BEACON: KNOWING WHEN AND HOW TO PERFORM AGENTIC VISUAL REASONING

Anonymous authors
Paper under double-blind review

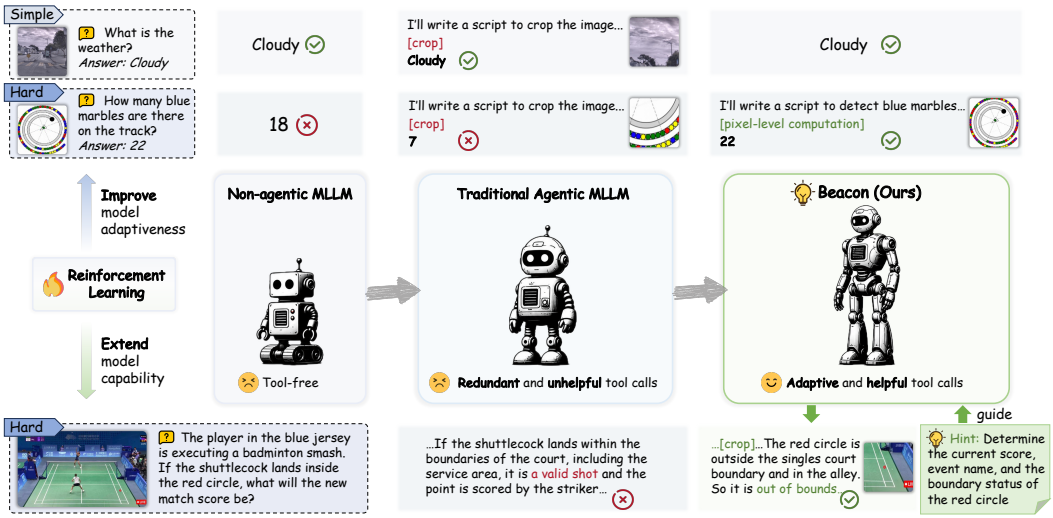
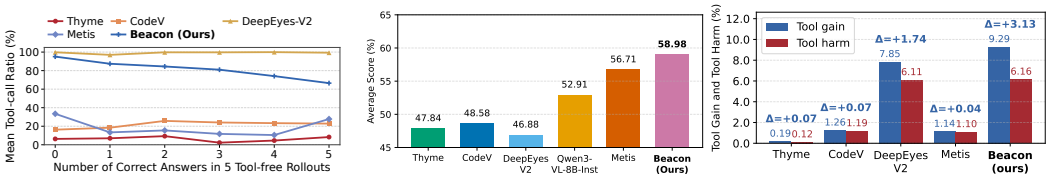


Figure 1: **Agentic visual reasoning models should use tools *adaptively* and *effectively*.**

ABSTRACT

The fundamental goal of agentic visual reasoning is to improve the success rate of multimodal large language models (MLLMs) on complex tasks. In this work, we rethink agentic visual reasoning through two key dimensions of tool use: ***Mode Adaptiveness*** and ***Tool Effect***. *Mode Adaptiveness* characterizes whether an MLLM can recognize when tools are truly necessary and invoke them accordingly, thereby avoiding unnecessary computational overhead while improving performance on challenging problems that require tool assistance. *Tool Effect* characterizes the actual impact of tool use: tools should extend the model's capabilities on problems unsolvable through tool-free reasoning, while avoiding additional errors on problems that the model can already solve without tools. We conduct a comprehensive analysis to quantify these two properties and empirically reveal that existing agentic visual reasoning models exhibit limited *Mode Adaptiveness*, while the gains produced by tool use on hard examples are largely offset by the harm introduced on easy examples that the models can already solve. Motivated by these observations, we propose **Beacon**, a novel agentic visual reasoning model trained with supervised fine-tuning (SFT) and reinforcement learning (RL). At the core of Beacon are the *Necessity-Aware Adaptive Reward* and the *Hint-Guided Capability Expansion* mechanism in the RL stage. *Necessity-Aware Adaptive Reward* encourages tool-free solutions when they succeed while preserving full reward for successful tool use when tool-free rollouts fail. Hint-Guided Capability Expansion uses verified, answer-free expert hints to recover learning signals from all-wrong rollout groups, aiming at extending the tool-use capability on the hardest problems. Across 13 benchmarks, Beacon achieves the highest average score among the evaluated open-source models and ranks first on 11 benchmarks. On five diagnostic benchmarks, it improves the average tool-available accuracy over its tool-free accuracy by 1.96 points and achieves the largest tool-gain minus tool-harm score (+3.14 points). These results show Beacon's advanced performance, *Mode Adaptiveness*, and the net benefit of tool use.

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.



(a) (b) (c)

Figure 2: **(a)** The tool-call ratio on samples with different levels of tool-free accuracy. **(b)** Average performance across 13 benchmarks. **(c)** Tool-induced performance gains and harms (see Section 2.1 for details). **Takeaway 1**: Beacon exhibits adaptive tool-invocation behavior that is largely absent from existing agentic visual reasoning models, invoking tools more frequently on samples that are difficult to solve through tool-free reasoning (Figure 2a). **Takeaway 2**: Beacon achieves the best average performance and the largest gap between tool gain and tool harm (Figure 2b and 2c), indicating that its tool use more effectively extends the model's capabilities beyond tool-free reasoning. In contrast, for prior models, the gains from tool use are largely offset by the errors it introduces, resulting in little improvement over tool-free reasoning.

1 Introduction

Agentic visual reasoning (Zhang et al., 2026; Yan et al., 2026; Hong et al., 2026; Hou et al., 2026), which leverages external tools to produce intermediate multimodal results that support final-answer generation, has emerged as an important paradigm for multimodal large language models (MLLMs). Although prior studies have demonstrated empirical gains, they often overlook two critical aspects of tool use (see Figure 2): (1) *tool-invocation adaptiveness*: whether a model can autonomously determine when tools are necessary based on factors such as tool-free solvability; and (2) *tool effect*: whether tool use extends the model's capabilities on problems that cannot be solved without tools while avoiding performance degradation on problems that are solvable through tool-free reasoning.

To investigate these aspects, we conduct a comprehensive evaluation of representative agentic visual reasoning models. Our analysis reveals that existing models generally exhibit limited adaptiveness in tool invocation, while the gains from tool use are often offset by the errors it introduces, yielding little improvement over tool-free reasoning. Motivated by these findings, we propose **Beacon**, an agentic visual reasoning model with improved tool-invocation adaptiveness (Figure 2a) and stronger tool-use capability, enabling genuine gains beyond tool-free reasoning (Figure 2c).

During inference, Beacon can autonomously generate Python code to perform diverse visual and numerical operations, including image cropping, annotation, enhancement, rotation, stitching, pixel-level computation, and complex numerical calculations. To train Beacon, we first construct a high-quality data synthesis pipeline that equips the model with fundamental code-use capabilities. We then introduce a reinforcement learning (RL) framework consisting of a necessity-aware adaptive reward, which encourages tool use only when needed, and a hint-guided rollout strategy, which exposes the model to effective tool-use trajectories on problems that cannot be solved without tools.

Experiments on 13 benchmarks show that Beacon achieves the best average performance among open-source models. Further analysis demonstrates that Beacon exhibits stronger tool-invocation adaptiveness and the largest gap between tool-induced gain and harm, indicating that our RL framework improves both the adaptiveness and effectiveness of tool use. We compare our work with previous works in Appendix A.

