REFLECTION-DRIVEN CODE-WITH-IMAGE REASONING

Anonymous authors
Paper under double-blind review

ABSTRACT

Many visual tasks ask for quantitative properties of an image, such as displacement, rotation angle, or color cast. Solving these tasks requires computation over image pixels rather than visual inspection alone. We study *Code-with-Image*, a problem-solving paradigm in which a large vision-language model designs an algorithm, implements it as code, and executes it in a general-purpose interpreter to compute the answer. Each attempt leaves behind code and execution records that provide evidence for diagnosing failures, creating an opportunity for models to improve their solution strategies through reflection. To evaluate both computational visual reasoning and improvement through reflection, we introduce *Code-with-Image Bench* (CwI-Bench), a benchmark of 30 procedurally generated tasks with verifiable answers. Its disjoint training, validation, and test instances separate strategy refinement and selection from evaluation on unseen instances. We propose Reflection-Driven Refinement, a framework that develops and refines reusable solution strategies through reflection on prior attempts without updating model weights. Without reflection, Code-with-Image outperforms chain-of-thought reasoning across all nine backbones. Our framework further raises test accuracy from 41.3% to 70.1% on GPT-5.6-luna and from 32.6% to 64.6% on Qwen3.5-27B. The strategies also transfer across model scales and families.

1 INTRODUCTION

Large vision–language models (LVLMs), which take an image and a question and answer in text, increasingly solve visual problems by actively manipulating images. In the emerging thinking-with-images paradigm (Su et al., 2025; Zheng et al., 2026; Hong et al., 2026; Su et al., 2026a; Zhao et al., 2025; Zhang et al., 2025; Hou et al., 2026; Song et al., 2026), a model may crop or zoom an image to inspect details (Wu & Xie, 2024; Zheng et al., 2026; Su et al., 2026a), sketch intermediate reasoning (Hu et al., 2024), adjust rotation or contrast (Zhang et al., 2025; Zhao et al., 2025), and combine them with web search (Hong et al., 2026; Wei et al., 2026). These operations help the model reveal useful visual evidence, but they do not directly compute the answer.

Many visual tasks require a different capability: the answer is not directly visible in any transformed view of the image and must instead be *computed from the pixels*. Examples include estimating the pixel displacement between two frames, measuring the skew angle of a scanned page, localizing a retouched region, or recovering the true color of a surface under an unknown color cast. Such problems arise in document processing, image forensics, photography, and scientific imaging. Solving them requires more than locating or inspecting visual evidence. The model must *construct and execute an algorithm* that turns pixel measurements into the requested answer. We ask: can an LVLM use an image as data for computation, rather than only as visual content to inspect?

We study this capability through *Code-with-Image*, a problem-solving paradigm in which the model treats the image as data, designs an algorithm for the task, implements the algorithm as code, and executes it in a general-purpose interpreter. Language reasoning determines which operations to perform and how to organize them, while code carries out the numerical and image-processing computations. The resulting program, intermediate values, and execution outputs also make the solution process explicit and inspectable. Our **first question** is therefore how well current LVLMs can devise computational procedures that recover answers directly from image pixels.

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.

Code-with-Image also creates an opportunity for self-improvement without updating model weights. When a model answers a visual question directly, a wrong prediction often provides little information about which reasoning step failed. In contrast, a Code-with-Image attempt leaves behind the code, intermediate outputs, and execution errors. These artifacts provide concrete evidence about why a solution failed. This leads to our **second question**: can an LVLM improve its own solution strategy by reflecting on previous attempts and reuse it on unseen instances?

To study the first question, we introduce **Code-with-Image Bench (CwI-Bench)**, a benchmark of 30 computational visual tasks spanning four families: transformation, perception and counting, localization, and color. Each task is defined by a generator that first samples a target answer and then renders an image from that target. This construction provides exact ground-truth answers and enables controlled generation of distinct instances. Each task has separate training, validation, and test sets. The model revises a reusable written strategy (or skill) on the training set, selects the best revision on validation, and evaluates it on unseen test instances.

To answer the second question, we propose **Reflection-Driven Refinement**, a framework that refines reusable skills in rounds: the solver attempts the training instances, the same model rewrites the skill from the records, and the final skill is selected on held-out validation instances. Observational reflection (O-Refl) analyzes the recorded attempts; executable reflection (E-Refl) runs diagnostics and tests repairs when the records leave the cause uncertain; cross-round reflection (C-Refl) reviews the sequence of revisions and recurring errors to revisit assumptions that survived earlier fixes.

Experiments across nine LVLM backbones show that computation itself improves accuracy on every backbone. Bare Code-with-Image improves accuracy over chain-of-thought reasoning on all nine backbones, from 14.4% to 32.6% on Qwen3.5-27B (Yang et al., 2025) and from 25.6% to 41.3% on GPT-5.6-luna (OpenAI, 2026). Reflection-Driven Refinement further raises them to 64.6% and 70.1%, respectively. Skills refined through reflection also transfer across model scales and families.

Contributions. (i) We formulate Code-with-Image and develop CwI-Bench, a benchmark that evaluates computational visual problem solving on unseen instances before and after reflection. (ii) We propose Reflection-Driven Refinement, which improves solution strategies without weight updates by analyzing recorded attempts, testing diagnoses in code, and reassessing the revision history. (iii) We provide evidence of gains from code execution and reflection across open and closed models, with ablations of the reflection modes, analysis of skill refinement, and cross-model skill transfer.

2 Related Work

Thinking with images. Visual tool use ranges from guided crop-and-zoom search (V* (Wu & Xie, 2024)) and sketched intermediate reasoning (Visual Sketchpad (Hu et al., 2024)) to RL-trained image inspection (DeepEyes (Zheng et al., 2026); Pixel Reasoner (Su et al., 2026a)). PyVision (Zhao et al., 2025; 2026), Thyme (Zhang et al., 2025), CodeV (Hou et al., 2026), and CodeDance (Song et al., 2026) let the model write code, and VisProg (Gupta & Kembhavi, 2023) and ViperGPT (Surís et al., 2023) compose vision APIs in programs; their code produces views or intermediate evidence that the model reads to answer, whereas in Code-with-Image the code computes the answer itself, making the solution strategy the object of reflection (§3.1). This extends program-of-thoughts (PoT) and program-aided (PAL) prompting (Chen et al., 2022; Gao et al., 2023) from text inputs to quantitative reasoning over image pixels. Recent evaluations examine gains attributable to tool use (Guo et al., 2026a) and robustness to image perturbations (Guo et al., 2026b).

Benchmarks for agentic visual reasoning. General-purpose suites (MMMU (Yue et al., 2024), MMStar (Chen et al., 2024a), MathVista (Lu et al., 2024), MathVerse (Zhang et al., 2024)) assess broad visual understanding; V*Bench (Wu & Xie, 2024) and HR-Bench (Wang et al., 2025) emphasize visual search and high-resolution perception. VisualToolBench (Guo et al., 2025), Agentic-MME (Wei et al., 2026), AgentVista (Su et al., 2026b), VTC-Bench (Zhu et al., 2026), and TIR-Bench (Li et al., 2026b) evaluate interactive reasoning and tool use. CwI-Bench combines computational visual tasks, constructively verified targets, and disjoint learning splits to evaluate initial solving and subsequent improvement on new instances.

Reflection and skill refinement. Reflexion (Shinn et al., 2023) and ExpeL (Zhao et al., 2024) distill experience for language reasoning and agent tasks; Self-Debug (Chen et al., 2024b) and Voyager

(Wang et al., 2023) use execution results for program repair and reusable skills. TextGrad (Yuksekgonul et al., 2024) uses textual feedback as gradients to optimize prompts. GEPA (Agrawal et al., 2026), EvoSkill (Alzubi et al., 2026), SkillOpt (Yang et al., 2026; Shen et al., 2026), and HDSO (Shang & Yang, 2026) optimize prompts or skills from trajectories and evaluation results, while SPyCE (Zhang & Qiu, 2026) refines visual skills alongside an RL-trained policy. Our work studies reflection and skill refinement for computational visual tasks, where solving requires algorithms that compute answers from image data and reflection refines reusable solution procedures.

3 Code-with-Image Task Regime and Benchmark

3.1 From thinking with images to Code-with-Image

Thinking with images (Su et al., 2025) answers a visual question by calling tools to crop, zoom, sketch, or adjust the input image (Wu & Xie, 2024; Zheng et al., 2026; Su et al., 2026a; Hu et al., 2024; Zhang et al., 2025; Zhao et al., 2025) and reasoning over the returned views. The LVLM infers the answer from these observations, an approach suited to questions whose difficulty lies in locating relevant evidence. A quantitative target requires explicit measurement: overlaying two views of a scene reveals *that* they are displaced, but does not directly quantify *by how much*.

Code-with-Image shifts from exposing visual evidence to computing the target. The image serves as data for an algorithm that the LVLM designs and executes as code in a general-purpose interpreter: establishing correspondences between views and aggregating their offsets yields the displacement through a sequence of dependent operations. The LVLM selects the operations and checks outputs; code performs the computations. This shifts the bottleneck: success depends less on visual perception than on whether the LVLM devises a suitable algorithm and orders its steps correctly.

The following equations contrast how the two approaches produce an answer. Let M denote the LVLM, I the input image(s), and h_0 the initial history containing the images and task prompt; $M(\cdot \mid h)$ is the LVLM's output distribution given history h. At step k, action a_k invokes image operation T_{a_k}, while program p_k runs in the stateful interpreter E. Each returns output o_k; $\oplus$ appends the call and its output to form history h_k. Both approaches terminate after K steps with answer $\hat{y}$, obtained through visual inference or directly from program execution:

$$\begin{aligned} &\text{thinking with images:} \quad a_k \sim M(\cdot \mid h_{k-1}),\ o_k = T_{a_k}(I), \quad h_k = h_{k-1} \oplus (a_k, o_k),\ \hat{y} \sim M(\cdot \mid h_K); \\ &\text{Code-with-Image:} \quad p_k \sim M(\cdot \mid h_{k-1}),\ o_k = E(p_k; I),\ h_k = h_{k-1} \oplus (p_k, o_k),\ \hat{y} = o_K. \end{aligned} \tag{1}$$

The distinction lies in what execution contributes to the final answer. In thinking with images, the LVLM infers the answer from the observations returned by the tools. In Code-with-Image, the program computes the answer directly, with o_K denoting the resulting target value. The challenge is therefore to design and implement an algorithm that computes the target from image pixels.

Existing thinking-with-images benchmarks assess how well a model uses tools to obtain visual observations (Table 10 in Appendix G.1). CwI-Bench differs in three ways. First, targets such as displacements and transformation parameters require computation, testing whether the LVLM can design and implement an algorithm that computes the answer from image pixels. Second, each task provides a generator that constructs images from known target values and checks that these targets can be recovered from the generated images. This enables new instances with verified ground-truth answers to be generated as needed. Third, each task provides disjoint training, validation, and test instances. Together, these properties support evaluation of reflection: a solution procedure can be represented as a reusable skill, its failures diagnosed using ground-truth answers and execution records, and the revised skill evaluated on unseen instances.

3.2 Benchmark construction

Each of the 30 tasks asks the solver to compute the target from image pixels using a general-purpose interpreter, without a task-specific API or reference toolchain. The four task families cover distinct computational challenges (Figure 1): *transformation and rearrangement* tasks recover hidden transformations or permutations; *perception and counting* tasks extract visual structure through techniques such as filtering and segmentation; *localization* tasks estimate coordinates; and *color* tasks recover color values and mixing relationships. Tight scoring tolerances and composite answers make reliable estimation through visual inspection difficult. Task definitions, tolerances, and

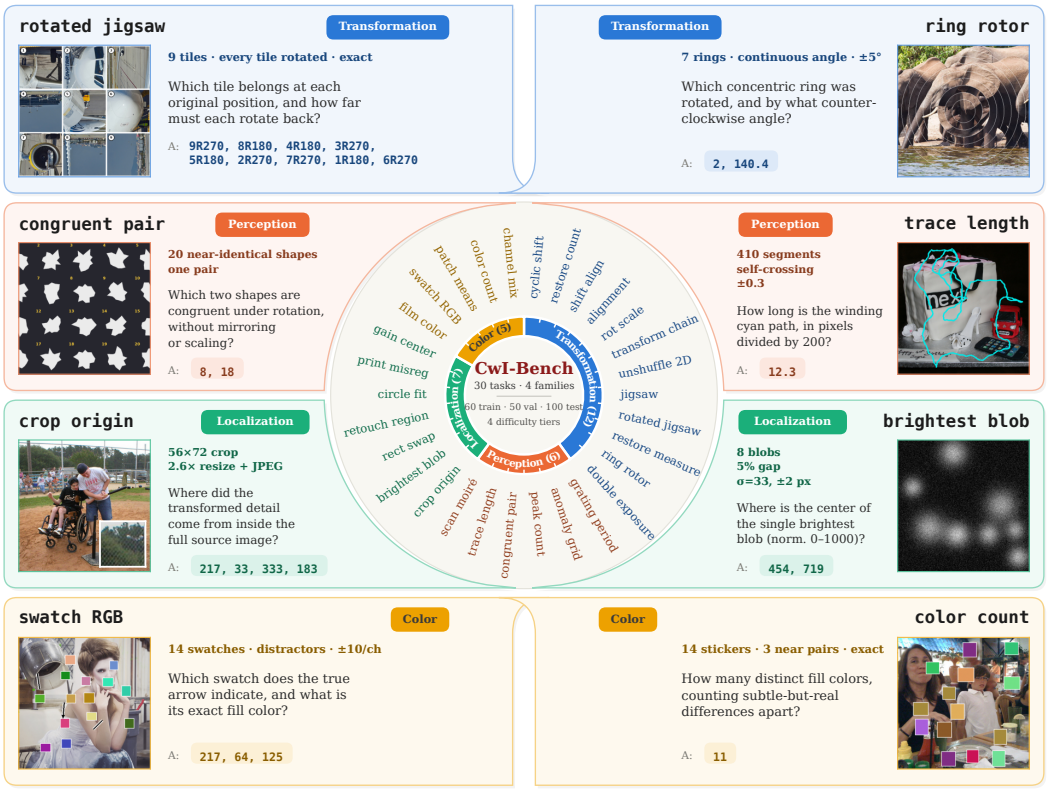


Figure 1: **Benchmark overview.** Rows show representative tasks from the four families, each with its input images, question, and constructive answer. The wheel groups all 30 tasks by family; its hub gives the shared protocol: 60 training, 50 validation, and 100 test cases per task.

examples appear in Appendix C. Generators construct images from known targets and apply task-specific uniqueness, separability, and independent recovery checks to the saved images (details in Appendix G). Each task provides 60 training, 50 validation, and 100 test instances, totaling 6,300 overall. Training photographs come from COCO (Lin et al., 2014) `train2017`, while test photographs come exclusively from `val2017`; an automated audit checks that source photographs do not overlap across splits. New labeled instances can be generated using this procedure. Training instances and their gold answers support reflection, while validation guides skill selection and revision. The same independent test set evaluates reasoning without code, bare Code-with-Image, and solving with the selected skills, which remain fixed during test evaluation.

