steps
Global structure	68.2	68.6	–	16 steps
Search	73.1	69.3	–	16 steps
State tracking	26.2	45.5	–	16 steps
Topology	71.3	74.8	–	16 steps
Tracing	40.3	67.0	68.3	16 steps