We summarize our contributions as follows:

- We analyze tool-invocation adaptiveness and the effects of tool use in representative agentic visual reasoning models. Our results reveal that existing models generally lack adaptive tool-use behavior, while the gains achieved on hard problems are largely offset by the errors introduced on problems that are already solvable through tool-free reasoning (Section 2.1).
- We develop a high-quality supervised fine-tuning (SFT) data synthesis pipeline, an RL data curation strategy, and a novel RL framework to train Beacon. These designs improve the model's tool-invocation adaptiveness and strengthen its ability to use tools to extend its capabilities beyond tool-free reasoning (Section 3).

- We evaluate Beacon on 13 benchmarks spanning diverse visual reasoning tasks. Beacon achieves the best average performance and exhibits stronger tool-invocation adaptiveness, together with the largest margin between tool-induced gain and harm, demonstrating that our RL framework improves both the adaptiveness and effectiveness of tool use (Section 4).

2 ANALYSIS

In this section, we conduct comprehensive experiments to analyze two different aspects of the tool use of agentic visual models: *Mode Adaptiveness* and *Tool Effect*.

2.1 THE PROPOSED METRICS

***Mode Adaptiveness* (*MA*)** measures whether a model uses tools less when a problem can be reliably solved through tool-free reasoning, and invokes tools when tool-free reasoning is insufficient.

To assess whether a problem can be reliably solved through tool-free reasoning, we require the model to answer using Prompt 1 (denoted as p_{text}) in Appendix B.1, which prohibits tool use. Unlike prior analyses that determine tool-free solvability from a single response (Guo et al., 2026), a procedure that can be sensitive to sampling stochasticity, we sample five tool-free responses for each problem. We classify a problem as "text-easy" if at least four responses are correct and as "text-hard" if at most one response is correct. Problems with two or three correct responses are regarded as ambiguous and excluded from the analysis. For tool-enabled reasoning, we adopt the official tool-use prompt of each model, denoted as p_{tool}. We minimally modify these prompts to permit, rather than mandate, tool use, thereby revealing each model's intrinsic mode-adaptation behavior.

Let x denote a problem sample and π_θ denote the model. Denote the "text-easy" and "text-hard" samples as $\mathcal{E}$ and $\mathcal{H}$, respectively. Let $u_{x,i}$ denote the usage of tools for the i-th answer of sample x under the tool-enabled prompt p_{tool}, i.e., $u_{x,i} = 1$ if the model uses tools for the i-th answer of sample x and $u_{x,i} = 0$ otherwise. *MA* consists of the following sub-metrics.

$$\textit{MA}_{\textit{text}} = \frac{1}{5|\mathcal{E}|} \sum_{x \in \mathcal{E}} \sum_{i=1}^{5} (1 - u_{x,i}), \qquad \textit{MA}_{\textit{tool}} = \frac{1}{5|\mathcal{H}|} \sum_{x \in \mathcal{H}} \sum_{i=1}^{5} u_{x,i}.$$

$\textit{MA}_{\textit{text}}$ measures the average ratio of tool-free responses on the "text-easy" samples. A higher value indicates that the model is better at avoiding unnecessary tool use. $\textit{MA}_{\textit{tool}}$ measures the average ratio of tool-assisted responses on the "text-hard" samples. A higher value indicates that the model is better at invoking tools when tool-free reasoning fails frequently. We define $\textit{MA}_{\text{mean}} = \frac{\textit{MA}_{\text{text}} + \textit{MA}_{\text{tool}}}{2}$.

***Tool Effect* (*TE*)** measures how tool use affects the model's task-solving performance compared to tool-free reasoning, including both the gains obtained on "text-hard" samples and harms introduced on "text-easy" samples. Let $c_{x,i}$ denote the correctness of the i-th response for sample x, i.e., $c_{x,i} = 1$ if the model answers correctly and $c_{x,i} = 0$ otherwise. Denote the total number of test samples as N. *Tool Effect* consists of the following sub-metrics:

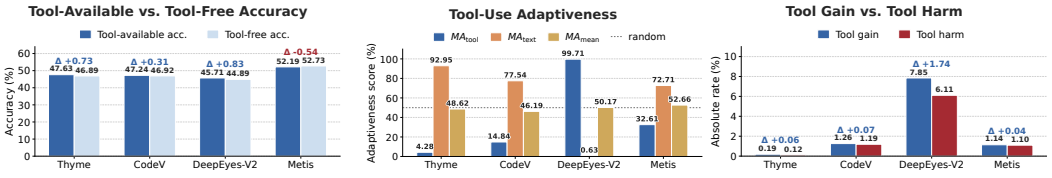
$$\textit{Tool-Gain} = \frac{1}{5N} \sum_{x \in \mathcal{H}} \sum_{i=1}^{5} u_{x,i} c_{x,i}, \qquad \textit{Tool-Harm} = \frac{1}{5N} \sum_{x \in \mathcal{E}} \sum_{i=1}^{5} u_{x,i} (1 - c_{x,i}).$$

Tool-Gain measures the fraction of all evaluation responses that both invoke tools and correctly answer a text-hard problem under the tool-enabled prompt p_{tool}. A higher value indicates that tool use is more effective at extending the model's problem-solving capability beyond tool-free reasoning. *Tool-Harm* measures the fraction of all evaluation responses that invoke tools but incorrectly answer a text-easy problem under p_{tool}. A lower value indicates that the model is better at avoiding performance degradation caused by ineffective tool use.

Accuracy metrics. We also compute several other metrics to better depict the behavior of agentic visual reasoning. To measure the overall performance of the two reasoning modes, we compute the accuracy under the tool-enabled prompt p_{tool} and the enforced-text prompt p_{text} averaged over five runs, denoted as *Tool-Available Acc.* and *Tool-Free Acc.*, respectively.

2.2 Results and Analysis

We evaluate several representative agentic visual reasoning models: Thyme (Zhang et al., 2026), DeepEyesV2 (Hong et al., 2026), CodeV (Hou et al., 2026), and Metis (Yan et al., 2026), on five benchmarks: HRBench4K (Wang et al., 2025c), BLINK (Fu et al., 2024), BabyVision (Chen et al., 2026), MathVista (Lu et al., 2024), and TIRBench (Li et al., 2025). We present the overall results of the metrics averaged over these five benchmarks in Figure 3 and summarize the key observations below. We leave the detailed results and analysis in Section 4.3.



(a) (b) (c)

Figure 3: **(a)** The accuracy of tool-available and tool-free reasoning. **(b)** The MA_{text}, MA_{tool}, and MA_{mean} of different models. **(c)** The *Tool-Gain* and *Tool-Harm* of different models.

Observation 1 (Figure 3a): Tool-enabled reasoning provides little improvement over tool-free reasoning for most existing agentic visual reasoning models. This finding is consistent with prior studies (Guo et al., 2026; Yang et al., 2026). By averaging results over five runs, we further consolidate this finding and show that this limited gain is robust to sampling stochasticity.

Observation 2 (Figure 3b): Existing agentic visual reasoning models exhibit limited reasoning-mode adaptiveness. A model that always uses tools or never uses tools achieves an MA_{mean} of 50%. Most evaluated models obtain an MA_{mean} close to or even below this baseline. Moreover, many consistently favor a single reasoning mode, either tool-free or tool-assisted reasoning, indicating that they fail to adapt their mode selection to the tool-free difficulty of each problem.