4 Reflection-Driven Refinement

Reflection-Driven Refinement refines reusable solution strategies for a frozen LVLM by updating a skill library through three reflection mechanisms (Figure 2). Observational reflection diagnoses failures from recorded attempts. Executable reflection runs code to obtain new evidence, investigate failure causes, and develop and test repairs. Cross-round reflection reviews past updates and outcomes to identify recurring failures, reassess earlier assumptions, and revise the overall procedure.

4.1 Skills and Learning Objective

Bare Code-with-Image provides an interpreter, but the LVLM must still select an algorithm, interpret task conventions, and implement the computation correctly. A *skill* is a written solution strategy for one task (Shang & Yang, 2026; Yang et al., 2026; Li et al., 2026a): when it applies, which computational steps to follow, and how to verify intermediate results. The LVLM develops it from training

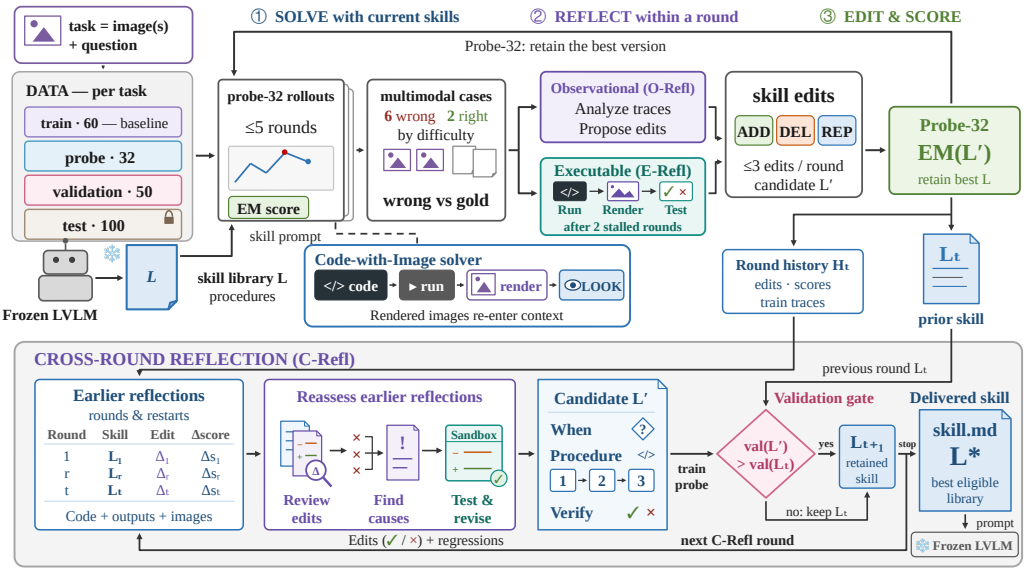


Figure 2: **Overview.** A frozen LVLM solves visual tasks with code. Observational reflection diagnoses errors from recorded trajectories, executable reflection tests these diagnoses and repairs in a sandbox, and cross-round reflection summarizes the revision history, analyzes dominant errors, and reconstructs the procedure. Validated skills guide later solves via the model's prompt.

attempts, and the frozen solver uses it when solving further instances of the same task, including unseen test instances. A skill records the chosen algorithm, its assumptions, and verification steps, and may include code. A *library* L collects the skills for one task.

Let T denote a task and $\mathcal{D}_T$ its distribution of labeled instances (x, y), where x contains the image(s) and prompt, and y is the constructively verified answer. Let M be the frozen LVLM, E its stateful Python interpreter, and $\mathcal{A}_E^M$ the resulting Code-with-Image solver (§3.1). We combine the skill library L and task input x to form the initial history $h_0 = L \oplus x$ in equation 1. The solver samples a trajectory $\tau = \big((p_k, o_k)\big)_{k=1}^{K} \sim \mathcal{A}_E^M(\cdot \mid L \oplus x)$, where K is the number of execution steps, p_k the program at step k, and o_k its output. Intermediate outputs may include text and rendered images; the final output o_K is the computed answer, denoted by $\hat{y}(\tau)$. Bare Code-with-Image uses $L = \varnothing$. Let $s_T(\hat{y}, y) \in \{0, 1\}$ indicate whether a prediction matches the ground-truth answer under the task's scoring tolerance. The learning objective is to maximize expected task accuracy:

$$J_T(L) \;:=\; \mathbb{E}_{(x,y)\sim\mathcal{D}_T,\ \tau\sim\mathcal{A}_E^M(\cdot\mid L\oplus x)}\big[\, s_T\big(\hat{y}(\tau), y\big)\,\big], \qquad L^\star \;=\; \arg\max_{L}\; J_T(L). \tag{2}$$

Here $J_T(L)$ is the expected accuracy over task instances and model sampling, and $L^\star$ denotes an optimal library. We search over skill libraries using training trajectories, ground-truth answers, and execution outputs; the model's weights remain fixed.

4.2 Reflection Mechanisms

The three mechanisms use the same frozen LVLM and training and validation splits to update the skill library, but differ in the evidence available for reflection and the scope of skill edits. Let t index reflection rounds and k execution steps. At round t, let L_t be the current library, B_t a batch of training trajectories with their inputs and ground-truth answers, and H_t a summary of earlier skill updates and evaluations. The mode $m_t \in \{\text{O-Refl}, \text{E-Refl}, \text{C-Refl}\}$ selects the reflection mechanism. Its procedure R_{m_t} proposes edits Δ_t:

$$\Delta_t = R_{m_t}(L_t; B_t, H_t), \qquad \widetilde{L}_{t+1} = \mathrm{Apply}(L_t, \Delta_t). \tag{3}$$

The operator Apply adds, replaces, or deletes skill entries under the selected mode's constraints, producing candidate library $\widetilde{L}_{t+1}$. The history H_t summarizes recent scores and skill entries for O-Refl and E-Refl, and provides the record of library versions, edits, and their outcomes for C-Refl.

Observational reflection (O-Refl). O-Refl diagnoses failures from recorded training attempts. The LVLM examines the current library and a batch of multimodal trajectories (prompts, code, outputs, and rendered images) with their ground-truth answers. Failed attempts are sampled across difficulty tiers and paired with successful attempts from the same task. Comparing them can reveal incorrect computations, missing checks, or instructions that conflict with successful strategies. In one call, the LVLM uses this diagnosis to propose skill edits for future instances.

The diagnosis is limited by the evidence available in these records. An incorrect answer may have several possible causes, and the relevant intermediate values may not have been printed or rendered. Further investigation may therefore require additional evidence from code execution.

Executable reflection (E-Refl). E-Refl runs code to obtain additional evidence, investigate failure causes, and develop and test repairs. It can inspect intermediate values from the original code and measure image properties not explicitly reported in the recorded outputs. A separate stateful sandbox E' is preloaded with the cases in B_t: their images, ground-truth and predicted answers, and original code. Starting from $\mathcal{E}_0 = (B_t, H_t)$, at investigation step j the LVLM writes diagnostic code q_j, receives the output u_j (text and rendered images), and appends both to the investigation history:

$$q_j \sim M(\cdot \mid L_t, \mathcal{E}_{j-1}), \quad u_j = E'(q_j), \quad \mathcal{E}_j = \mathcal{E}_{j-1} \oplus (q_j, u_j). \tag{4}$$

The LVLM uses the evidence gathered during this investigation to propose skill edits Δ_t. These edits draw on the observed causes of failure and the behavior of candidate repairs on the same cases, translating the experimental findings into instructions for future task instances.

Cross-round reflection (C-Refl). Successive local edits may retain early assumptions and add exceptions without correcting the underlying strategy. A short score history may not provide enough evidence to identify which assumptions need to change. C-Refl therefore reviews the full record from O-Refl and E-Refl: library versions, edits introducing or changing each rule and subsequent score changes, and the solver's execution records from all earlier rounds. It examines which rule changes were followed by gains or regressions. It then groups remaining training failures by type and inspects representative cases of the main types to distinguish strategy flaws from failures to follow useful rules. It also identifies effective steps in successful attempts and tests proposed changes in the same sandbox. Based on this evidence, C-Refl can remove poorly supported rules, combine complementary steps, or rewrite the library as a single ordered procedure. This broader scope allows it to revisit assumptions retained through earlier local edits. Accepted and rejected candidates are recorded with their outcomes to guide subsequent diagnoses and skill revisions.

Reflection loop. The loop starts from an empty skill library and the bare solver's training trajectories. O-Refl proposes edits from recorded attempts. Candidates are scored on a fixed training subset (the probe), and E-Refl is invoked when progress stalls. Search continues from the latest candidate, retaining the best library separately. C-Refl can use the accumulated search history to reconsider earlier choices and revise the overall procedure. It starts from the selected library, or from an empty one when earlier runs provide useful experience but no qualifying library. The final library is selected by validation accuracy. Search, selection, and budget details are given in Appendix G.4, with full pseudocode for the three reflection mechanisms in Appendix F.

5 Experiments

5.1 Experimental setup

We evaluate chain-of-thought reasoning with thinking disabled/enabled (Instruct/Thinking), bare Code-with-Image, and three thinking-with-images agents on the same test instances (Table 1). Scores are recomputed from archived trajectories using a shared exact-match scorer with task-specific tolerances. On four LVLMs (hereafter 9B, gemma-26B, 27B, and luna), we compare the full pipeline with three skill optimizers, TextGrad (Yuksekgonul et al., 2024), GEPA (Agrawal et al., 2026), and SkillOpt (Yang et al., 2026) (Table 2), and ablate its reflection mechanisms (Table 3). The optimizers update task-specific skill libraries for the same frozen solver on the same training and validation instances, with optimization budgets matched to our pipeline by the total number of LVLM inference calls during skill optimization. For each method, the open LVLMs use three seeds

Table 1: **Benchmark evaluation.** Mean test accuracy by family, all 30 tasks. Agents (Zheng et al., 2026; Hong et al., 2026; Zhang et al., 2025) use native protocols. **Bold**: best per model and column; subscripts: gain over Thinking. Per-task results for the four models with reflection: Appendix B; shared protocols and model-specific settings: Appendix A.

Model	Method	Task family				Overall
		Transformation	Perception	Localization	Color	
DeepEyes-7B	Think w/ image	1.8	3.8	0.1	2.4	1.9
Thyme-RL	Think w/ image	1.5	4.3	0.6	2.0	1.9
DeepEyesV2-RL	Think w/ image	1.3	4.3	1.1	3.0	2.2
Qwen3.5-9B	Instruct	4.9	13.8	6.0	6.4	7.2
	Thinking	**9.5**	**19.3**	7.3	7.4	10.6
	Code-with-Image	8.9$_{-0.6}$	16.7$_{-2.6}$	**8.6**$_{+1.3}$	**13.2**$_{+5.8}$	**11.1**$_{+0.5}$
Qwen3.5-35B-A3B	Instruct	6.3	19.5	5.6	9.2	9.3
	Thinking	10.4	24.5	7.6	10.0	12.5
	Code-with-Image	**22.6**$_{+12.2}$	**35.5**$_{+11.0}$	**18.9**$_{+11.3}$	**24.6**$_{+14.6}$	**24.6**$_{+12.1}$
Qwen3.5-27B	Instruct	6.1	18.3	4.6	8.8	8.6
	Thinking	12.6	24.8	11.6	10.2	14.4
	Code-with-Image	**27.9**$_{+15.3}$	**47.0**$_{+22.2}$	**28.3**$_{+16.7}$	**32.4**$_{+22.2}$	**32.6**$_{+18.2}$
Qwen3.5-122B-A10B	Instruct	6.2	19.2	6.1	9.4	9.3
	Thinking	11.6	23.3	10.3	13.4	13.9
	Code-with-Image	**24.7**$_{+13.1}$	**45.0**$_{+21.7}$	**35.0**$_{+24.7}$	**33.6**$_{+20.2}$	**32.7**$_{+18.8}$
gemma-4-12B	Instruct	4.5	17.3	12.7	7.2	9.4
	Thinking	6.1	**20.2**	12.4	7.6	10.6
	Code-with-Image	**6.7**$_{+0.6}$	16.5$_{-3.7}$	**17.9**$_{+5.5}$	**27.2**$_{+19.6}$	**14.7**$_{+4.1}$
gemma-4-26B	Instruct	6.8	17.5	11.4	11.4	10.8
	Thinking	8.2	17.2	14.9	8.0	11.5
	Code-with-Image	**16.8**$_{+8.6}$	**40.5**$_{+23.3}$	**33.9**$_{+19.0}$	**41.2**$_{+33.2}$	**29.6**$_{+18.1}$
MiniMax-M3	Instruct	3.3	15.7	1.0	5.6	5.6
	Thinking	4.4	16.0	2.0	7.6	6.7
	Code-with-Image	**17.4**$_{+13.0}$	**35.2**$_{+19.2}$	**23.6**$_{+21.6}$	**33.6**$_{+26.0}$	**25.1**$_{+18.4}$
DeepSeek-V4-Flash-Vision	Instruct	4.0	15.5	5.0	4.6	6.6
	Thinking	7.7	19.8	3.0	8.8	9.2
	Code-with-Image	**18.2**$_{+10.5}$	**39.8**$_{+20.0}$	**35.4**$_{+32.4}$	**39.6**$_{+30.8}$	**30.1**$_{+20.9}$
GPT-5.6-luna	Instruct	11.9	16.2	10.3	15.0	12.9
	Thinking	29.8	30.7	16.4	22.0	25.6
	Code-with-Image	**35.4**$_{+5.6}$	**36.2**$_{+5.5}$	**41.7**$_{+25.3}$	**61.2**$_{+39.2}$	**41.3**$_{+15.7}$

per task, and the qualifying library with the highest validation accuracy is selected. If no library qualifies, the solver uses bare Code-with-Image. Test instances are reserved for final evaluation. Detailed model settings, search budgets, and selection criteria are in Appendices A and G.

5.2 Results

Code-with-Image versus chain-of-thought. Table 1 makes three points. First, the three thinking-with-images agents score 1.9% to 2.2%: their crops, zooms, and searches aid visual inspection, but these tasks require algorithms that compute answers directly from image pixels. Second, answering directly is not enough: even GPT-5.6-luna, the strongest model, stays below 30% with chain-of-thought (25.6% with thinking enabled), and the other eight LVLMs stay below 15%. Third, computing the answer raises accuracy directly: bare Code-with-Image exceeds chain-of-thought on all nine LVLMs, from 25.6% to 41.3% on GPT-5.6-luna and from 14.4% to 32.6% on Qwen3.5-27B, with gains of 12 to 21 points on seven of them; the two smallest models gain the least, since the benefit depends on devising a correct algorithm, which is challenging for weak models.

Strategy refinement. The full pipeline reaches 64.6% on 27B and 70.1% on luna (Table 2), improving on bare Code-with-Image by 24.2 to 32.0 points across the four LVLMs. With its refined library, the 27B solver surpasses bare luna (41.3%) without any weight update. Under the same optimization budget, the full pipeline outperforms the best-performing baseline in overall accuracy for each LVLM, with margins ranging from 3.4 points on 9B and gemma-26B to 6.2 on luna. Even with about 1.6× that budget, SkillOpt remains 1.8 and 1.6 points behind on 9B and 27B, respectively. Appendix E.3 presents a case study illustrating all three reflection mechanisms.