Observation 3 (Figure 3c): Tool use provides limited benefit on "text-hard" samples for existing agentic visual reasoning models. Most evaluated models exhibit a low *Tool-Gain*. Although their *Tool-Harm* is also limited, *Tool-Gain* does not substantially exceed *Tool-Harm*. Consequently, tool use yields little overall performance benefit (Figure 3a).

Hypothesis of issue attribution. We hypothesize that this issue arises from two primary factors: (1) the absence of an explicit objective that promotes adaptive behavior, and (2) a fundamental limitation of RL with verifiable rewards—its difficulty in expanding the model's capabilities beyond those acquired from the pretraining distribution.

3 Method

3.1 Overview

Motivated by the observations in Section 2.2, we propose Beacon, which adopts an SFT-then-RL training paradigm. During SFT, we equip the model with fundamental code-use capabilities using synthesized trajectories spanning diverse visual reasoning tasks. During RL, we introduce adaptive reward shaping and a hint-guided rollout mechanism to further improve reasoning-mode adaptiveness and enhance the genuine performance gains brought by code use.

Given a visual query consisting of an image and a question, Beacon autonomously decides whether code execution is necessary. When code is invoked, the model generates a Python snippet enclosed in `<tool_call>...</tool_call>` to manipulate the image or perform numerical computation. After execution, the resulting output is returned as an observation enclosed in `<tool_response>...</tool_response>`. Then the model generates an observation about the tool results `<observation>...</observation>` and continue reasoning, invoke code again, or produce the final answer.

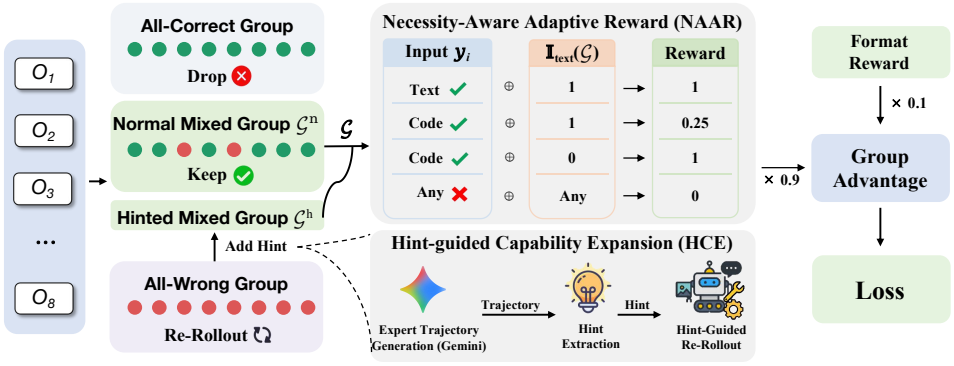


Figure 4: **Illustration of the RL process of Beacon.** The *Necessity-Aware Adaptive Reward* (NAAR) and *Hint-Guided Capability Expansion* (HCE) are designed to improve reasoning-mode adaptiveness and expand the tool-use capability on the most challenging problems, respectively.

3.2 Training Data Construction

Overview. We construct a training data pool from open-source datasets. For SFT, we first evaluate the base model, Qwen3-VL-8B-Instruct (Bai et al., 2025a), on the source data and retain challenging examples. We then use Gemini 3.1 Pro (Google DeepMind, 2026) to generate code-assisted reasoning trajectories for these examples and further refine the generated trajectories with the same model, obtaining about 15K code-assisted trajectories. For RL, we evaluate the resulting SFT model on the same data pool and select the hard examples for reinforcement learning. Statistics of the constructed training data are reported in Table 5, with additional details provided in Appendix C.

3.3 Cold-start SFT

We use a standard cross-entropy loss for cold-start SFT, and mask the code output (i.e., the content between `<tool_response>...</tool_response>`). To avoid biasing the model towards code use and mitigate forgetting, we inject 30K correct pure-text trajectories generated by the base model for the samples with accuracy ≥ 0.6 among five rollouts into the $\sim$15K code-assisted trajectories, obtaining $\sim$45K SFT data. See Appendix D.1 for more details of SFT training.

3.4 Reinforcement Learning

We adopt GRPO (Shao et al., 2024) as the base RL algorithm, with our designed *Necessity-Aware Adaptive Reward* and *Hint-guided Capability Expansion* mechanisms to further improve the model's reasoning mode adaptiveness and the true gain of code use.

3.4.1 Necessity-Aware Adaptive Reward

A central challenge in adaptive reasoning is to discourage unnecessary tool use without suppressing the model's ability to solve difficult problems with code. A strict binary preference for tool-free reasoning may over-penalize code-based responses even when they produce correct answers, which can introduce conflicts between the adaptive objective and the task-accuracy objective. To balance reasoning efficiency and task success, we introduce the *Necessity-Aware Adaptive Reward* (NAAR), which prioritizes tool-free reasoning when it is sufficient, while still assigning partial credit to correct code-based solutions.

For each rollout group $\mathcal{G}$, let $\mathbb{I}_{\text{text}}(\mathcal{G})$ indicate whether the group contains at least one correct tool-free response. The adaptive reward for a response y_i is defined as

$$R_{\text{adaptive}}(y_i;\mathcal{G}) = \begin{cases} 1, & \mathbb{I}_{\text{text}}(\mathcal{G}) = 1,\ y_i \text{ is tool-free and correct,} \\ 0.25, & \mathbb{I}_{\text{text}}(\mathcal{G}) = 1,\ y_i \text{ uses code and is correct,} \\ 1, & \mathbb{I}_{\text{text}}(\mathcal{G}) = 0,\ y_i \text{ uses code and is correct,} \\ 0, & \text{otherwise.} \end{cases} \tag{1}$$

When a problem can be solved through tool-free reasoning, NAAR assigns the highest reward to correct tool-free responses and a reduced reward to correct code-based responses, thereby encouraging the model to avoid unnecessary tool calls. Importantly, correct code-based responses are not assigned zero reward, since they still demonstrate successful reasoning and may provide useful learning signals. When no tool-free response in the group is correct, correct code-based responses receive the full reward, encouraging the model to use tools when they are necessary for solving the problem. Therefore, NAAR implements a soft preference for tool-free reasoning rather than a hard prohibition on tool use, preserving task-solving capability while improving necessity awareness.

We use the same prompt for SFT, RL rollouts, and evaluation to avoid distribution shift (see Appendix D.3 for the prompt), which instructs the model to autonomously decide whether to use code.

3.4.2 HINT-GUIDED CAPABILITY EXPANSION

A fundamental limitation of conventional RLVR is that hard examples are often "wasted" when the policy fails to solve them within a limited number of rollouts. In this case, all responses receive similarly low rewards and thus provide little useful group-relative learning signal. Nevertheless, these hard examples are particularly valuable for expanding the capability boundary of the policy, which aligns with the goal of tool use. To better exploit such examples, we propose *Hint-guided Capability Expansion* (HCE), which injects expert-generated hints into the rollout process to help the policy discover successful reasoning and tool-use trajectories on the hard examples.