Table 2: **Strategy refinement results** (test accuracy, all 30 tasks). Ours is the full pipeline (O-Refl, E-Refl, C-Refl); every baseline is allocated the same optimization budget (1.6× ours where marked), and skill rows select among seeds on validation. Code-with-Image: bare solver without skills. Subscripts: gain over bare Code-with-Image; **bold**: best per model and column.

Model	Method	Task family				Overall
		Transformation	Perception	Localization	Color	
Qwen3.5-9B	Code-with-Image	8.9	16.7	8.6	13.2	11.1
	+ TextGrad	27.9$_{+19.0}$	27.7$_{+11.0}$	27.3$_{+18.7}$	26.2$_{+13.0}$	27.4$_{+16.3}$
	+ GEPA	33.4$_{+24.5}$	32.3$_{+15.6}$	44.3$_{+35.7}$	26.8$_{+13.6}$	34.6$_{+23.5}$
	+ SkillOpt	25.2$_{+16.3}$	39.5$_{+22.8}$	40.0$_{+31.4}$	21.6$_{+8.4}$	30.9$_{+19.8}$
	+ SkillOpt (1.6× budget)	28.3$_{+19.4}$	**42.0**$_{+25.3}$	**47.3**$_{+38.7}$	**32.8**$_{+19.6}$	36.2$_{+25.1}$
	Ours	**37.5**$_{+28.6}$	40.0$_{+23.3}$	44.0$_{+35.4}$	28.2$_{+15.0}$	**38.0**$_{+26.9}$
gemma-4-26B	Code-with-Image	16.8	40.5	33.9	41.2	29.6
	+ TextGrad	39.7$_{+22.9}$	61.3$_{+20.8}$	55.6$_{+21.7}$	43.6$_{+2.4}$	48.4$_{+18.8}$
	+ GEPA	37.1$_{+20.3}$	**67.7**$_{+27.2}$	60.0$_{+26.1}$	48.0$_{+6.8}$	50.4$_{+20.8}$
	+ SkillOpt	38.2$_{+21.4}$	62.7$_{+22.2}$	56.1$_{+22.2}$	**54.2**$_{+13.0}$	49.9$_{+20.3}$
	Ours	**41.0**$_{+24.2}$	65.2$_{+24.7}$	**68.0**$_{+34.1}$	51.2$_{+10.0}$	**53.8**$_{+24.2}$
Qwen3.5-27B	Code-with-Image	27.9	47.0	28.3	32.4	32.6
	+ TextGrad	50.6$_{+22.7}$	75.3$_{+28.3}$	63.0$_{+34.7}$	37.8$_{+5.4}$	56.3$_{+23.7}$
	+ GEPA	43.8$_{+15.9}$	78.2$_{+31.2}$	**68.6**$_{+40.3}$	**47.4**$_{+15.0}$	57.1$_{+24.5}$
	+ SkillOpt	52.4$_{+24.5}$	83.7$_{+36.7}$	65.3$_{+37.0}$	46.4$_{+14.0}$	60.7$_{+28.1}$
	+ SkillOpt (1.6× budget)	55.8$_{+27.9}$	87.7$_{+40.7}$	66.1$_{+37.8}$	46.4$_{+14.0}$	63.0$_{+30.4}$
	Ours	**57.3**$_{+29.4}$	**91.8**$_{+44.8}$	66.0$_{+37.7}$	47.2$_{+14.8}$	**64.6**$_{+32.0}$
GPT-5.6-luna	Code-with-Image	35.4	36.2	41.7	61.2	41.3
	+ TextGrad	53.2$_{+17.8}$	68.3$_{+32.1}$	57.1$_{+15.4}$	75.8$_{+14.6}$	60.9$_{+19.6}$
	+ GEPA	58.1$_{+22.7}$	69.2$_{+33.0}$	61.4$_{+19.7}$	75.0$_{+13.8}$	63.9$_{+22.6}$
	+ SkillOpt	52.3$_{+16.9}$	55.0$_{+18.8}$	67.7$_{+26.0}$	76.0$_{+14.8}$	60.4$_{+19.1}$
	Ours	**63.6**$_{+28.2}$	**74.5**$_{+38.3}$	**69.3**$_{+27.6}$	**81.6**$_{+20.4}$	**70.1**$_{+28.8}$

Table 3: **Where the gain comes from** (test points, all 30 tasks). The gain of Ours over bare Code-with-Image (Total) and the points added in turn by observational reflection (+ O-Refl), executable reflection (+ E-Refl), and cross-round reflection (+ C-Refl).

		Transf.	Percep.	Local.	Color	**Overall**			Transf.	Percep.	Local.	Color	**Overall**
Qwen3.5-9B	+ O-Refl	+10.3	+13.5	+10.5	+9.2	+10.8	Qwen3.5-27B	+ O-Refl	+12.7	+24.0	+28.6	+0.0	+16.5
	+ E-Refl	+10.9	+2.1	+18.0	−2.0	+8.7		+ E-Refl	+13.4	−7.2	+9.1	+4.2	+6.8
	+ C-Refl	+7.4	+7.7	+6.9	+7.8	+7.4		+ C-Refl	+3.3	+28.0	+0.0	+10.6	+8.7
	Total	**+28.6**	**+23.3**	**+35.4**	**+15.0**	**+26.9**		**Total**	**+29.4**	**+44.8**	**+37.7**	**+14.8**	**+32.0**
gemma-4-26B	+ O-Refl	+11.3	+14.8	+20.4	+0.0	+12.2	GPT-5.6-luna	+ O-Refl	+17.9	+26.3	+20.2	+15.0	+19.7
	+ E-Refl	+5.0	+9.7	+7.3	+1.6	+5.9		+ E-Refl	+3.0	+8.8	+0.2	+3.6	+3.6
	+ C-Refl	+7.9	+0.2	+6.4	+8.4	+6.1		+ C-Refl	+7.3	+3.2	+7.2	+1.8	+5.5
	Total	**+24.2**	**+24.7**	**+34.1**	**+10.0**	**+24.2**		**Total**	**+28.2**	**+38.3**	**+27.6**	**+20.4**	**+28.8**

Ablation. Table 3 compares the gains from successive reflection configurations, with additional validation results in Appendix G.7. Of the 24.2 to 32.0 points gained in test accuracy over bare Code-with-Image, observational reflection alone contributes 10.8 to 19.7; adding executable and cross-round reflection yields further gains of 3.6 to 8.7 and 5.5 to 8.7 points, respectively. With O-Refl and E-Refl alone, 83–92% of seeds on the open LVLMs stop early because the best probe score has not improved for five consecutive rounds. A higher round limit would not extend these runs. Additional seeds yield diminishing returns, with the third adding 1.8–2.9 validation points. By comparison, C-Refl adds 6.5–8.5 points over the selected libraries at roughly the cost of one more seed, about a quarter of the pipeline's optimization budget. Full results are in Appendix G.6.

Reflection quality. Figure 3 compares reflection performance across LVLMs. Panel (a) shows normalized validation gains over reflection rounds (metric defined in Appendix G.5). Across the eight rounds shown, luna captures about half of the remaining gap to perfect accuracy, with its curve rising mainly within the first four rounds; 27B reaches 0.37, with its steepest rise between rounds 5 and 6, while 9B and gemma-26B remain below a quarter. Panel (b) fixes the solver to 9B and varies

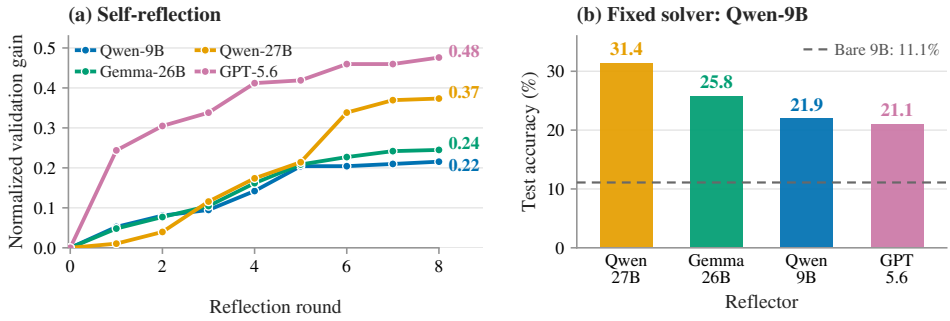


Figure 3: **Reflection performance. (a)** Normalized validation gains over reflection rounds (30-task mean). **(b)** Test accuracy of Qwen3.5-9B with different reflectors (one seed of O-Refl and E-Refl per task); dashed line: bare Code-with-Image on the same 9B solver.

Table 4: **Skill transfer** (test accuracy, all 30 tasks). Rows: solvers; columns: skill authors; shaded diagonal: same solver and skill author. Missing skills fall back to the recipient's bare Code-with-Image. Subscripts: gain over bare Code-with-Image. **Bold**: best library per solver.

Solver	Bare Code-with-Image	Runs the skill library written by			
		Qwen3.5-9B	gemma-4-26B	Qwen3.5-27B	GPT-5.6-luna
Qwen3.5-9B	11.1	38.0$_{+26.9}$	34.8$_{+23.7}$	**47.4**$_{+36.3}$	22.7$_{+11.6}$
gemma-4-26B	29.6	43.0$_{+13.4}$	53.8$_{+24.2}$	**57.0**$_{+27.4}$	47.9$_{+18.3}$
Qwen3.5-27B	32.6	44.7$_{+12.1}$	52.1$_{+19.5}$	**64.6**$_{+32.0}$	45.2$_{+12.6}$
GPT-5.6-luna	41.3	49.0$_{+7.7}$	55.4$_{+14.1}$	66.0$_{+24.7}$	**70.1**$_{+28.8}$

the reflector. Using 27B and gemma-26B as reflectors yields test accuracies of 31.4% and 25.8%, respectively, compared with 21.9% for self-reflection and 11.1% for bare Code-with-Image; luna as reflector yields 21.1%. In the separate skill-transfer experiment, using 27B's full-pipeline library brings 9B to 47.4% (Table 4), below 27B's own 64.6%. These results suggest that gains depend on both the reflector and the solver's ability to apply the resulting skills.

Skill transfer. The refined skills transfer across LVLMs, and the pattern of transfer shows what a library carries. The 27B library is the best choice for every open solver: it raises 9B to 47.4% and gemma-26B to 57.0%, above their own refined libraries (38.0% and 53.8%; Table 4). The accuracy it yields rises with the recipient's strength, so the recipient's ability to carry out the written procedure bounds the gain. Luna-written skills show the other side of this bound: they improve every open LVLM over bare Code-with-Image yet trail each recipient's own refined library, because they are written at luna's level of execution and a weaker solver cannot carry them out. Conversely, 27B-written skills bring luna to 66.0%, below its own 70.1%: they are written for what 27B can execute and leave luna's stronger execution unused. A library thus encodes strategy at its author's level of execution and transfers best to solvers that can execute that level.

6 Conclusion

We presented Code-with-Image, in which a model answers a visual question by designing an algorithm that computes the answer from the pixels and executing it, and CwI-Bench, whose generated tasks with disjoint training, validation, and test instances make improvement through reflection measurable on unseen instances. Bare Code-with-Image improves overall accuracy over chain-of-thought on all nine evaluated backbones. Reflection-Driven Refinement raises it further with frozen weights, from 32.6% to 64.6% on Qwen3.5-27B and from 41.3% to 70.1% on GPT-5.6-luna, ahead of three skill optimizers given the same budget, and the refined skills transfer across model scales and families. With its generators and learning splits, CwI-Bench provides a testbed for studying how models construct and improve computational solutions to visual problems.

AI USE STATEMENT

In this work, we used generative AI tools to assist with code implementation following author-specified methods and experimental protocols, help design benchmark tasks and implement the procedural generators that produce the synthetic datasets, and provide feedback on experimental results to support their interpretation. We have not used generative AI tools to design the proposed methods or experimental protocols, and providing key ingredients for mathematical proofs and writing proofs are not applicable to this work. Additionally, we used generative AI tools to improve the clarity and readability of the manuscript. We have reviewed all AI-assisted work: we checked and tested the code, verified AI-assisted interpretations of results against experimental records, and reviewed and revised the manuscript. We take responsibility for the final content of this work, including text, claims, or artifacts produced with the aid of generative AI.

REFERENCES

Lakshya A Agrawal, Shangyin Tan, Dilara Soylu, Noah Ziems, Rishi Khare, Krista Opsahl-Ong, Arnav Singhvi, Herumb Shandilya, Michael J Ryan, Meng Jiang, et al. Gepa: Reflective prompt evolution can outperform reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 8479–8565, 2026.

Salaheddin Alzubi, Noah Provenzano, Jaydon Bingham, Weiyuan Chen, and Tu Vu. Evoskill: Automated skill discovery for multi-agent systems. *arXiv preprint arXiv:2603.02766*, 2026.

Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Yuhang Zang, Zehui Chen, Haodong Duan, Jiaqi Wang, Yu Qiao, Dahua Lin, et al. Are we on the right way for evaluating large vision-language models? *Advances in Neural Information Processing Systems*, 37:27056–27087, 2024a.

Wenhu Chen, Xueguang Ma, Xinyi Wang, and William W Cohen. Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks. *arXiv preprint arXiv:2211.12588*, 2022.

Xinyun Chen, Maxwell Lin, Nathanael Schärli, and Denny Zhou. Teaching large language models to self-debug. In *International Conference on Learning Representations*, volume 2024, pp. 8746–8825, 2024b.

Luyu Gao, Aman Madaan, Shuyan Zhou, Uri Alon, Pengfei Liu, Yiming Yang, Jamie Callan, and Graham Neubig. Pal: Program-aided language models. In *International conference on machine learning*, pp. 10764–10799. PMLR, 2023.

Jiawei Guo, Donglei Yu, Yu Chen, Xiang Wang, Shuai Li, Xinpei Zhao, Huaxing Liu, Qinghao Wang, and Minpeng Liao. Do multimodal agents really benefit from tool use? a systematic study of capability gains. *arXiv preprint arXiv:2606.02357*, 2026a.

Xingang Guo, Utkarsh Tyagi, Advait Gosai, Paula Vergara, Jayeon Park, Ernesto Gabriel Hernández Montoya, Chen Bo Calvin Zhang, Bin Hu, Yunzhong He, Bing Liu, et al. Beyond seeing: Evaluating multimodal llms on tool-enabled image perception, transformation, and reasoning. *arXiv preprint arXiv:2510.12712*, 2025.

Zirun Guo, Minjie Hong, Feng Zhang, Kai Jia, and Tao Jin. Thinking with programming vision: Towards a unified view for thinking with images. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 33467–33476, 2026b.

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *2023 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 14953–14962. IEEE, 2023.

Jack Hong, Chenxiao Zhao, ChengLin Zhu, Weiheng Lu, Guohai Xu, and Xing Yu. Deepeyesv2: Toward agentic multimodal model, 2026. URL https://arxiv.org/abs/2511.05271.

Xinhai Hou, Shaoyuan Xu, Manan Biyani, Moyan Li, Jia Liu, Todd C Hollon, and Bryan Wang. Codev: Code with images for faithful visual reasoning via tool-aware policy optimization. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 21500–21510, 2026.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379, 2024.