Concretely, for each problem, we first sample a group $\mathcal{G}^{\mathrm{n}} = \{y_i^{\mathrm{n}}\}_{i=1}^{N}$ from the policy using the original prompt x, where $N = 8$ in our experiments. If none of these responses is correct, we regard the problem as a hard example and invoke a strong expert model (Gemini-3.1-Pro) to construct an answer-free hint. The generated hint is then injected into the rollout prompt, allowing the policy to explore reasoning and tool-use behaviors that are difficult to discover from its original policy distribution. During policy optimization, we remove the hint from the model input while retaining the trajectories generated with the hint, thereby transferring the hint-assisted behaviors back to the original, hint-free policy and avoiding reliance on hints at inference time.

Hint generation. Directly prompting the expert model to produce a hint may result in unreliable guidance for two reasons. First, the expert model may itself fail to solve the problem correctly. Second, hint steps produced without actual execution may be impractical or unhelpful. We therefore adopt a two-stage hint-generation procedure. First, the expert model is prompted to solve the hard example by producing a complete code-assisted reasoning trajectory, and we retain only the correct ones. Second, the expert model extracts the crucial reasoning steps (text or code-use steps) and the expected subgoal of each step from the verified trajectory and converts them into an answer-free hint. Details of the hint-generation prompts are provided in Appendix D.4.

Hinted rollout. Let h denote the hint generated for a hard example with original question prompt x. We append the hint to the original question to construct the hinted prompt $x^{\mathrm{h}} = x \oplus h$, where $\oplus$ denotes prompt concatenation. The hint contains only intermediate instructions, expected subgoals, and useful tool-use strategies; it does not contain the final answer. We then sample another group of N responses from the policy model π_θ under the hinted prompt x^{h}: $y_i^{\mathrm{h}} \sim \pi_\theta\left(\cdot \mid x^{\mathrm{h}}\right)$ and get the hinted group $\mathcal{G}^{\mathrm{h}} = \left\{y_i^{\mathrm{h}}\right\}_{i=1}^{N}$. The hinted groups $\mathcal{G}^{\mathrm{h}}$ are combined with normal groups $\mathcal{G}^{\mathrm{n}}$, i.e., groups sampled from the original prompt x for subsequent policy optimization. For the hinted groups, when computing the policy update, we preserve the generated trajectory y_i^{h} but replace x^{h} with the original prompt x to avoid the model relying on hints at inference time.

3.4.3 POLICY OPTIMIZATION WITH NORMAL AND HINTED ROLLOUTS

Let $\mathcal{B}_{\mathrm{n}}$ and $\mathcal{B}_{\mathrm{h}}$ denote the sets of retained responses from normal and hinted groups, respectively. For either type of group $\mathcal{G}$, we combine the format reward R_{format} with the previously defined necessity-aware adaptive reward:

$$R(y_i; \mathcal{G}) = 0.1 R_{\mathrm{format}}(y_i) + 0.9 R_{\mathrm{adaptive}}(y_i; \mathcal{G}), \tag{2}$$

in which the format reward is defined as $R_{\mathrm{format}}(y_i) = 1$ if each code block is followed by a corresponding observation block and the final answer is put in `<answer>...</answer>`, and

$R_{\text{format}}(y_i) = 0$ otherwise. For each group, we compute the group-relative advantage for response y_i as $A_i = \frac{R(y_i;\mathcal{G}) - \mu_{\mathcal{G}}}{\sigma_{\mathcal{G}} + 1e{-}6}$, where $\mu_{\mathcal{G}}$ and $\sigma_{\mathcal{G}}$ are the mean and standard deviation of the rewards in group $\mathcal{G}$, respectively.

Let $y_{i,t}$ be the t-th trainable token and let $\tau_{i,<t}$ denote the preceding interaction history, including previous model tokens and environment observations. We use different importance-sampling ratios for normal and hinted groups. For a response in a normal group, the importance-sampling ratio is $\rho^{\text{n}}_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}|x,\tau_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x,\tau_{i,<t})}$, where θ_{old} denotes the model parameter before the update. For a hinted response, we use the following importance sampling ratio: $\rho^{\text{h}}_{i,t}(\theta) = \frac{\pi_\theta(y_{i,t}|x,\tau_{i,<t})}{\pi_{\theta_{\text{old}}}(y_{i,t}|x^{\text{h}},\tau_{i,<t})}$. By removing the hint from the input prompt, we transfer the capability obtained through hint-guided exploration into the hint-free policy. Following Nath et al. (2025), we retain the hint in the context of the old policy to mitigate off-policy issues and stabilize training.

Final Training Objective. The final actor loss follows the standard clipped GRPO objective:

$$\mathcal{L}_{\text{GRPO}}(\theta) = -\frac{1}{Z} \sum_{b\in\{\text{n,h}\}} \sum_{y_i\in\mathcal{B}_b} \sum_{t\in\mathcal{T}_i} \min\left(\rho^b_{i,t}(\theta)A_i,\ \text{clip}\left(\rho^b_{i,t}(\theta), 1-\epsilon, 1+\epsilon\right) A_i\right), \tag{3}$$

where $\epsilon = 0.2$ and $Z = \sum_{b\in\{\text{n,h}\}} \sum_{y_i\in\mathcal{B}_b} |\mathcal{T}_i|$. Here, $\mathcal{T}_i$ contains only trainable tokens and excludes tool responses.

4 Experiments

4.1 Experimental Setup

Training details. We take Qwen3-VL-8B-Instruct (Bai et al., 2025a) as the base model. For SFT, we train the model for 4 epochs with a peak learning rate of $1e^{-5}$. For the RL stage, we train the model for 1 epoch with a learning rate of $1e^{-6}$. See Appendix D for more details.

Evaluation Benchmarks. As Yang et al. (2026) suggested, the evaluation for agentic visual reasoning models should consider challenging benchmarks that necessitate visual tool use. Therefore, we evaluate our model on a wide range of tasks, including high-resolution visual search: V* (Wu & Xie, 2024), HRBench (Wang et al., 2025c), Visual Probe (Lai et al., 2025) (hard); spatial and perceptual reasoning: RealWorldQA (xAI, 2024), BLINK (Fu et al., 2024), BabyVision (Chen et al., 2026); quantitative and diagrammatic reasoning: ChartQAPro (Masry et al., 2025), MathVista (Lu et al., 2024), MathVision (Wang et al., 2024), compositional and agentic visual reasoning: VisualPuzzles (Song et al., 2025b), GameQA (Tong et al., 2025), TIRBench (Li et al., 2025). These benchmarks go beyond simple object recognition or local perception, requiring models to perform complex visual search, spatial transformation, diagrammatic interpretation, intermediate-state imagination, etc.

Baselines. We compare our model with both strong open-source and closed-source models. For open-source models, we include Qwen3-VL-8B-Instruct (Bai et al., 2025a), PixelReasoner (Su et al., 2025a), Thyme (Zhang et al., 2026), DeepEyesV2 (Hong et al., 2026), CodeV (Hou et al., 2026), AdaTooler-V (Wang et al., 2026), PyVision-RL (Zhao et al., 2026), and Metis (Yan et al., 2026). For closed-source models, we include Gemini3.1-pro (Google DeepMind, 2026).

4.2 Main Results

The evaluation results are presented in Tables 1 and 2. Beacon-8B achieves the highest average performance among all open-source models, ranking first on 11 of the 13 benchmarks and outperforming its base model by an average of 6.07 points.

4.3 Analysis

In this section, we present detailed results and analysis of *Mode Adaptiveness* and *Tool Effect*, which are evaluated on the same five benchmarks as Section 2.2. Among samples correctly answered under p_{text} and answered without tools under p_{tool}, we define *Text-Retain* as the fraction that remain correct

Table 1: **Comparison on high-resolution visual search, spatial, and perceptual reasoning benchmarks.** Avg. denotes the unweighted mean across the seven benchmarks and is reported only for models with complete results. **Bold** and underline indicate the best and second-best performance among open-source models, respectively. Gray indicates that the result is reported in the original paper.

Model	High-Resolution Visual Search				Spatial & Perceptual Reasoning			Avg.
	V*	HR-Bench 4K	HR-Bench 8K	VisualProbe	RealWorldQA	BLINK	BabyVision	
Closed-Source Models								
Gemini 3.1 Pro	89.01	90.75	86.38	40.57	79.22	81.22	48.45	73.66
Open-Source Models								
Qwen3-VL-8B-Instruct	84.82	78.13	75.75	37.74	72.29	62.86	12.37	60.57
Pixel-Reasoner-7B	78.01	71.25	67.75	33.96	70.20	55.97	11.86	55.57
Thyme-7B	82.72	77.50	72.13	47.17	70.59	54.81	14.43	59.91
DeepEyesV2-7B	80.63	75.62	65.88	32.08	62.35	54.29	14.18	55.00
CodeV-7B	84.29	75.62	67.50	40.57	70.59	58.13	12.37	58.44
AdaTooler-V-7B	89.8	–	–	–	–	–	–	–
PyVision-7B	88.7	78.1	74.3	–	–	–	–	–
Metis-8B	**90.58**	83.50	81.63	48.11	71.90	62.65	14.18	64.65
Beacon-8B (Ours)	89.01	**84.25**	**81.88**	**50.00**	**73.20**	**65.97**	**18.04**	**66.05**
Δ over Qwen3-VL-8B-Inst.	+4.19	+6.12	+6.13	+12.26	+0.91	+3.11	+5.67	+5.48

Table 2: **Comparison on quantitative, diagrammatic, compositional, and agentic visual reasoning benchmarks.** Averages are computed over all available reported results.

Model	Quantitative & Diagrammatic Reasoning			Compositional & Agentic Visual Reasoning			Avg.
	ChartQAPro	MathVista	MathVision	VisualPuzzles	GameQA	TIR-Bench	
Closed-Source Models							
Gemini 3.1 Pro	75.46	90.40	89.47	73.46	81.00	47.57	76.23
Open-Source Models							
Qwen3-VL-8B-Instruct	41.66	76.40	53.82	36.22	36.70	19.01	43.97
Pixel-Reasoner-7B	51.03	70.80	28.09	33.90	28.10	16.71	38.11
Thyme-7B	38.66	68.90	25.82	24.14	26.00	19.01	33.76
DeepEyesV2-7B	50.92	67.30	27.30	34.33	27.50	17.04	37.40
CodeV-7B	46.06	69.80	26.02	34.85	28.00	17.70	37.07
AdaTooler-V-7B	–	74.5	–	–	–	–	–
PyVision-7B	–	–	28.7	–	–	19.8	–
Metis-8B	52.79	**77.80**	53.49	40.50	38.50	21.65	47.46
Beacon-8B (Ours)	**58.48**	77.10	**54.57**	**42.89**	**47.30**	**24.03**	**50.73**
Δ over Qwen3-VL-8B-Inst.	+16.82	+0.70	+0.75	+6.67	+10.60	+5.02	+6.76

under p_{tool}. A higher value indicates greater robustness of tool-free reasoning across prompts. The results are shown in Table 3. We summarize the key observations below.

Observation 1: Beacon can further improve its already strong tool-free reasoning by using tools. When tools are available, Beacon consistently outperforms tool-free reasoning across all datasets and shows the largest average performance gain Δ_{Acc} (+1.96%), while other models exhibit weaker average Δ_{Acc} with tools (below +1%). Note that the tool-free reasoning of Beacon still shows clear improvements over the base model (51.57%, compared to the average accuracy of Qwen3-VL-8B-Instruct on these five datasets in Table 1 and 2, which is 49.75%), indicating that the clear performance gain of the tool use of Beacon does not come at the cost of weakening the tool-free reasoning capability.

Observation 2: Beacon shows the best reasoning-mode adaptiveness among evaluated models. Although Beacon shows a preference for using code (the MA_{text} of Beacon is relatively low), it exhibits the best average MA_{mean}, demonstrating that it can adaptively choose the appropriate reasoning mode for different problems.

Observation 3: Beacon achieves the largest gap between Tool-Gain and Tool-Harm, along with the highest *Text-Retain*. Beacon invokes tools more frequently than most baselines, achieving the highest Tool-Gain while also incurring the highest Tool-Harm. Its greater success on text-hard samples outweighs the errors in tool-assisted responses on text-easy samples, yielding the largest

Table 3: **The metrics that reflect the *Mode Adaptiveness* and *Tool Effect* of agentic visual reasoning models (%).** The definition of all metrics can be found in Section 2.1. ↑ and ↓ indicate that higher and lower values are better, respectively. Δ_{Acc} = *Tool-Available Acc.* − *Tool-Free Acc.* and Δ_{TE} = *Tool-Gain* − *Tool-Harm*. Positive and non-positive deltas are shown in green and red, respectively. Detailed results are in Appendix E.2.

Dataset	Method	Accuracy			Mode Adaptiveness			Tool Effects			
		Tool-Available Acc. ↑	*Tool-Free Acc.* ↑	Δ_{Acc} ↑	MA_{tool} ↑	MA_{text} ↑	MA_{mean} ↑	*Tool-Gain* ↑	*Tool-Harm* ↓	Δ_{TE} ↑	*Text-Retain* ↑
Average	Thyme	47.63	46.89	+0.73	4.28	92.95	48.62	0.19	0.12	+0.06	81.43
	DeepEyesV2	45.71	44.89	+0.83	99.71	0.63	50.17	7.85	6.11	+1.74	–
	CodeV	47.24	46.92	+0.31	14.84	77.54	46.19	1.26	1.19	+0.07	71.39
	Metis	52.19	52.73	-0.54	32.61	72.71	52.66	1.14	1.10	+0.04	82.98
	Beacon (Ours)	53.53	51.57	**+1.96**	94.75	22.91	**58.83**	9.29	6.16	**+3.13**	**91.00**

Table 4: **Component-wise ablation study of Beacon.** "Overall Acc." indicates the average accuracy across all 13 benchmarks of Table 1 and 2. The other metrics are defined in Section 2.1 and are averaged over the five benchmarks and five runs in Table 3.

SFT	GRPO	NAAR	HCE	Overall Acc. ↑	*Tool-Available Acc.* ↑	*Tool-Free Acc.* ↑	MA_{mean} ↑	Δ_{TE} ↑
				52.91	–	–	–	–
✓				56.91	52.02	51.85	56.20	+1.61
✓	✓			57.10	52.25	51.40	56.30	+1.39
✓	✓	✓		57.75	53.25	**52.24**	**59.68**	+2.54
✓	✓		✓	57.62	52.94	51.82	58.36	+2.96
✓	✓	✓	✓	**58.98**	**53.53**	51.57	58.83	+3.14

Δ_{TE} of +3.14 percentage points. Meanwhile, its highest *Text-Retain* indicates that, when it chooses to answer without tools, it most consistently preserves correct tool-free answers across prompts.