Jiaxi Li, Ke Deng, Yun Wang, Jingyuan Huang, Yucheng Shi, Qiaoyu Tan, Jin Lu, and Ninghao Liu. Online skill learning for web agents via state-grounded dynamic retrieval. *arXiv preprint arXiv:2606.04391*, 2026a.

Ming Li, Jike Zhong, Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Yuxiang Lai, Chen Wei, Konstantinos Psounis, and Kaipeng Zhang. Tir-bench: A comprehensive benchmark for agentic thinking-with-images reasoning. In *European Conference on Computer Vision*, pp. 464–481. Springer, 2026b.

Tsung-Yi Lin, Michael Maire, Serge Belongie, James Hays, Pietro Perona, Deva Ramanan, Piotr Dollár, and C Lawrence Zitnick. Microsoft coco: Common objects in context. In *European conference on computer vision*, pp. 740–755. Springer, 2014.

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*, volume 2024, pp. 23439–23554, 2024.

OpenAI. Gpt-5.6: Frontier intelligence that scales with your ambition, July 2026. URL https://openai.com/index/gpt-5-6/. Accessed: 2026-09-21.

Fangxin Shang and Yehui Yang. Hypothesis-driven skill optimization for llm agents. *arXiv preprint arXiv:2606.22330*, 2026.

Yifei Shen, Bo Li, and Xinjie Zhang. Skillopt-lite: Better and faster agent self-evolution via one line of vibe. *arXiv preprint arXiv:2607.03451*, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652, 2023.

Qi Song, Honglin Li, Yingchen Yu, Haoyi Zhou, Lin Yang, Song Bai, Qi She, Zilong Huang, and Yunqing Zhao. Codedance: A dynamic tool-integrated mllm for executable visual reasoning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 19186–19195, 2026.

Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel space reasoning via curiosity-driven reinforcement learning. *Advances in Neural Information Processing Systems*, 38:8222–8251, 2026a.

Zhaochen Su, Peng Xia, Hangyu Guo, Zhenhua Liu, Yan Ma, Xiaoye Qu, Jiaqi Liu, Yanshu Li, Kaide Zeng, Zhengyuan Yang, et al. Thinking with images for multimodal reasoning: Foundations, methods, and future frontiers. *arXiv preprint arXiv:2506.23918*, 2025.

Zhaochen Su, Jincheng Gao, Hangyu Guo, Zhenhua Liu, Lueyang Zhang, Xinyu Geng, Shijue Huang, Peng Xia, Guanyu Jiang, Cheng Wang, et al. Agentvista: Evaluating multimodal agents in ultra-challenging realistic visual scenarios. *arXiv preprint arXiv:2602.23166*, 2026b.

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *2023 IEEE/CVF International Conference on Computer Vision (ICCV)*, pp. 11854–11864. IEEE, 2023.

Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291*, 2023.

Wenbin Wang, Liang Ding, Minyan Zeng, Xiabin Zhou, Li Shen, Yong Luo, Wei Yu, and Dacheng Tao. Divide, conquer and combine: A training-free framework for high-resolution image perception in multimodal large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 39, pp. 7907–7915, 2025.

Qianshan Wei, Yishan Yang, Siyi Wang, Jinglin Chen, Binyu Wang, Jiaming Wang, Shuang Chen, Zechen Li, Yang Shi, Yuqi Tang, et al. Agentic-mme: What agentic capability really brings to multimodal intelligence? *arXiv preprint arXiv:2604.03016*, 2026.

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. In *2024 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pp. 13084–13094. IEEE, 2024.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, et al. Qwen3 technical report. *arXiv preprint arXiv:2505.09388*, 2025.

Yifan Yang, Ziyang Gong, Weiquan Huang, Qihao Yang, Ziwei Zhou, Zisu Huang, Yan Li, Xuemei Gao, Qi Dai, Bei Liu, et al. Skillopt: Executive strategy for self-evolving agent skills. *arXiv preprint arXiv:2605.23904*, 2026.

Xiang Yue, Yuansheng Ni, Kai Zhang, Tianyu Zheng, Ruoqi Liu, Ge Zhang, Samuel Stevens, Dongfu Jiang, Weiming Ren, Yuxuan Sun, et al. Mmmu: A massive multi-discipline multimodal understanding and reasoning benchmark for expert agi. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pp. 9556–9567, 2024.

Mert Yuksekgonul, Federico Bianchi, Joseph Boen, Sheng Liu, Zhi Huang, Carlos Guestrin, and James Zou. Textgrad: Automatic" differentiation" via text. *arXiv preprint arXiv:2406.07496*, 2024.

Renrui Zhang, Dongzhi Jiang, Yichi Zhang, Haokun Lin, Ziyu Guo, Pengshuo Qiu, Aojun Zhou, Pan Lu, Kai-Wei Chang, Yu Qiao, et al. Mathverse: Does your multi-modal llm truly see the diagrams in visual math problems? In *European Conference on Computer Vision*, pp. 169–186. Springer, 2024.

Ru Zhang and Weijie Qiu. Spyce: Skill-policy co-evolution for multimodal agents. *arXiv preprint arXiv:2607.13854*, 2026.

Yi-Fan Zhang, Xingyu Lu, Shukang Yin, Chaoyou Fu, Wei Chen, Xiao Hu, Bin Wen, Kaiyu Jiang, Changyi Liu, Tianke Zhang, et al. Thyme: Think beyond images. *arXiv preprint arXiv:2508.11630*, 2025.

Andrew Zhao, Daniel Huang, Quentin Xu, Matthieu Lin, Yong-Jin Liu, and Gao Huang. Expel: Llm agents are experiential learners. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 19632–19642, 2024.

Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Ming Li, Qilong Wu, Kaipeng Zhang, and Chen Wei. Pyvision: Agentic vision with dynamic tooling. *arXiv preprint arXiv:2507.07998*, 2025.

Shitian Zhao, Shaoheng Lin, Ming Li, Haoquan Zhang, Wenshuo Peng, Kaipeng Zhang, and Chen Wei. Pyvision-rl: Forging open agentic vision models via rl. *arXiv preprint arXiv:2602.20739*, 2026.

Ziwei Zheng, Minghao Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, and Chao Shen. Deepeyes: Incentivizing" thinking with images" via reinforcement learning. In *International Conference on Learning Representations*, volume 2026, pp. 126775–126798, 2026.

Xuanyu Zhu, Yuhao Dong, Rundong Wang, Yang Shi, Zhipeng Wu, Yinlun Peng, YiFan Zhang, Yihang Lou, Yuanxing Zhang, Ziwei Liu, et al. Vtc-bench: Evaluating agentic multimodal models via compositional visual tool chaining. *arXiv preprint arXiv:2603.15030*, 2026.

Appendix

Table of contents

A. Evaluation Configuration **14**

A.1 Shared protocol 14

A.2 Model settings 14

B. Per-Task Results for Four Models Evaluated with Reflection **15**

Qwen3.5-27B 17

GPT-5.6-luna 18

Qwen3.5-9B 15

gemma-4-26B 16

C. Task Cards and Worked Examples **19**

C.1 Transformation & rearrangement 19

C.2 Perception & counting 22

C.3 Localization 24

C.4 Color 26

D. Reflection Prompts (Verbatim) **28**

E. Exhibits: a Delivered Skill and a Reflection Session **30**

E.1 A delivered skill, verbatim excerpt 30

E.2 Executable-reflection session excerpts 31

E.3 One task through all three reflection modes 32

F. Full Pseudocode **35**

G. Benchmark and Protocol Details **36**

G.1 Comparison with visual reasoning benchmarks 36

G.2 Computational challenges and target verification 36

G.3 Effect of a larger round budget 37

G.4 Reflection loop: search, selection, and limits 37

G.5 Retrospective analysis of reflection rounds 38

G.6 Why cross-round reflection is needed 38

G.7 Ablating executable reflection: validation uplift 39

A EVALUATION CONFIGURATION

All arms are scored on the same held-out test items with the same exact-match scorer, and all Code-with-Image arms use the same sandbox. This section gives the shared protocol and the settings of every model in Table 1; only Qwen3.5-9B, Qwen3.5-27B, gemma-4-26B, and GPT-5.6-luna have refinement runs.

A.1 SHARED PROTOCOL

Every Code-with-Image arm runs the bounded act–execute–observe loop of §3.1: at most 5 rounds per episode, a forced final answer on the last round, and a 40,960-token episode budget. The sandbox is a persistent Jupyter kernel per episode with an isolated working directory; every execution returns stdout plus every rendered image. Split sizes, mode-specific budgets, and selection rules are specified in Appendix G, and skill formats in Appendix D. At solve time a delivered library is injected into the system prompt; nothing else changes.

A.2 MODEL SETTINGS

Qwen3.5 family (9B, 27B, 35B-A3B, 122B-A10B). Open weights, served locally (bfloat16, 40,960-token window) under one protocol. Code-with-Image transport is the native XML tool protocol: `<tool_call>` blocks carry the code, and tool responses return stdout and rendered images. Solver decoding is temperature 0.7, top-p 0.8, top-k 20, presence penalty 1.5, with thinking disabled; the Instruct CoT baseline uses the same decoding, and the Thinking baseline uses temperature 1.0, top-p 0.95 in the model's native thinking mode. Refinement on 9B and 27B uses 3 seeds.

gemma-4 family (12B, 26B-A4B). Open weights, served locally (bfloat16). Code-with-Image transport is the model's native function-calling chat template. All decoding (solver, Instruct, and Thinking with native think markers) follows the family's official generation config (temperature 1.0, top-p 0.95, top-k 64). 26B uses the 40,960-token window; 12B encodes images into more tokens and is served with a 65,536-token window so that the inputs fit, with the same 5 rounds. At test time, the gemma Code-with-Image system prompt carries a short note that the input images are preloaded as a Python variable, identical in the bare, skill, and skill-optimizer test arms.

GPT-5.6-luna. Closed weights, accessed through the provider API. Code-with-Image transport is chat-completions function tools with reasoning disabled inside the tool loop; sampling is the provider default. The Instruct baseline runs with reasoning off; the Thinking baseline enables the provider's reasoning mode. Refinement uses a single seed and 8 rounds with the same prompts, edit grammar, and selection rule. All methods, including the three skill optimizers, use this single seed on luna, so comparisons on this model stay matched. One seed and 8 rounds suffice here: luna's gains arrive mainly within its first four rounds (Figure 3).

MiniMax-M3 and DeepSeek-V4-Flash-Vision. Closed weights, accessed through each provider's OpenAI-compatible API and evaluated with the same API harness and prompts as GPT-5.6-luna; sampling is the provider default. Code-with-Image uses native function calling with reasoning disabled. MiniMax-M3 switches between disabled (Instruct, Code-with-Image) and adaptive (Thinking) reasoning and carries the same preloaded-image note as the gemma models; DeepSeek-V4-Flash-Vision sets its reasoning effort to `none` for Instruct and Code-with-Image and to `medium` for Thinking.

Thinking-with-images agents. DeepEyes-7B, Thyme-RL, and DeepEyesV2-RL run from their released checkpoints under their authors' native protocols, tools, prompt templates, and inference settings, on the same test items and scorer, with full-resolution inputs. All three agents answer 94–98% of items; their correct answers concentrate on tasks where inspection suffices.

B PER-TASK RESULTS FOR FOUR MODELS EVALUATED WITH REFLECTION

Tables 5–8 give per-task Instruct, Thinking, bare Code-with-Image, and Ours results for the four models evaluated with reflection in Tables 1 and 2, one model per page.

Table 5: **Qwen3.5-9B per-task test accuracy (%).** All 30 tasks, using skills refined through its own reflection (3 seeds of the O-Refl–E-Refl loop, followed by C-Refl); columns as in Table 7.

Task	Instruct	Thinking	Code-with-Image		Δ
			No skill	Ours	
Transformation & rearrangement					
cyclic shift	0.0	0.0	8.0	**84.0**	**+76.0**
restore count	11.0	21.0	6.0	**22.0**	**+16.0**
shift align	0.0	0.0	7.0	**92.0**	**+85.0**
alignment	0.0	0.0	12.0	**95.0**	**+83.0**
rot scale	6.0	15.0	17.0	**31.0**	**+14.0**
transform chain	23.0	50.0	41.0	**93.0**	**+52.0**
unshuffle 2D	6.0	13.0	2.0	1.0	−1.0
jigsaw	0.0	0.0	6.0	**22.0**	**+16.0**
rotated jigsaw	0.0	2.0	0.0	0.0	0.0
restore measure	10.0	11.0	6.0	**8.0**	**+2.0**
ring rotor	3.0	2.0	2.0	2.0	0.0
double exposure	0.0	0.0	0.0	0.0	0.0
Perception & counting					
grating period	12.0	5.0	4.0	**26.0**	**+22.0**
anomaly grid	35.0	53.0	60.0	**83.0**	**+23.0**
peak count	22.0	37.0	14.0	**61.0**	**+47.0**
congruent pair	8.0	12.0	16.0	**49.0**	**+33.0**
trace length	3.0	8.0	6.0	**21.0**	**+15.0**
scan moiré	3.0	1.0	0.0	0.0	0.0
Localization					
crop origin	8.0	10.0	10.0	**79.0**	**+69.0**
brightest blob	0.0	0.0	21.0	**79.0**	**+58.0**
rect swap	23.0	29.0	18.0	**88.0**	**+70.0**
retouch region	10.0	11.0	4.0	**22.0**	**+18.0**
circle fit	1.0	1.0	4.0	**19.0**	**+15.0**
print misreg	0.0	0.0	2.0	**20.0**	**+18.0**
gain center	0.0	0.0	1.0	1.0	0.0
Color					
film color	0.0	1.0	4.0	4.0	0.0
swatch RGB	4.0	5.0	20.0	20.0	0.0
patch means	1.0	1.0	16.0	**55.0**	**+39.0**
color count	27.0	29.0	26.0	**62.0**	**+36.0**
channel mix	0.0	1.0	0.0	0.0	0.0
Mean (30 tasks)	7.2	10.6	11.1	**38.0**	**+26.9**

Table 6: **gemma-4-26B per-task test accuracy (%).** All 30 tasks, using skills refined through its own reflection (3 seeds of the O-Refl–E-Refl loop, followed by C-Refl); columns as in Table 7.