Additional analysis. In Appendix E.1, we analyze *Tool Effect* on a fixed set of "text-easy" and "text-hard" samples instead of the model-dependent sets considered in Table 3. In Appendix E.4, we also analyze the evolution of the training dynamics of the RL stage.

4.4 Ablation Study

We conduct ablation studies to evaluate the contribution of each component of Beacon to the overall performance, *Mode Adaptiveness*, and *Tool Effect*. The results are presented in Table 4. We observe that our full method has the best overall performance, while using the necessity-aware adaptive reward alone achieves the best *Mode Adaptiveness*. The NAAR and the HCE module improve the overall *Mode Adaptiveness* and *Tool Effect* compared to the SFT+GRPO and SFT models, while directly conducting GRPO brings little improvement. The complete results of the ablation study are in Appendix E.3.

5 Conclusion

In this work, we revisit agentic visual reasoning from the perspective of *Mode Adaptiveness* and *Tool Effect*, to characterize whether a model can select the appropriate reasoning mode for each problem and whether tool use genuinely extends its capabilities beyond tool-free reasoning. Through a systematic evaluation of representative agentic visual reasoning models, we find that existing models often exhibit a strong preference for a single reasoning mode, while the gains brought by tools are largely offset by errors introduced through ineffective tool use. Motivated by these findings, we propose Beacon, an agentic visual reasoning model trained with a high-quality data synthesis pipeline and a tailored reinforcement learning framework. In particular, the *Necessity-Aware Adaptive Reward* encourages the model to avoid unnecessary tool calls without suppressing useful tool-assisted reasoning, while Hint-guided Capability Expansion recovers informative learning signals from difficult examples that conventional RL fails to solve. Extensive experiments across 13 diverse benchmarks demonstrate that Beacon achieves strong overall performance while substantially improving reasoning-mode adaptiveness and the relative gain of tool use.

AI USE STATEMENT

For the research project, we use Gemini 3.1 Pro to synthesize the SFT training data and provide hints in the HCE module for the RL training. For the paper writing, we leverage GPT 5.6 Sol to polish the language and fix typos. Generative AI tools were not used to make research decisions or to determine the conclusions reported in this paper. The authors are responsible for the design of the method and experiments, the verification and interpretation of the results, and the final content of the manuscript.

REFERENCES

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, et al. Qwen3-vl technical report. *arXiv preprint arXiv:2511.21631*, 2025a.

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025b.

Liang Chen, Weichu Xie, Yiyan Liang, Hongfeng He, Hans Zhao, Zhibo Yang, Zhiqi Huang, Haoning Wu, Haoyu Lu, Yiping Bao, et al. Babyvision: Visual reasoning beyond language. *arXiv preprint arXiv:2601.06521*, 2026.

Haodong Duan, Junming Yang, Yuxuan Qiao, Xinyu Fang, Lin Chen, Yuan Liu, Xiaoyi Dong, Yuhang Zang, Pan Zhang, Jiaqi Wang, et al. Vlmevalkit: An open-source toolkit for evaluating large multi-modality models. In *ACM MM*, 2024.

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive. In *European Conference on Computer Vision*, pp. 148–166. Springer, 2024.

Google DeepMind. Gemini 3 flash. `https://deepmind.google/models/model-cards/gemini-3-flash/`, 2025.

Google DeepMind. Gemini 3.1 Pro: Model Card. `https://deepmind.google/models/model-cards/gemini-3-1-pro/`, 2026.

Garvin 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*, 2026.

Chaoqun He, Renjie Luo, Yuzhuo Bai, Shengding Hu, Zhen Thai, Junhao Shen, Jinyi Hu, Xu Han, Yujie Huang, Yuxiang Zhang, et al. Olympiadbench: A challenging benchmark for promoting agi with olympiad-level bilingual multimodal scientific problems. In *ACL*, 2024.

Jack Hong, Chenxiao Zhao, ChengLIn Zhu, Weiheng Lu, Guohai Xu, and XingYu. Deepeyesv2: Toward agentic multimodal model. In *ICLR*, 2026.

Xinhai Hou, Shaoyuan Xu, Manan Biyani, Mayan Li, Jia Liu, Todd C Hollon, and Bryan Wang. Codev: Code with images for faithful visual reasoning via tool-aware policy optimization. In *CVPR*, 2026.

Xin Lai, Junyi Li, Wei Li, Tao Liu, Tianjian Li, and Hengshuang Zhao. Mini-o3: Scaling up reasoning patterns and interaction turns for visual search. *arXiv preprint arXiv:2509.07969*, 2025.

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. *arXiv preprint arXiv:2511.01833*, 2025.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. In *ICLR*, 2019.

Pan Lu, Ran Gong, Shibiao Jiang, Liang Qiu, Siyuan Huang, Xiaodan Liang, and Song-Chun Zhu. Inter-gps: Interpretable geometry problem solving with formal language and symbolic reasoning. In *ACL*, 2021.

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 *ICLR*, 2024.

Yan Ma, Weiyu Zhang, Tianle Li, Du Linge, Xuyang Shen, and Pengfei Liu. What does vision tool-use reinforcement learning really learn? disentangling tool-induced and intrinsic effects for crop-and-zoom. In *ICML*, 2026.

Ahmed Masry, Mohammed Saidul Islam, Mahir Ahmed, Aayush Bajaj, Firoz Kabir, Aaryaman Kartha, Md Tahmid Rahman Laskar, Mizanur Rahman, Shadikur Rahman, Mehrad Shahmohammadi, et al. Chartqapro: A more diverse and challenging benchmark for chart question answering. In *ACL Findings*, 2025.

Vaskar Nath, Elaine Lau, Anisha Gunjal, Manasi Sharma, Nikhil Baharte, and Sean Hendryx. Adaptive guidance accelerates reinforcement learning of reasoning models. *arXiv preprint arXiv:2506.13923*, 2025.

Ji Qi, Ming Ding, Weihan Wang, Yushi Bai, Qingsong Lv, Wenyi Hong, Bin Xu, Lei Hou, Juanzi Li, Yuxiao Dong, and Jie Tang. Cogcom: A visual language model with chain-of-manipulations reasoning. In *ICLR*, 2025.

Runqi Qiao, Qiuna Tan, Minghan Yang, Guanting Dong, Peiqing Yang, Shiqiang Lang, Enhui Wan, Xiaowan Wang, Yida Xu, Lan Yang, et al. V-thinker: Interactive thinking with images. *arXiv preprint arXiv:2511.04460*, 2025.

Gabriel Sarch, Linrong Cai, Qunzhong Wang, Haoyang Wu, Danqi Chen, and Zhuang Liu. Vero: An open rl recipe for general visual reasoning. *arXiv preprint arXiv:2604.04917*, 2026.

Gabriel Herbert Sarch, Snigdha Saha, Naitik Khandelwal, Ayush Jain, Michael J. Tarr, Aviral Kumar, and Katerina Fragkiadaki. Grounded reinforcement learning for visual reasoning. In *NeurIPS*, 2025.

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, YK Li, Yang Wu, et al. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *arXiv preprint arXiv:2402.03300*, 2024.