Task	Instruct	Thinking	Code-with-Image		Δ
			No skill	Ours	
Transformation & rearrangement					
cyclic shift	0.0	0.0	23.0	**98.0**	**+75.0**
restore count	21.0	28.0	14.0	**24.0**	**+10.0**
shift align	0.0	0.0	46.0	**94.0**	**+48.0**
alignment	0.0	0.0	35.0	**89.0**	**+54.0**
rot scale	0.0	4.0	4.0	**20.0**	**+16.0**
transform chain	43.0	41.0	64.0	**97.0**	**+33.0**
unshuffle 2D	4.0	8.0	2.0	**6.0**	**+4.0**
jigsaw	0.0	0.0	0.0	0.0	0.0
rotated jigsaw	0.0	0.0	0.0	0.0	0.0
restore measure	14.0	17.0	11.0	**26.0**	**+15.0**
ring rotor	0.0	0.0	2.0	**21.0**	**+19.0**
double exposure	0.0	0.0	0.0	**17.0**	**+17.0**
Perception & counting					
grating period	5.0	12.0	73.0	**90.0**	**+17.0**
anomaly grid	37.0	36.0	75.0	**96.0**	**+21.0**
peak count	49.0	38.0	35.0	**58.0**	**+23.0**
congruent pair	6.0	2.0	29.0	**62.0**	**+33.0**
trace length	3.0	7.0	21.0	**22.0**	**+1.0**
scan moiré	5.0	8.0	10.0	**63.0**	**+53.0**
Localization					
crop origin	0.0	6.0	39.0	**89.0**	**+50.0**
brightest blob	10.0	4.0	37.0	**92.0**	**+55.0**
rect swap	28.0	53.0	55.0	**94.0**	**+39.0**
retouch region	38.0	34.0	36.0	**55.0**	**+19.0**
circle fit	4.0	6.0	40.0	**62.0**	**+22.0**
print misreg	0.0	1.0	24.0	**78.0**	**+54.0**
gain center	0.0	0.0	6.0	6.0	0.0
Color					
film color	2.0	1.0	22.0	**54.0**	**+32.0**
swatch RGB	27.0	16.0	87.0	76.0	−11.0
patch means	8.0	3.0	40.0	**52.0**	**+12.0**
color count	19.0	18.0	56.0	**73.0**	**+17.0**
channel mix	1.0	2.0	1.0	1.0	0.0
Mean (30 tasks)	10.8	11.5	29.6	**53.8**	**+24.2**

Table 7: **Qwen3.5-27B per-task test accuracy (%).** Held-out test-100, all 30 tasks. Instruct and Thinking are CoT with thinking mode off vs. on. Under Code-with-Image, "No skill" is the bare solver and "Ours" uses the library delivered by the pipeline; Δ is their difference. Skills are delivered only past the $+3$pp validation gate; where none was delivered, Ours is the bare solver.

Task	Instruct	Thinking	Code-with-Image		Δ
			No skill	Ours	
Transformation & rearrangement					
cyclic shift	0.0	0.0	44.0	**100.0**	**+56.0**
restore count	8.0	40.0	24.0	**82.0**	**+58.0**
shift align	0.0	0.0	55.0	**100.0**	**+45.0**
alignment	0.0	0.0	72.0	**97.0**	**+25.0**
rot scale	8.0	9.0	25.0	**88.0**	**+63.0**
transform chain	37.0	52.0	73.0	**98.0**	**+25.0**
unshuffle 2D	3.0	19.0	8.0	8.0	0.0
jigsaw	0.0	0.0	22.0	**33.0**	**+11.0**
rotated jigsaw	1.0	1.0	2.0	**4.0**	**+2.0**
restore measure	14.0	29.0	3.0	**19.0**	**+16.0**
ring rotor	2.0	1.0	7.0	**35.0**	**+28.0**
double exposure	0.0	0.0	0.0	**24.0**	**+24.0**
Perception & counting					
grating period	22.0	18.0	72.0	**98.0**	**+26.0**
anomaly grid	45.0	65.0	83.0	**95.0**	**+12.0**
peak count	20.0	44.0	37.0	**88.0**	**+51.0**
congruent pair	9.0	10.0	67.0	**92.0**	**+25.0**
trace length	11.0	8.0	14.0	**82.0**	**+68.0**
scan moiré	3.0	4.0	9.0	**96.0**	**+87.0**
Localization					
crop origin	9.0	11.0	32.0	**96.0**	**+64.0**
brightest blob	0.0	3.0	28.0	**89.0**	**+61.0**
rect swap	17.0	38.0	62.0	**89.0**	**+27.0**
retouch region	5.0	25.0	13.0	**62.0**	**+49.0**
circle fit	0.0	0.0	17.0	**37.0**	**+20.0**
print misreg	1.0	4.0	40.0	**82.0**	**+42.0**
gain center	0.0	0.0	6.0	**7.0**	**+1.0**
Color					
film color	1.0	5.0	39.0	29.0	-10.0
swatch RGB	8.0	7.0	31.0	**57.0**	**+26.0**
patch means	1.0	4.0	34.0	**81.0**	**+47.0**
color count	34.0	35.0	58.0	**69.0**	**+11.0**
channel mix	0.0	0.0	0.0	0.0	0.0
Mean (30 tasks)	8.6	14.4	32.6	**64.6**	**+32.0**

Table 8: **GPT-5.6-luna per-task test accuracy (%).** The model uses its own Code-with-Image solver, reflections, and skills; columns as in Table 7. Luna settings: Appendix A.2.

Task	Instruct	Thinking	Code-with-Image		Δ
			No skill	Ours	
Transformation & rearrangement					
cyclic shift	73.0	69.0	85.0	**100.0**	**+15.0**
restore count	12.0	66.0	30.0	**99.0**	**+69.0**
shift align	0.0	36.0	83.0	**97.0**	**+14.0**
alignment	0.0	11.0	79.0	**96.0**	**+17.0**
rot scale	16.0	35.0	30.0	**100.0**	**+70.0**
transform chain	11.0	60.0	61.0	**100.0**	**+39.0**
unshuffle 2D	3.0	8.0	6.0	**8.0**	**+2.0**
jigsaw	0.0	1.0	3.0	**6.0**	**+3.0**
rotated jigsaw	0.0	1.0	1.0	**10.0**	**+9.0**
restore measure	26.0	48.0	27.0	**59.0**	**+32.0**
ring rotor	2.0	2.0	7.0	**56.0**	**+49.0**
double exposure	0.0	21.0	13.0	**32.0**	**+19.0**
Perception & counting					
grating period	27.0	24.0	91.0	91.0	0.0
anomaly grid	16.0	72.0	54.0	**61.0**	**+7.0**
peak count	33.0	39.0	31.0	**60.0**	**+29.0**
congruent pair	11.0	33.0	20.0	**58.0**	**+38.0**
trace length	10.0	16.0	1.0	**82.0**	**+81.0**
scan moiré	0.0	0.0	20.0	**95.0**	**+75.0**
Localization					
crop origin	3.0	19.0	29.0	**89.0**	**+60.0**
brightest blob	1.0	13.0	83.0	83.0	0.0
rect swap	34.0	13.0	52.0	**62.0**	**+10.0**
retouch region	33.0	42.0	36.0	36.0	0.0
circle fit	1.0	28.0	42.0	**49.0**	**+7.0**
print misreg	0.0	0.0	15.0	**99.0**	**+84.0**
gain center	0.0	0.0	35.0	**67.0**	**+32.0**
Color					
film color	3.0	6.0	89.0	**100.0**	**+11.0**
swatch RGB	49.0	55.0	81.0	**87.0**	**+6.0**
patch means	4.0	13.0	64.0	**78.0**	**+14.0**
color count	17.0	31.0	63.0	**95.0**	**+32.0**
channel mix	2.0	5.0	9.0	**48.0**	**+39.0**
Mean (30 tasks)	12.9	25.6	41.3	**70.1**	**+28.8**

C TASK CARDS AND WORKED EXAMPLES

C.1 TRANSFORMATION & REARRANGEMENT

cyclic shift

Question. The image was produced by cyclically shifting the original picture (pixels wrap around the edges). What are the shift amounts (dx, dy)? Tolerance $\pm$2px, both required.

Construction. A COCO square is rolled by a uniformly drawn (dx, dy); tiers lower the prominence of the wrap seam.

GT guarantee. Constructive (shift known); flat/low-gradient images rejected by an information screen so the wrap seam is detectable.

restore count

Question. First undo a cyclic shift, then count the shapes entirely in the left half of the restored image. Answer: one integer.

Construction. Flat coloured shapes on a plain light background (exact masks recorded), then the frame is rolled.

GT guarantee. Constructive masks; a seam-recovery screen re-derives the shift from the file; "count in a fixed region" makes the restore step mandatory (cyclic shifts preserve global counts).

shift align

Question. Image 2 equals image 1 rotated CCW by θ about the center, then cyclically shifted by (dx, dy). Report dx, dy, θ (generation grids 8px / 15°; scored at $\pm$2px / $\pm 1.875^\circ$).

Construction. Circular window (white outside) deliberately removes the wrap-seam shortcut; the two parameters contaminate each other's estimators.

GT guarantee. Constructive; a brute grid re-estimation on the saved files must return the true triple as unique argmin with margin, else rejected.

alignment

Question. Two images are overlapping crops of the same larger picture. Report the shift (dx, dy) between them, $\pm$2px.

Construction. Two windows are cropped from one larger COCO square at offset (dx, dy); tiers scale the offset magnitude.

GT guarantee. Constructive; flat or low-texture crops are rejected so the shift is identifiable.

rot scale

Question. Image 2 equals image 1 zoomed in by $s \geq 1$ about the center, then rotated CCW by θ through a circular window. Report θ, s ($\pm 3^\circ$ and ± 0.05, both required).

Construction. Rotation/scale applied inside an inscribed disc; tiers move from right-angle/integer-scale to arbitrary parameters.

GT guarantee. Constructive; recovery screened for uniqueness on generation.

transform chain

Question. Image 2 was produced from image 1 by some combination of cyclic shifts, right-angle rotations, and mirror flips in unknown order. Report the *canonical form*: `shift(dx,dy); rot(k); fliph(f)`.

Construction. Random operation chains; the answer is the unique $D_4 \times$shift decomposition (group structure guarantees uniqueness; raw operation order is provably non-identifiable, which we discovered and designed around).

GT guarantee. The generator embeds a brute-force canonical decomposer (independent second algorithm); samples with more than one lattice solution are rejected.

unshuffle 2D

Question. Row order and column order were independently shuffled (tiles within a row/column stay together). Report both restoring permutations.

Construction. Rows and columns permuted with badges on the left/top edges; tiers scale grid size (2×3 to 4×4).

GT guarantee. Constructive; row-strip and column-strip uniqueness/information screens; degenerate double-identity permutations rejected.

jigsaw

Question. The photo was cut into numbered tiles and shuffled. For each original position, name the displayed tile that belongs there.
Construction. $r \times c$ tiles with number badges; tiers scale grid size (3×3 to 6×6).
GT guarantee. Constructive permutation; tiles screened for uniqueness and information content (flat-center images rejected).

rotated jigsaw

Question. Tiles are shuffled and each rotated by $0/90/180/270°$. For each original position give the displayed tile and the CCW rotation that restores it (e.g., `3R90`).
Construction. Square tiles; tiers from 2×2 to 3×3 with forced rotations at the top tier.
GT guarantee. Constructive; an orientation-distinctness screen (every tile must differ from all rotations of every tile) makes the joint answer pixel-unique.

restore measure

Question. Undo the shift, then report what percentage of the left half is covered by textured polygonal patches (± 1.5pp).
Construction. Polygons filled with patches from another photo (no single-color threshold segments them); frame rolled.
GT guarantee. Constructive pixel-exact masks; a mandatory-restore screen rejects samples where the unrestored answer would be within twice the tolerance.

ring rotor

Question. An image is divided into concentric rings and exactly one ring is rotated. Report the ring index and signed angle (index exact, angle within $\pm 5°$).
Construction. A COCO photograph is remapped into annuli with visible boundaries; one annulus receives an item-specific rotation and tiers vary ring count and ambiguity.
GT guarantee. Constructive (ring and angle recorded); an independent polar-coordinate correlation search verifies a unique best ring/angle pair.

double exposure

Question. An image mixes a dominant photograph with one or two fainter cyclically shifted copies. Recover every signed shift, ordered from the heavier to lighter ghost (each component within ± 8px).
Construction. One or two wrap-around translations with distinct weights are composited with the source; tiers vary ghost count, strength, and separation.
GT guarantee. Constructive (shifts and weights recorded); an independent decomposition verifies count, order, magnitude, and sign, including cases where unsigned autocorrelation is ambiguous.

Worked examples. Representative instances with gold answers.

shift align



What are dx, dy (multiples of 8, between -64 and 64) and theta?

Answer: `64, 64, 90`

alignment



What is the content shift (dx, dy) in pixels?

Answer: `207, -187`

rot scale



What are theta (in degrees, 0-360) and s?

Answer: `210.7, 1.069`

transform chain



What is the canonical form?

Answer: `shift(48,32); rot(0); fliph(1)`

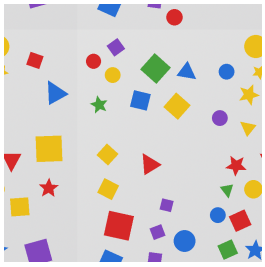
cyclic shift



Find the shift (dx, dy) in pixels.

Answer: `171, 201`

restore count



Undo the shift to restore the original picture, then count how many shapes lie ENTIRELY in the LEFT HALF of the restored picture.

Answer: `18`

unshuffle 2D



In what order should the numbered rows be arranged (top to bottom), and the numbered columns (left to right), to reconstruct the original photo?

Answer: `rows: 2, 1, 4, 3; cols: 2, 4, 3, 1`

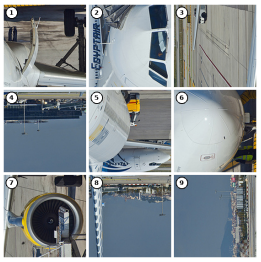
jigsaw



In what order should the numbered pieces be arranged to reconstruct the original image? Please provide your answer as a sequence indicating where each numbered piece should be placed in the final arrangement.

Answer: `[25, 8, 4, 36, 33, 14, 27, 32, . . . ]` (36-piece permutation)

rotated jigsaw



For each ORIGINAL position, which displayed tile belongs there, and by how many degrees must it be rotated COUNTERCLOCKWISE to restore its original orientation? Answer exactly in this format (one term per original position): 3R90, 1R0, 4R270, 2R0.

Answer: `9R270, 8R180, 4R180, 3R270, 5R180, 2R270, 7R270, 1R180, 6R270`

restore measure



After undoing the shift, what percentage of the LEFT HALF of the original picture's area is covered by the textured polygonal patches?

Answer: `8.3`

ring rotor



Concentric rings; exactly one was rotated. Which ring, and by how many degrees?

Answer: `2, 140.4`

double exposure



The image blends a dominant photo with fainter cyclically shifted ghosts. Report the shift (dx, dy) of every ghost, heavier first.

Answer: `-176, 171`

C.2 Perception & counting

grating period

Question. A faint sinusoidal stripe grating was added over the whole photo. Report the stripe period in pixels (one decimal, ± 1.5px).

Construction. A low-amplitude sinusoid at a random orientation and period is added to a COCO photo; tiers lower the amplitude and widen the period range.