Guangming Sheng, Chi Zhang, Zilingfeng Ye, Xibin Wu, Wang Zhang, Ru Zhang, Yanghua Peng, Haibin Lin, and Chuan Wu. Hybridflow: A flexible and efficient rlhf framework. *arXiv preprint arXiv: 2409.19256*, 2024.

Yang Shi, Jiaheng Liu, Yushuo Guan, Zhenhua Wu, Yuanxing Zhang, Zihao Wang, Weihong Lin, Jingyun Hua, Zekun Wang, Xinlong Chen, et al. Mavors: Multi-granularity video representation for multimodal large language model. In *Proceedings of the 33rd ACM International Conference on Multimedia*, pp. 10994–11003, 2025.

Mohammad Shoeybi, Mostofa Patwary, Raul Puri, Patrick LeGresley, Jared Casper, and Bryan Catanzaro. Megatron-lm: Training multi-billion parameter language models using model parallelism. *arXiv preprint arXiv:1909.08053*, 2019.

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. *arXiv preprint arXiv:2512.17312*, 2025a.

Yueqi Song, Tianyue Ou, Yibo Kong, Zecheng Li, Graham Neubig, and Xiang Yue. Visualpuzzles: Decoupling multimodal reasoning evaluation from domain knowledge. *arXiv preprint arXiv:2504.10342*, 2025b.

Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel space reasoning via curiosity-driven reinforcement learning. In *NeurIPS*, 2025a.

Zhaochen Su, Linjie Li, Mingyang Song, Yunzhuo Hao, Zhengyuan Yang, Jun Zhang, Guanjie Chen, Jiawei Gu, Juntao Li, Xiaoye Qu, and Yu Cheng. Openthinkimg: Learning to think with images via visual tool reinforcement learning. *arXiv preprint arXiv:2505.08617*, 2025b.

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*, 2026.

Jingqi Tong, Jixin Tang, Hangcheng Li, Yurong Mou, Ming Zhang, Jun Zhao, Yanbo Wen, Fan Song, Jiahao Zhan, Yuyang Lu, et al. Game-rl: Synthesizing multimodal verifiable game data to boost vlms' general reasoning. *arXiv preprint arXiv:2505.13886*, 2025.

Shengbang Tong, Ellis Brown, Penghao Wu, Sanghyun Woo, Manoj Middepogu, Sai C Akula, Jihan Yang, Shusheng Yang, Adithya Iyer, Xichen Pan, et al. Cambrian-1: A fully open, vision-centric exploration of multimodal llms. *NeurIPS*, 2024.

Chaoyang Wang, Kaituo Feng, Dongyang Chen, Zhongyu Wang, Zhixun Li, Sicheng Gao, Meng Meng, Xu Zhou, Manyuan Zhang, Yuzhang Shang, and Xiangyu Yue. Adatooler-v: Adaptive tool-use for images and videos. *arXiv preprint arXiv:2512.16918*, 2026.

Fei Wang, Xingyu Fu, James Y Huang, Zekun Li, Qin Liu, Xiaogeng Liu, Mingyu Derek Ma, Nan Xu, Wenxuan Zhou, Kai Zhang, et al. Muirbench: A comprehensive benchmark for robust multi-image understanding. In *ICLR*, 2025a.

Ke Wang, Junting Pan, Weikang Shi, Zimu Lu, Houxing Ren, Aojun Zhou, Mingjie Zhan, and Hongsheng Li. Measuring multimodal mathematical reasoning with math-vision dataset. *NeurIPS*, 2024.

Qixun Wang, Yang Shi, Yifei Wang, Yuanxing Zhang, Pengfei Wan, Kun Gai, Xianghua Ying, and Yisen Wang. Monet: Reasoning in latent visual space beyond images and language. *arXiv preprint arXiv:2511.21395*, 2025b.

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 *AAAI*, 2025c.

Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. In *CVPR*, pp. 13084–13094, 2024.

xAI. RealWorldQA. `https://huggingface.co/datasets/xai-org/RealworldQA`, 2024. A benchmark for evaluating real-world spatial understanding.

Shilin Yan, Jintao Tong, Hongwei Xue, Xiaojun Tang, Yangyang Wang, Kunyu Shi, Guannan Zhang, Ruixuan Li, and Yixiong Zou. Act wisely: Cultivating meta-cognitive tool use in agentic multimodal models. *arXiv preprint arXiv:2604.08545*, 2026.

Wenjie Yang, Siqi Zhu, and Zengfeng Huang. Position: Your VLM may not be thinking with interleaved images. In *ICML Position Paper Track*, 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 *CVPR*, 2024.

Yi-Fan Zhang, Xingyu Lu, Shukang Yin, Chaoyou Fu, Wei Chen, Xiao Hu, Bin Wen, Kaiyu Jiang, Changyi Liu, Tianke Zhang, Haonan Fan, Kaibing Chen, Jiankang Chen, Haojie Ding, Kaiyu Tang, Zhang Zhang, Liang Wang, Fan Yang, Tingting Gao, and Guorui Zhou. Thyme: Think beyond images. In *ICLR*, 2026.

Yusen Zhang, Wenliang Zheng, Aashrith Madasu, Peng Shi, Ryo Kamoi, Hao Zhou, Zhuoyang Zou, Shu Zhao, Sarkar Snigdha Sarathi Das, Vipul Gupta, et al. Hrscene: How far are vlms from effective high-resolution image understanding? In *ICCV*, 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.

Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. Swift:a scalable lightweight infrastructure for fine-tuning. *arXiv preprint arXiv:2408.05517*, 2024.

Jiyang Zheng, Jialiang Shen, Yu Yao, Min Wang, Yang Yang, Dadong Wang, and Tongliang Liu. Chain-of-focus prompting: Leveraging sequential visual cues to prompt large autoregressive vision models. In *ICLR*, 2025.

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 CONTENTS

A Related Work **15**

B Experimental Details **16**

B.1 Details of the Metric Analysis Experiments 16

B.2 Details of the Evaluation Experiments 16

C Training Data Details **16**

C.1 Overview 16

C.2 Data Sources 17

C.3 SFT Data Synthesis 19

D Training Details **23**

D.1 Details of Cold-start SFT 23

D.2 Details of RL 23

D.3 System Prompts for SFT, RL, and Evaluation 24

D.4 Details for Hint Generation 24

E Additional Experimental Results **25**

E.1 Measuring the Tool Effects on Fixed "Text-easy" and "Text-hard" Sets 25

E.2 Detailed Results of the Adaptiveness and Tool Effects 26

E.3 Detailed Results of Ablation Study 26

E.4 Training Dynamics of the RL Stage 27

F Discussion **28**

F.1 Limitations 28

F.2 Failed Attempts 29

G Case Study **29**

G.1 Reasoning Trajectory Examples 29

G.2 Hint Examples 29

A Related Work

Agentic Visual Reasoning. Recent MLLMs (Bai et al., 2025a; Shi et al., 2025; Wang et al., 2025b) have moved beyond static visual inputs by interleaving textual reasoning with intermediate visual interactions. Existing approaches enable models to revisit image regions, crop or edit images, generate auxiliary visual representations, or execute code for visual manipulation and numerical computation (Qi et al., 2025; Su et al., 2025a; Zheng et al., 2025; Zhu et al., 2026; Sarch et al., 2025; Su et al., 2025b; Zhang et al., 2026; Hong et al., 2026; Qiao et al., 2025; Song et al., 2025a).