GT guarantee. Constructive (period known exactly); an amplitude/detectability screen requires a dominant off-DC peak in the image's Fourier magnitude so the period is recoverable in principle.

anomaly grid

Question. A synthetic grid (4×4 to 7×7 by tier) repeats one drawn shape with small position, tilt, and size jitter; a few cells deviate beyond that jitter. List all anomalous cells.
Construction. K cells (2–4 by tier) receive a size change or a one-channel colour shift; tiers enlarge the grid and shrink the deviation.
GT guarantee. A separability screen on the saved files requires changed-cell deltas to dominate the natural within-cell variation by a fixed factor, so the answer set is recoverable in principle.

peak count

Question. The dark image contains faint glowing spots in background noise. How many distinct spots are there? Answer: one integer (exact).
Construction. N soft Gaussian blobs at low signal-to-noise are placed over noise; tiers raise N and lower the SNR.
GT guarantee. Constructive count; a detectability screen requires each planted blob's peak to exceed the local noise floor by a fixed margin and to be separable from its neighbours.

congruent pair

Question. Several filled shapes are shown; exactly one pair is congruent up to rotation (no mirroring or scaling; mirrored look-alikes are decoys). Name the pair.
Construction. Shapes generated with controlled pairwise dissimilarity margins; one planted congruent pair.
GT guarantee. Constructive; margin screens keep the planted pair uniquely congruent.

trace length

Question. A single continuous winding, self-crossing cyan path is drawn. Report its length in pixels divided by 200 (one decimal, ± 0.3).
Construction. A random winding polyline of controlled total length is rendered at fixed width; tiers add length, sharper bends, and self-crossings.
GT guarantee. Constructive (polyline length known exactly); the path is drawn opaque at a fixed stroke width so its geometry is fully determined by the saved file.

scan moiré

Question. A halftone print with known dot pitch is re-scanned at an unknown scale. Recover the scale factor from the resulting moiré pattern (within ± 0.004).
Construction. A COCO photograph is halftoned on a 6px lattice, resized by a non-unit scale, and center-cropped; tiers move the spectral peak progressively closer to the nominal frequency.
GT guarantee. Constructive (scale recorded); a windowed sub-pixel FFT solver must recover it within 0.002, while an integer-bin shortcut is screened to miss the tolerance.

Worked examples. Representative instances with gold answers.

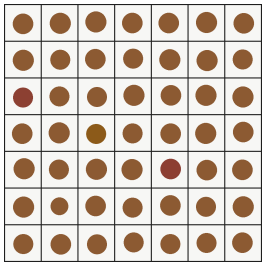
grating period



What is the period of the stripes, in pixels from one stripe to the next?

Answer: `8.2`

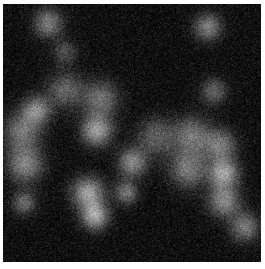
anomaly grid



A 7×7 grid of similar objects; a few cells are anomalous beyond natural variation. Identify every anomalous cell.

Answer: `15, 24, 33, 37`

peak count



How many distinct glowing spots are there?

Answer: `22`

congruent pair



Which two shapes are congruent?

Answer: `8, 18`

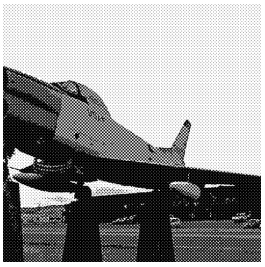
trace length



How long is the cyan path in pixels? Report the length in pixels DIVIDED BY 200, to one decimal (for example, a 730-pixel path is 3.7).

Answer: `12.3`

scan moiré



A halftone print was rescanned at an unknown scale r. Report r to within ± 0.004.

Answer: `0.9811`

C.3 Localization

crop origin

Question. The second image is a small detail cropped from the first (then slightly resized and re-compressed). Locate the source rectangle in the first image (box, IoU≥ 0.5; coordinates normalized to 0–1000).
Construction. A random sub-rectangle is cropped, resized and re-JPEG'd; tiers shrink the crop and raise the resize factor and brightness jitter.
GT guarantee. Constructive box; a template-matching screen requires the true location to rank in the top five by normalized cross-correlation, so the crop is localizable despite the pixel mismatch.

brightest blob

Question. Among several glowing blobs of differing intensity, report the center of the single brightest one (normalized 0–1000, ± 2px).
Construction. Multiple Gaussian blobs of differing peak intensity are placed over noise; tiers narrow the intensity gap between the brightest and the runner-up.
GT guarantee. Constructive; a margin screen keeps the brightest blob's peak separated from the second-brightest by a fixed factor, so the argmax is unambiguous.

rect swap

Question. Two same-size rectangles in image 2 had their contents swapped. Report both rectangles (`x1,y1,x2,y2` normalized to 0–1000; each box IoU≥ 0.5, order-free).
Construction. Rectangles snapped to an 8px grid; contents exchanged.
GT guarantee. Blob-recovery screen: thresholded difference must yield exactly two components matching the true boxes; too-similar patches rejected.

retouch region

Question. One rectangular region was texture-smoothed (a blur/beauty filter) with soft edges. Locate the retouched rectangle (box, IoU≥ 0.5; normalized 0–1000).
Construction. A random rectangle is low-pass filtered and feathered into a COCO photo; tiers soften the boundary and shrink the region.
GT guarantee. Constructive box; a detectability screen requires a measurable drop in local high-frequency energy inside the region versus outside, so the boundary is recoverable in principle.

circle fit

Question. Most dots lie along a circular arc (with scatter); a few are distractors. If the arc were completed into a full circle, report its center (normalized 0–1000, ± 4px).
Construction. Dots are sampled along an arc with Gaussian scatter plus random off-arc distractors; tiers raise the scatter and shorten the arc.
GT guarantee. The recorded center is the least-squares circle fit to the on-arc dots, kept only when it lies within 8px of the constructed center.

print misreg

Question. Two rotated colour halftone lattices share pitch and orientation, but one is shifted by a signed non-integer offset. Recover (dx, dy) within ± 0.3px.
Construction. Cyan and magenta dot layers are rendered with item-specific pitch and rotation, then displaced by a non-integer offset smaller than half a pitch.
GT guarantee. Constructive (offset recorded); independent lattice fitting verifies pitch, angle, correspondence, and phase displacement from the saved image.

gain center

Question. Given a reference image and a relit copy with two or three soft spotlights plus an ambient tilt, report every spotlight centre ordered by strength (within ± 15px).
Construction. Pairwise-distinct Gaussian gain fields and a linear illumination plane are multiplied into the source photograph; tiers vary the number, strength, width, and overlap of lights.
GT guarantee. Constructive (centres and gains recorded); robust regression on the image ratio verifies all centres and rejects visually brightest-point shortcuts.

Worked examples. Representative instances with gold answers.

crop origin



Where in the first image was the detail taken from? Report the crop rectangle in first-image coordinates as x1,y1,x2,y2 (normalized 0–1000).

Answer: `217,33,333,183`

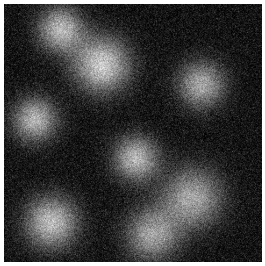
rect swap



Which two rectangles were swapped? Give each as x1,y1,x2,y2 (normalized 0–1000).

Answer: `383,250,500,333` ↔ `133,483,250,567`

brightest blob



Give the coordinates x,y of the CENTER of the single brightest blob, (normalized 0–1000).

Answer: `379,235`

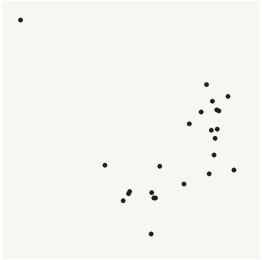
retouch region



Locate the retouched rectangle. Report it as x1,y1,x2,y2 (normalized 0–1000).

Answer: `750,83,950,250`

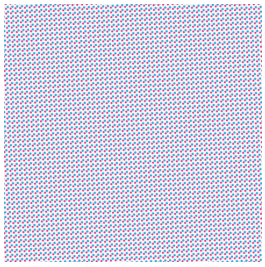
circle fit



If the arc were completed into a full circle, where would its CENTER be? Report the center coordinates x,y, (normalized 0–1000).

Answer: `552,479`

print misreg



A cyan and a magenta halftone lattice share pitch and rotation, but one layer is slightly offset. Report the offset (dx, dy) in pixels.

Answer: `-3.39, 3.39`

gain center



Report the centers of all spotlights, ordered from strongest to weakest peak gain.

Answer: `195, 319; 117, 118`

C.4 COLOR

film color

Question. A rectangular paint swatch is crossed by a translucent colour film of stated opacity. Report the RGB of the film itself, as at 100% opacity (per-channel ± 10).
Construction. A flat swatch is painted, then a translucent coloured strip is alpha-composited across its middle; the top and bottom of the swatch stay uncovered.
GT guarantee. Constructive (film colour recorded); un-blending the covered band with the swatch colour read from the uncovered region recovers it, and a contrast screen makes reporting the blended or the underlying colour fail.

swatch RGB

Question. A black arrow points at exactly one of several solid-colour swatches. Report that swatch's RGB (per-channel ± 10).
Construction. Flat, noise-free swatches are painted on the photo; an outlined arrow's tip touches the target swatch; distractor line segments may be added.
GT guarantee. Constructive; the pointed swatch is a flat fill so its RGB is exact, and a screen ensures the arrow selects one swatch.

patch means

Question. A red rectangle marks a region; magenta annotation dots were sprinkled inside it after capture. Report the average RGB of the photo pixels inside the rectangle that the dots do not cover (per-channel ± 10).
Construction. The rectangle is drawn on a COCO photo and annotation dots are softly composited inside; the target is the mean over the pixels left uncovered by the dots.
GT guarantee. Constructive (mean over uncovered pixels recorded); by construction, including the dots or masking only their pure-magenta pixels misses the tolerance.

color count

Question. Several solid-colour stickers are pasted; some share the exact same fill and some differ only subtly. How many distinct fill colours are there? Answer: one integer (exact).
Construction. Stickers with a controlled number of distinct fills plus ± 3/channel noise; tiers add near-duplicate colour pairs.
GT guarantee. Constructive (distinct-colour count known); a margin screen keeps same-colour pairs within the noise band and different-colour pairs beyond it.

channel mix

Question. A camera applies unknown 3×3 channel crosstalk. Using eight labelled calibration patches, recover the true sRGB colour of a held-out target patch (per-channel ± 8).
Construction. A near-identity mixing matrix acts in linear-light space before conversion back to sRGB; tiers increase off-diagonal crosstalk.
GT guarantee. Constructive (matrix and target recorded); a linear-light calibration solver must recover the target, while naive sRGB linear and affine fits are screened to fail by a wide margin.

Worked examples. Representative instances with gold answers.

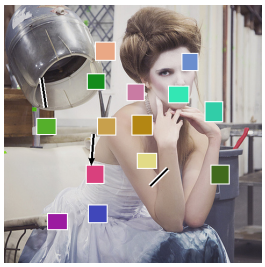
film color



What is the colour of the FILM itself?

Answer: `217,169,196`

swatch RGB



What is the exact fill colour (RGB) of the swatch the arrow points to?

Answer: `217,64,125`

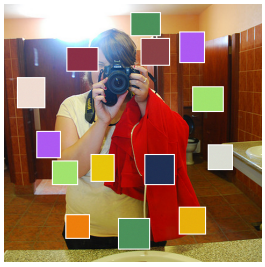
patch means



What is the AVERAGE colour (mean RGB) of the PHOTO pixels strictly inside the rectangle? Exclude the red border and everything the magenta dots cover.

Answer: `134,111,92`

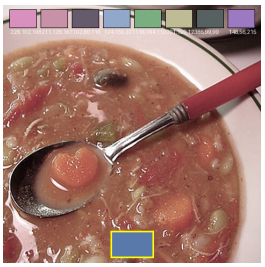
color count



How many DISTINCT fill colours are used across all stickers? Two stickers count as the same colour only if their underlying fill is identical; a subtle-but-real difference counts as a different colour.

Answer: `11`

channel mix



The image was pushed through an unknown channel-mixing matrix. Recover the target patch's true sRGB colour (each channel within ± 8).

Answer: `32, 98, 191`

D REFLECTION PROMPTS (VERBATIM)

Both prompts below are task-agnostic: the identical string is used for every task; the task never appears by name. The observational prompt is the single-call reflection; the executable reflection prompt drives the sandboxed debugging session (rendered images from executed code are returned to the model each step). Wording is reproduced verbatim; typography and spacing are adjusted for readability.

Observational reflection
Task-agnostic prompt

You previously attempted several instances of ONE task type using a stateful Python sandbox (the transcripts, your code, its outputs`{img_note}`, the ground-truth answers, and your wrong answers are shown above).

Reflect on WHY your attempts failed, then EDIT your skill library for this task type. Your future self will be given this library before attempting new instances.

Each skill MUST have exactly three sections:

When: <trigger condition>

Procedure: <numbered concrete steps>

Verify: <which INTERMEDIATE quantity to RENDER as an image (not just print), what it should look like when the procedure is working, and what it looks like when something is silently wrong>

Rules:

- Express changes as explicit edit operations:

```
<edit op="add" name="short-skill-name">When: ...
Procedure:
1. ...
Verify: ...</edit>
<edit op="replace" name="existing-name">...new full body...</edit>
<edit op="delete" name="existing-name"></edit>
```

- At most 3 edit operations per reflection. Keep the library at <= 6 skills, each under 250 words (the Verify section does not count toward the limit).
- General: must work for ANY instance of this task type. Never include answers, coordinates or values specific to the examples above.
- If the same skill has been edited repeatedly without improvement (see the attendance table), question the METHOD itself rather than its parameters – consider an alternative approach as a new skill.

Observational reflection
Task-agnostic prompt
continued

- Check the Verify sections: is there any intermediate computation your procedures rely on that is never rendered and inspected? Silent numeric bugs are invisible in printed numbers but obvious in a rendered image.
- Some attempts shown above SUCCEEDED: identify what worked there and do not break it.
- If the remaining failures look inherently hard and the library already covers them as well as it can, you may output exactly NO-EDIT instead of forcing changes.

Output ONLY the edit operations (or NO-EDIT).

Executable reflection
Task-agnostic prompt

You are debugging your own past failures on ONE task type, in a live Python sandbox.

Preloaded variables (already in the sandbox):

- `cases`: list of dicts, one per case: `cases[i]['images']` (list of PIL images), `cases[i]['gold']` (ground-truth answer string), `cases[i]['pred']` (your wrong answer; None if this was a SUCCESSFUL case), `cases[i]['code']` (list of the code strings you ran in that attempt), `cases[i]['prompt']` (the task instruction)
- Successful cases (`pred is None`) show approaches of yours that WORKED.

Your job, in this order:

1. **INVESTIGATE with code:** re-run suspect computations from your failing code on the actual images, print intermediate values AND dtypes, render intermediate arrays as images and look at them, and test hypotheses by checking against the known gold. Compare, line by line, how your successful and failing attempts compute the same quantity. Find the ROOT CAUSE, not the symptom.
2. **Verify the fix:** show with code that the corrected computation recovers the gold on at least 2 of the failing cases.
3. Only then, output skill-library edits in EXACTLY this format (and nothing after them):

```
<edit op="add" name="short-skill-name">When: ...
Procedure:
1. ...
Verify: ...</edit>
```

(ops: add / replace / delete; at most 3 edits; library <= 6 skills; each skill <= 250 words; every skill must have When/Procedure/Verify; Verify = which intermediate to RENDER and what correct vs broken looks like. Skills must be GENERAL for this task type — never include answers or values specific to these cases.)

Write ```` ```python ```` code blocks to use the sandbox; I will run them and return outputs. When you are confident, emit the edits.

Forced wrap-up
Issued when the investigation budget is exhausted

Investigation budget is exhausted. Based on everything you verified above, output your skill-library edit operations NOW, in exactly the format specified (at most 3 edit ops; each skill When/Procedure/Verify; general, no case-specific values). **Output ONLY the edit operations.**

E EXHIBITS: A DELIVERED SKILL AND A REFLECTION SESSION

E.1 A DELIVERED SKILL, VERBATIM EXCERPT

An excerpt from the library delivered for scan moiré on Qwen3.5-27B, written entirely by the model. With it, test accuracy rises from 9% with bare Code-with-Image to 96%. The constants it uses (the 6.0 px print pitch, the range of r, and the tolerance) are stated in the task prompt. At solve time the full library is appended to the system prompt. Markdown is rendered for readability, the wording is unchanged, and ellipses mark omitted material, including a reference implementation and a short second skill of reminders.

Delivered skill (excerpt)
Scan moiré • Qwen3.5-27B

Skill: `main-procedure`

When:

The task is to determine the rescaling factor r of a halftone-printed image that was re-scanned, given the original pitch (6.0 px) and the visible moiré/beat pattern.

Task restatement + domain facts:

… The "moiré" pattern is not the primary signal; the primary signal is the **second harmonic** of the halftone grid in the frequency domain. Due to the square-wave nature of halftone dots, the second harmonic (frequency $2f_0$) is often stronger and more distinct than the fundamental (f_0). The beat frequency (difference between print and scan grids) is extremely low and weak; relying on it directly causes precision errors (NEAR_MISS) or misidentification (MISIDENTIFIED_PEAK). The correct approach measures the spatial frequency of the strong harmonic peaks, converts to period, and derives r.

Pipeline:

1. **Preprocessing**: Load `image_clue[0]`, convert to grayscale, and subtract the mean to center the DC component.
2. **FFT Computation**: Compute the 2D FFT (`scipy.fft.fft2`), shift zero-frequency to center (`fftshift`), and calculate the power spectrum ($|FFT|^2$).
3. **Harmonic Search Window**: Define a search region for the **second harmonic**.
   - Calculate expected fundamental bin index: k_{fund} = image_width/original_pitch (e.g., $480/6 = 80$).
   - Target the second harmonic radius: $R_{target} = 2 \times k_{fund}$ (e.g., 160).
   - Define a radial search window: $[R_{target} - 10, R_{target} + 10]$ (e.g., [150, 170]).
4. **Peak Detection**: … **Filter**: Discard any peak where the calculated period falls outside the physical range $[4.0, 8.0]$ px.
5. **Precision Calculation**:
   - For valid peaks, compute the exact distance from the center (dy, dx).
   - Calculate observed harmonic period: P_{obs} = image_width$/\sqrt{dx^2 + dy^2}$.
   - Derive fundamental period: $P_{fund} = P_{obs} \times 2$.
   - Calculate candidate r: $r = P_{fund}/6.0$.
6. **Aggregation**: Collect all valid r candidates. If multiple exist, take the median. If none found in the harmonic window, fallback to searching the fundamental window $[k_{fund} - 10, k_{fund} + 10]$ (less robust but necessary if harmonics are suppressed).

Validation before answering (excerpt):

- **Sanity Check**: The calculated r must be in $(0.93, 1.07)$. If outside, check if you measured the fundamental instead of the harmonic (which would yield $r \approx 0.5$ or 2.0) or picked up noise.
- **Consistency**: Ensure the detected peak radius corresponds to the expected harmonic order. For a 480px image, the second harmonic should be near radius 160. A peak at radius $\sim$80 implies you found the fundamental; double your period calculation.

Delivered skill (excerpt)
Scan moiré • Qwen3.5-27B continued

Common pitfalls:

- **MISIDENTIFIED_PEAK**: Do not select the highest magnitude peak globally; it is often the DC spike or high-order noise. Always restrict search to the specific annulus defined by $2 \times (\text{width}/\text{pitch})$.
- **INCORRECT_PERIOD_TO_SCALE_MAPPING**: Remember that the FFT gives frequency. If measuring the 2nd harmonic, the period is W/radius. The *fundamental* period is $2 \times (W/\text{radius})$. Failing to multiply by 2 yields an r that is half the true value.
- **NEAR_MISS**: Integer pixel indices are insufficient. Use the exact float distance $\sqrt{dx^2 + dy^2}$ from the center to the peak pixel to achieve sub-pixel precision required for ± 0.004 tolerance.

E.2 Executable-reflection session excerpts

Machine-extracted excerpts from the executable-reflection session in round 4 of the peak-count seed traced in Table 9 (Qwen3.5-27B), launched after observational reflection had stalled. The sandbox holds training cases with their ground-truth answers (see the executable-reflection prompt in Appendix D). Steps are numbered within the session; line breaks and indentation are restored, and ellipses mark omitted material.

Executable-reflection session
Peak count • Step 5 • Machine-extracted excerpts

Assistant

Let me run the actual code from a failing case to see what's happening:

```
# Test on Case 1 (gold=20, pred=18)
case1 = cases[1]
img_pil = case1['images'][0]
img = np.array(img_pil)
gray = img if len(img.shape) == 2 else cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
gray_norm = gray.astype(float) / gray.max()
blobs = blob_log(gray_norm, min_sigma=8, max_sigma=35, num_sigma=10, threshold=0.05)
print(f"Number of blobs detected: {len(blobs)}")
print(f"Gold answer: {case1['gold']}")
for thresh in [0.05, 0.1, 0.15, 0.2]:
    blobs_t = blob_log(gray_norm, min_sigma=8, max_sigma=35, num_sigma=10, threshold=
  thresh)
    print(f"Threshold {thresh}: {len(blobs_t)} blobs")
...
```

Execution output

```
Number of blobs detected: 18
Gold answer: 20
Threshold 0.05: 18 blobs
Threshold 0.1: 18 blobs
Threshold 0.15: 13 blobs
Threshold 0.2: 7 blobs
```

Executable-reflection session
Peak count • Steps 6–7 • Machine-extracted excerpts

Assistant (step 6)

I can see the issue! The blob detection is finding 18 blobs but the gold answer is 20. Looking at the visualization, some blobs are being missed or merged. Let me investigate further by looking at different sigma ranges and also try the watershed approach:

Executable-reflection session
Peak count • Steps 6–7 • Machine-extracted excerpts continued

```
for min_s in [5, 8, 10]:
    for max_s in [30, 35, 40]:
        blobs = blob_log(gray_norm, min_sigma=min_s, max_sigma=max_s, num_sigma=10,
  threshold=0.05)
        print(f"min_sigma={min_s}, max_sigma={max_s}: {len(blobs)} blobs")
...
```

Execution output (excerpt)

```
min_sigma=5, max_sigma=30: 19 blobs
min_sigma=5, max_sigma=35: 18 blobs
min_sigma=8, max_sigma=30: 19 blobs
min_sigma=8, max_sigma=35: 18 blobs
...
```

Assistant (step 7)

I see the issue - watershed requires a 3-channel image. Let me fix that and also investigate why we're getting 18 instead of 20:

Execution output (excerpt)

```
Testing different thresholds with min_sigma=5, max_sigma=30:
Threshold 0.01: 256 blobs
Threshold 0.02: 43 blobs
Threshold 0.03: 19 blobs
...
=== Trying Watershed Approach ===
Seed threshold 0.3: 13 seeds
...
Watershed regions: 13
```

Executable-reflection session
Peak count • Step 8 and closing summary • Machine-extracted excerpts

Assistant (step 8)

I see the problem now! The watershed approach is merging blobs because they're too close together. The blob_log method with min_sigma=5, max_sigma=30 gives 19 blobs, but we need 20.

Assistant (closing summary, excerpt)

The watershed approach fails because merged blobs don't separate properly. The root cause is that simple blob detection or watershed alone isn't robust enough. We need a hybrid approach that:

- Uses blob detection with adaptive parameters based on image characteristics
- Falls back to contour-based counting when blob detection is close but not exact
- Uses multiple methods and selects the best result

E.3 One task through all three reflection modes

Peak count on Qwen3.5-27B asks how many soft, faint glowing spots a dark, noisy 480×480 image contains. Bare Code-with-Image scores 37% on test, the library delivered by O-Refl and E-Refl 47%, and the full pipeline 88%. Table 9 follows the seed whose library was delivered; every entry is taken from the run records.

Table 9: **Peak count, Qwen3.5-27B: the reflection record of one seed.** Probe: accuracy on the 32 training-probe items after the round's edit.

Round	Mode	What the reflection did	Probe (%)
–	–	bare solver, no skill	18.8
1	O-Refl	reads the failed attempts and adds a rule: Otsu threshold, then count contours	28.1
2–3	O-Refl	adds a second rule built on a Laplacian-of-Gaussian blob detector; refines both	28.1, 25.0
4	E-Refl	two rounds without a new best, so reflection moves to the sandbox: it re-runs a failed attempt (18 spots found, 20 present), sweeps the detector's threshold and scale range, observes that watershed merges touching spots, and adds a third rule that cross-checks two detectors. *Delivered*: validation 46% against 38% bare	43.8
5–9	O-Refl, E-Refl	further rules are added; the library grows from 610 to 1,042 words. No round beats round 4, and after five such rounds the search closes	$\leq$ 40.6
C-Refl	C-Refl	reviews the record above and replaces the competing rules by one procedure (below). Validation 86%	78.1

How C-Refl finds a better procedure. Each rule in Table 9 was written from one batch of cases and a short score history, and the library grows from 610 to 1,042 words as rules are added, so competing rules accumulate. C-Refl instead reviews the whole record at once: the call that writes the delivered procedure receives the evidence in the first box below, which answers three questions. (i) *Which edits helped?* The round-4 edit raised probe accuracy by 18.8 points, whereas the round-5 rule lowered it by 6.3, so one change is worth keeping and the other is not. (ii) *Which way does the library err?* 16 of its 17 numeric failures are under-counts, by a median of one spot. (iii) *What did the successful attempts use?* Four of the eight successful attempts take their count from an LoG or DoG blob detector, three of them at threshold 0.05, whereas Otsu thresholding without cleanup produced 809 noise components in one attempt. A sandbox run of the LoG detector at threshold 0.05 reproduces a previously solved case but finds 19 of 20 spots on a previously failed one, so the under-count persists. C-Refl therefore replaces the competing rules with one procedure (second box): it keeps the round-4 idea of cross-checking two detectors, runs the LoG detector at this threshold, drops Otsu-based counting, and re-checks at a lower threshold when the count looks low. The procedure raises validation accuracy from 46% to 86%, and six later rounds find no further improvement.

Cross-round reflection input
Peak count • Machine-extracted excerpts

Cross-seed record

```
CROSS-SEED (final libraries
  s1 v5  undelivered,
  s2 v4  sel50 0.46,
  s3 v10 undelivered):
  rules recurring in >= 2 seeds (robust): none
```

(i) Revision history: which edits helped

```
[s2] delivered v4 (sel50 0.46 vs base 0.38)
trfast by version:
  base 0.19 -> v1:0.28  v2:0.28  v3:0.25  v4:0.44
               v5:0.38  v6:0.31  v8:0.28  v9:0.41
largest-effect edits:
  v1 (+0.09): added count-blobs-otsu
  v4 (+0.19): added count-blobs-hybrid; [...]
  v5 (-0.06): added count-blobs-simple-contours
```

Cross-round reflection input
Peak count • Machine-extracted excerpts continued

(ii) Error statistics: which way the library errs

ERROR STATISTICS over 17 numeric failures: **16 under-estimates** (median -1), 1 over-estimates (median $+1$), 0 exact; median |error| 1. ONE-SIDED: the procedure is biased in that direction [. . .]

(iii) Successful attempts: detector settings and a rejected approach

ANCHOR: blob_detection_method = `blob_log` (Laplacian of Gaussian) or `blob_dog` (Difference of Gaussians) [. . .]

ANCHOR: threshold = 0.05 [. . .]

REJECT: Otsu thresholding followed by connected components without morphological cleaning because it produced too many noise artifacts (e.g., 809 components); from: attempt 3

Sandbox check of the LoG detector at threshold 0.05

(computed by the harness from RESULT lines)

```
case 0 (solved-before): calc=9  gold=9  -> REPRODUCED
  (exact under the task's scoring rule)
case 1 (failed-before): calc=19 gold=20 -> MISMATCH
```

Delivered procedure (excerpt)
Peak count • Qwen3.5-27B • written by C-Refl

Skill: `main-procedure`

When:

The task is to count distinct, soft, faint glowing spots (blobs) in a 480x480 dark, noisy grayscale image where simple thresholding often merges nearby objects or misses faint ones.

Pipeline (excerpt):

1. **Preprocessing**: Convert input to grayscale float array normalized to [0, 1].
2. **Primary Detection (LoG)**: Run `skimage.feature.blob_log` with **min_sigma=5**, **max_sigma=15**, **threshold=0.05**.
3. **Secondary Detection (DoG)**: Run `skimage.feature.blob_dog` with identical sigma ranges (**min_sigma=5**, **max_sigma=15**) and **threshold=0.05**.
4. **Consistency Check**: Compare counts from LoG (N_{log}) and DoG (N_{dog}).
   - If $|N_{log} - N_{dog}| \leq 1$, accept the average (rounded).
   - If difference > 1, prefer the method with higher spatial coverage (usually LoG for faint spots) but validate against visual density.

Validation before answering (excerpt):

- **Bias Correction**: If the count seems suspiciously low compared to the number of visible bright regions, assume under-detection (NEAR_MISS bias). Re-run LoG with **threshold=0.04** or **min_sigma=4** as a secondary check, but prioritize the 0.05/5-15 configuration unless it yields zero.

Common pitfalls (excerpt):

- **UNDER_ESTIMATE_LARGE**: Aggressive Gaussian blurring or Otsu thresholding merges touching blobs. Avoid watershed on raw Otsu masks without careful seed separation; prefer feature-based detectors (LoG/DoG).

A fallback step, a worked example, and the reference code are omitted.

F FULL PSEUDOCODE

Algorithms 1 and 2 describe trajectory-level skill refinement and its executable-reflection session. Algorithm 3 gives cross-round reflection. Selection rules and configurations: Appendix G.

Algorithm 1 Trajectory-level skill refinement: one task, frozen model, one seed

▷ ***Phase 0: collect bare Code-with-Image experience (no skill)***
solve the full training slice with the standard protocol; keep every trajectory (code, outputs, rendered images, answer)
probe ← a fixed, difficulty-balanced subset of the training instances
library ← empty; incumbent ← none

▷ ***Phase 1: refine the skill library*** *(fixed round budget with early stopping)*
for round $= 1, 2, \ldots$ **do**
  cases ← stratified failures from the latest training rollout + successes from Phase 0
  if the probe score has not improved for two rounds **then**
    edits ← EXECUTABLEREFLECT(cases, library) ▷ *Algorithm 2: experiment, then edit*
  else
    edits ← one reflection call that reads the cases (with images) and proposes edits
  end if
  apply the edit ops to library ▷ *skills inside the incumbent cannot be deleted*
  probeScore ← accuracy of library on the fixed probe items
  remember (library, probeScore); update the incumbent on a new best score
  continue from this candidate, even if its score dropped; stop on patience or round budget
end for

▷ ***Phase 2: select and gate*** *(on held-out validation items reflection has never seen)*
rank versions that improve on bare probe accuracy by the probe margin
evaluate the highest-ranked version on validation; if it fails, try the runner-up once
if validation score $\geq$ bare Code-with-Image + margin (equation 5) **then**
  deliver that library
else
  deliver nothing ▷ *retain bare Code-with-Image for this task*
end if

Algorithm 2 EXECUTABLEREFLECT: the interventional phase of self-reflection

sandbox ← Python session preloaded with, for each case: its images, the gold answer, the model's wrong answer, and the code the model originally ran
for step $= 1, 2, \ldots$ (bounded budget) **do**
  model writes investigation code ▷ *its own choice: re-run old code, print values and dtypes, render intermediates, test a fix against gold*
  execute; return the text output and the rendered images to the model
  if model outputs skill edits **then**
    return edits ▷ *it decides on its own when the cause is verified*
  end if
end for
forced wrap-up: one final call: "based on what you verified, output the edits now"
return edits

Algorithm 3 CROSSROUNDREFLECT: the last escalation step, one task

history ← library versions, edits with score changes, and execution records of O-Refl and E-Refl
library ← the selected library, or empty if none was delivered
for round $= 1, 2, \ldots$ **do**
  diagnosis ← earlier revisions reassessed against history; remaining failures classified
  evidence ← sandbox tests of procedures taken from successful attempts
  candidates ← bounded edits or a full procedure rewrite ▷ *from diagnosis and evidence*
  candidate ← best of candidates on the training probe
  accept or reject candidate
  history ← history + this round's outcome; stop on patience or round budget
end for
deliver the best accepted library; otherwise keep the earlier one

G BENCHMARK AND PROTOCOL DETAILS

G.1 COMPARISON WITH VISUAL REASONING BENCHMARKS

Table 10 compares CwI-Bench with representative visual reasoning benchmarks.

Table 10: **Comparison with visual reasoning benchmarks.** Tools and free code refer to interactive visual-tool use. Generator: the benchmark ships an instance generator, so new instances can be sampled without bound; learning splits: disjoint train/validation/test instances within each task; reflection: evaluating skill improvement on new instances through reflection. ✗: not part of the reported benchmark protocol. Task/category counts follow each benchmark's convention.

Benchmark	Tasks	QA	Tools	Multi-turn	Free code	Generator	Learning splits	Reflection
MMStar (2024a)	6	1.5K	✗	✗	✗	✗	✗	✗
MathVista (2024)	5	6.1K	✗	✗	✗	✗	✗	✗
V*Bench (2024)	2	191	✓	✗	✗	✗	✗	✗
HR-Bench 4K (2025)	6	200	✓	✗	✗	✗	✗	✗
VisualToolBench (2025)	5	1.2K	✓	✓	✓	✗	✗	✗
Agentic-MME (2026)	35	418	✓	✓	✓	✗	✗	✗
VTC-Bench (2026)	9	680	✓	✓	✓	✗	✗	✗
TIR-Bench (2026b)	13	1.2K	✓	✓	✓	✗	✗	✗
CwI-Bench (ours)	**30**	∞ (6.3K)	✓	✓	✓	✓	✓	✓

CwI-Bench pairs computational visual tasks with generators that construct images from known answers. Labels come from the generating computation and recovery checks, without manual annotation, and new instances can be sampled for learning and evaluation. Disjoint train, validation, and test splits support evaluation of initial solving performance and improvement through reflection.

G.2 COMPUTATIONAL CHALLENGES AND TARGET VERIFICATION

The benchmark emphasizes quantities that are difficult to recover reliably by inspection, such as pixel-level coordinates and continuous transformation parameters. Many answers are composites scored by exact match, including a canonical transform decomposition, a full permutation, or a tuple of coordinates. These targets motivate explicit computation, while task-specific uniqueness and separability screens reject ambiguous instances. Performance without code is measured empirically under the same scorer.

Each task has four difficulty tiers. Generators construct images from known targets and use independent recovery checks, such as enumeration, blob recovery, or grid re-estimation, to verify the saved instances. These checks certify that an instance is solvable and that shortcut answers fail; they are

never shown to a model and prescribe no method, since scoring depends on the answer alone. Composite transforms are reduced to canonical decompositions to remove equivalent answers. Training photographs come from COCO (Lin et al., 2014) `train2017`, test exclusively from `val2017`; an automated audit checks source-image overlap. Every task has 60 training, 50 validation, and 100 test instances, all disjoint.

G.3 Effect of a larger round budget

CwI-Bench fixes the inference budget at 5 rounds and 40,960 tokens, and reaching the answer within it is part of what the benchmark measures: a strategy that needs more interaction to reach the same answer is a weaker strategy. Table 11 shows what relaxing that budget alone buys. Doubling it to 10 rounds and 81,920 tokens raises bare overall accuracy by 4.3 points on 9B and 9.0 on 27B. On the two Qwen executors, refined skills gain 26.9 and 32.0 points at the default budget (Table 2). The two costs also differ in kind: a larger budget is paid on every test instance, whereas skills are acquired once, offline.

Table 11: **Effect of a larger round budget.** Bare Code-with-Image test accuracy (family means, %) at the default budget (5 rounds, 40,960 tokens) vs. a relaxed budget (10 rounds, 81,920 tokens). Same items and scorer as Table 1.

Model	Budget	Transformation	Perception	Localization	Color	Overall
Qwen3.5-9B	5 rounds / 41k	8.9	16.7	8.6	13.2	11.1
	10 rounds / 82k	**14.1**	**20.2**	**9.6**	**20.8**	**15.4**
Qwen3.5-27B	5 rounds / 41k	27.9	47.0	28.3	32.4	32.6
	10 rounds / 82k	**35.9**	**56.8**	**39.7**	**39.4**	**41.6**

G.4 Reflection loop: search, selection, and limits

Search. Let D_{tr} and D_{val} denote the training and validation sets for a task. O-Refl and E-Refl use a fixed, difficulty-balanced probe $D_{\mathrm{probe}} \subset D_{\mathrm{tr}}$. Each round edits the latest candidate library and evaluates it on the probe; search can continue after a score decrease. The best library so far is retained as the incumbent, and its entries are protected from deletion. A short history record lists the previous six rounds' scores and skill entries. Executable reflection is invoked after two rounds without a new best probe score; after five such rounds the seed closes.

Cross-round review. C-Refl follows when the selected library scores below 0.90 on validation, or when no library was delivered but at least three distinct training items were solved and bare validation accuracy is below 0.90. Each such task receives one C-Refl run of up to 12 rounds. It works from the saved libraries and execution records of the seeds: for each seed it compares successive library versions, summarizes their score trajectories and the edits with the largest score changes, and traces the edits that introduced or modified the selected library's rules; it also notes rules that recur across seeds. Each round classifies the remaining failures, analyzes the leading error types, and extracts formulas, thresholds, and operating steps from successful programs, checking candidates in the sandbox on solved and failed training cases. From this evidence it proposes local edits, alternative edits, or a complete `main-procedure`; accepted and rejected edits, residual failures, and regressions inform later rounds.

Selection and delivery. At the end of a seed, versions that improve on bare probe accuracy by at least 3 percentage points are ranked by probe score. The highest-ranked version, followed by the runner-up only if needed, is evaluated on D_{val}. A library qualifies for delivery if

$$\hat{J}_{\mathrm{val}}(L) \;\geq\; \hat{J}_{\mathrm{val}}(\varnothing) + \gamma, \qquad \gamma = 0.03. \tag{5}$$

Here $\hat{J}_{\mathrm{val}}(L)$ is the empirical accuracy of library L on D_{val}, $\varnothing$ denotes no library, and γ is the minimum validation gain for delivery. The first qualifying candidate is retained within a seed, and the qualifying library with the highest validation score is chosen across seeds.

Limits. The split sizes are $|D_{\mathrm{tr}}| = 60$, $|D_{\mathrm{probe}}| = 32$ (8 per difficulty tier), and $|D_{\mathrm{val}}| = 50$. Open executors use up to 12 reflection rounds and 3 seeds; luna uses 8 rounds and one seed. Executable sessions allow up to 10 investigation steps followed by a final edit request. Solver episodes keep the default budget of Appendix A (5 rounds, 40,960 tokens).

Skill-optimizer baselines. TextGrad, GEPA, and SkillOpt use exactly our setting: the same frozen solver, the same training, validation, and test instances, and the same seeds per task (three on the open LVLMs and one on luna); their skills enter the solver's system prompt in the same way as ours. Budgets are measured as the total number of LVLM inference calls during skill optimization. TextGrad receives 1.00–1.06× our budget, GEPA 1.05–1.12×, and SkillOpt 1.01–1.15×; the extended SkillOpt rows in Table 2 receive about 1.6×. For every method, the library with the highest validation accuracy is delivered, and a task without a qualifying library uses bare Code-with-Image.

G.5 Retrospective analysis of reflection rounds

Figure 3 uses the archived validation evaluations of the reflection rounds. Within each seed, validation scores are obtained at the finale for the probe winner, and for the runner-up only if needed (selection rule in Appendix G.4). For each task, the analysis first selects the seed using its full-run validation result. Within that seed, it retains the highest-scoring archived version that passes the +3pp gate and was authored by round $R = 8$, if one exists.

Index tasks by $i = 1, \ldots, N$, with $N = 30$. Let a_i be the retained version's authoring round, and let e_i^0 and e_i be the bare solver's and retained version's validation accuracies as fractions. For a retained version, define its normalized gain as $g_i = (e_i - e_i^0)/(1 - e_i^0)$. If no version qualifies, set $g_i = 0$ and $a_i = \infty$; this also covers tasks with $e_i^0 = 1$, which cannot pass the positive-gain gate. The plotted attribution is

$$G(r) = \frac{1}{N} \sum_{i=1}^{N} g_i \, \mathbf{1}[a_i \le r], \tag{6}$$

where $r = 0, \ldots, R$ and $\mathbf{1}[\cdot]$ is the indicator function. Each retained version's gain is credited from its authoring round onward.

Panel (b) of Figure 3 reports test accuracy for runs in which the solver is fixed to Qwen3.5-9B and only the reflector changes: the observational and executable reflection calls, including the investigation code the reflector writes and runs in E-Refl, go to Qwen3.5-9B itself, Qwen3.5-27B, gemma-4-26B, or GPT-5.6-luna, while the solver's probe and validation runs stay on 9B as in the main pipeline. Each arm runs one seed per task from the same bare-9B round 0. The 27B library of Table 4 was refined over three seeds with C-Refl; the swap arms use one seed without it. The plotted test accuracy averages, over the 30 tasks, the delivered library's test score, or the bare solver's where no version passed the gate.

G.6 Why cross-round reflection is needed

The O-Refl–E-Refl loop ends because it stops improving, not because its budget runs out: 83–92% of its seeds on the open executors stop after five rounds without a new best training-probe score, so a larger round budget would go unused (Table 12). C-Refl continues from the selected library. It runs at most once per task and takes about a quarter of the pipeline's budget, roughly the price of one more seed. A further seed would buy less: each seed adds less than the one before, and C-Refl adds as much as the second and third seeds together. Seed gains are changes in the validation accuracy of the best of k seeds, averaged over all subsets of size k. On held-out test instances, C-Refl adds 7.4, 6.1, and 8.7 points on 9B, gemma-26B, and 27B (Table 3).

Table 12: **C-Refl against further seeds.** Early stop: share of O-Refl–E-Refl seeds that stop after five rounds without a new best training-probe score. C-Refl budget: share of the pipeline's LVLM inference calls used in C-Refl. Gains: validation points added by the first, second, and third seed, and by C-Refl.

			Validation points added by			
Executor	Early stop	C-Refl budget	1st seed	2nd seed	3rd seed	C-Refl
Qwen3.5-9B	92%	28%	+12.2	+5.0	+2.9	**+8.5**
gemma-4-26B	83%	25%	+15.3	+3.7	+1.8	**+6.5**
Qwen3.5-27B	91%	19%	+19.7	+4.8	+2.9	**+7.7**
Average	89%	24%	+15.7	+4.5	+2.5	**+7.6**

G.7 Ablating executable reflection: validation uplift

Table 13: **Ablating executable reflection.** O-Refl replaces escalated rounds with further observational reflection; both arms otherwise run the same loop, 3 independent seeds per task, for the three open executors in bare-Code-with-Image order. Uplift is the accuracy gain over bare Code-with-Image on the validation split, in percentage points; a run that delivers no skill counts 0. Mean uplift averages the three seeds. Best-seed uplift keeps, for each task, the seed with the best validation score, the selection rule of the O-Refl–E-Refl loop (defined in Appendix G.4). Skills delivered is the share of the 30×3 task×seed runs that ship a skill. Δ: E-Refl minus O-Refl; **bold**: the larger of the two arms.

Metric	**Arm**	**Task family**				**Overall**
		Transformation	Perception	Localization	Color	
Qwen3.5-9B						
Mean uplift (pp)	E-Refl	**+12.2**	**+12.3**	**+16.4**	**+6.1**	**+12.2**
	O-Refl	+8.1	+11.4	+11.8	+5.6	+9.2
	Δ	+4.1	+0.9	+4.6	+0.5	+3.0
Best-seed uplift (pp)	E-Refl	**+21.3**	**+21.3**	**+25.7**	+7.6	**+20.1**
	O-Refl	+10.2	+19.0	+18.3	**+8.8**	+13.6
	Δ	+11.1	+2.3	+7.4	−1.2	+6.5
Skills delivered (%)	E-Refl	**36.1**	**77.8**	**57.1**	40.0	**50.0**
	O-Refl	33.3	72.2	38.1	40.0	43.3
	Δ	+2.8	+5.6	+19.0	+0.0	+6.7
gemma-4-26B						
Mean uplift (pp)	E-Refl	**+14.2**	**+18.9**	**+22.5**	**+3.7**	**+15.3**
	O-Refl	+8.6	+14.7	+20.0	+1.9	+11.4
	Δ	+5.6	+4.2	+2.5	+1.8	+3.9
Best-seed uplift (pp)	E-Refl	**+18.2**	**+28.3**	**+29.4**	**+6.4**	**+20.9**
	O-Refl	+12.3	+19.0	+23.1	+3.6	+14.7
	Δ	+5.9	+9.3	+6.3	+2.8	+6.2
Skills delivered (%)	E-Refl	**52.8**	**94.4**	**76.2**	**26.7**	**62.2**
	O-Refl	27.8	55.6	61.9	13.3	38.9
	Δ	+25.0	+38.8	+14.3	+13.4	+23.3
Qwen3.5-27B						
Mean uplift (pp)	E-Refl	**+20.2**	+17.3	**+32.7**	**+3.5**	**+19.7**
	O-Refl	+7.3	**+20.7**	+28.9	+0.0	+13.8
	Δ	+12.9	−3.4	+3.8	+3.5	+5.9
Best-seed uplift (pp)	E-Refl	**+29.7**	+20.7	**+41.7**	**+10.4**	**+27.5**
	O-Refl	+15.0	**+29.0**	+36.6	+0.0	+20.3
	Δ	+14.7	−8.3	+5.1	+10.4	+7.2
Skills delivered (%)	E-Refl	**75.0**	72.2	**95.2**	**20.0**	**70.0**
	O-Refl	55.6	**83.3**	85.7	0.0	58.9
	Δ	+19.4	−11.1	+9.5	+20.0	+11.1