These studies have demonstrated the potential of external tools for fine-grained perception and complex multimodal reasoning. However, most evaluations primarily report aggregate task accuracy, providing limited evidence about whether the model can appropriately choose between direct reasoning and tool-augmented reasoning, and whether tool use genuinely extends the model's capabilities beyond tool-free reasoning.

Adaptive and Efficient Tool Use. Several recent studies have recognized that indiscriminate tool invocation can introduce unnecessary computation and propose various RL strategies to encourage more adaptive tool use (Song et al., 2025a; Wang et al., 2026; Yan et al., 2026).

While these works provide pioneering insights, their adaptive goals may still suffer from several limitations. CodeDance (Song et al., 2025a) uses group-level accuracy to determine the optimization direction of its adaptive reward: high or low group accuracy indicates that tools should be used less or more frequently, respectively. However, this design does not condition the reward on whether a particular trajectory has actually invoked a tool. As a result, it may incorrectly penalize the tool-use behavior of a successful tool-assisted trajectory simply because it belongs to a high-accuracy group. AdaTooler-V (Wang et al., 2026) instead labels training samples as requiring or not requiring tools according to the tool-induced performance gain of a teacher model, Qwen2.5-VL-72B (Bai et al., 2025b). Such teacher-derived supervision may suffer from distribution mismatch between the policy and the teacher, while also being constrained by the teacher's own tool-use capability. Metis (Yan et al., 2026) adopts a simpler strategy that generally encourages fewer tool calls, but this objective lacks task-dependent adaptiveness and may suppress tool use even when it is necessary for solving difficult problems.

Our adaptive reward design (Section 3.4.1) addresses these limitations by following two principles: (1) mode-conditioned labeling, which labels each problem as requiring or not requiring tools based on both tool-free and tool-assisted performance, and (2) online labeling, which labels each problem based on the current policy's performance rather than a fixed teacher model, mitigating off-policy issues.

Understanding the Actual Benefit of Visual Tools. A growing line of diagnostic work questions whether the gains of agentic visual reasoning genuinely arise from intermediate visual tools. Yang et al. (2026) find that interleaved visual results contribute marginally over tool-free reasoning and that improvements mainly originate from SFT. Ma et al. (2026) show that tool-use RL mainly reduces errors introduced by tool use, while making limited progress in correcting problems that remain unsolved without tools. Similarly, Guo et al. (2026) find that tool-induced performance gains and harms both contribute only marginally to the overall performance.

Despite these important observations, existing analyses do not provide a unified characterization of the reasoning-mode adaptiveness and the effect of tool execution. Moreover, they typically study tool-free solvability through a binary outcome from a single inference run, which is intrinsically unstable and could be easily perturbed by stochasticity, while obscuring how autonomous invocation behavior varies across different levels of tool-free solvability. Lastly, to the best of our knowledge, no prior work has proposed an explicit solution to improve the genuine benefit of tool use.

In contrast, our analysis framework (Section 2.1) considers both adaptiveness and the effect of tool use, while providing a more robust evaluation with multiple inference runs. We also propose a novel RL framework to improve both aspects (Section 3.4).

B Experimental Details

B.1 Details of the Metric Analysis Experiments

Prompt Box 1: Prompt to enforce tool-free answers

```
You are a helpful assistant.

Solve the following problem step by step by reasoning directly from the
provided image and question.

You must not use any external tools, code execution, Python, or search. Do not
output <code> blocks or any tool-call format.

Output format must be:
<think>...</think>
<answer>...</answer>
```

B.2 Details of the Evaluation Experiments

We build our evaluation framework on the open-source VLMEvalKit toolkit (Duan et al., 2024). For each model, we implement the tool-use inference pipeline following its official implementation, including the prompts, tool-call format, and execution procedure. Each model may perform at most 20 rounds of tool calls. We set the decoding temperature to 0.5 for all models.

For answer evaluation, we first apply rule-based matching between the model's final prediction and the ground-truth answer. If the rule-based matching fails, we provide both the prediction and the ground-truth answer to Gemini 3.1 Pro (Google DeepMind, 2026) or Gemini 3 Flash (Google DeepMind, 2025) for semantic answer judgment.

C Training Data Details

C.1 Overview

In this section, we will introduce the construction pipeline of our training data.

1. Source data selection. We select source datasets according to three principles: **1) Diversity:** the data should span a broad range of tasks and domains to promote generalization; **2) Quality:** the ground-truth annotations should be reliable and minimally noisy; and **3) Difficulty:** the tasks should be sufficiently challenging to benefit from tool use and expand the model's capabilities. Following these principles, we collect data from 12 datasets, covering real-world perception, chart understanding, OCR, STEM, spatial reasoning, and real-world agentic reasoning. Most of these datasets contain human-annotated labels. To prevent evaluation contamination, we remove all samples that overlap with the test sets used in our experiments.

2. SFT data synthesis. We construct the SFT data in three stages. (1) We sample five responses from the base model, Qwen3-VL-8B-Instruct, and retain examples that are answered correctly at most twice. (2) For these hard examples, we use Gemini 3.1 Pro (Google DeepMind, 2026) to generate code-assisted reasoning trajectories containing Python code and the corresponding execution outputs, retaining only trajectories that yield correct answers. (3) Because the generated trajectories may still contain low-quality reasoning or tool-use steps, we introduce an additional refinement procedure using Gemini 3.1 Pro.

3. RL data construction. We evaluate the SFT model on the same data pool and retain the examples that are answered correctly no more than three times out of five attempts. These hard examples constitute the RL training set, on which the model is further optimized using reward signals that account for both answer correctness and the effectiveness of tool use.

Table 5 summarizes the statistics of the training data.

Table 5: **Training data statistics.** We report both the total number of raw samples and the number of samples retained after data filtering. The final retained SFT data contains 30K additional trajectories of problems that are correctly answered by the base model Qwen3-VL-8B-Instruct.

Data Type	Original	Filtered	Final retained
SFT Data	212,353	15,705	45,705
RL Data	45,886	15,709	15,709



Figure 5: **Training data sources of Beacon.**

C.2 Data Sources

Beacon leverages a diverse collection of publicly available multimodal data sources spanning mathematical reasoning, chart understanding, embodied interaction, spatial perception, game reasoning, and general visual reasoning.

Specifically, the training data are collected from Geometry3K (Lu et al., 2021), OlympiadBench (He et al., 2024), AgentVista (Su et al., 2026), MuirBench (Wang et al., 2025a), HRScene (Zhang et al., 2025), CV-Bench (Tong et al., 2024), MMMU (Yue et al., 2024), and Vero (Sarch et al., 2026). These data sources cover a broad range of task formats and visual reasoning requirements, enabling Beacon to expose models to heterogeneous scenarios with different levels of tool dependency.

Mathematical and geometry-oriented sources, including Geometry3K (Lu et al., 2021) and OlympiadBench (He et al., 2024), provide training samples that require visual mathematical reasoning, geometric diagram understanding, symbolic manipulation, and multi-step scientific problem solving. HRScene (Zhang et al., 2025) and CV-Bench (Tong et al., 2024) enrich the training corpus with high-resolution image understanding, region-sensitive perception, object counting, spatial relation reasoning, and 2D/3D visual understanding. AgentVista (Su et al., 2026) provides tool-oriented and agentic visual reasoning data, where models are encouraged to perform long-horizon reasoning with intermediate visual operations. Finally, MMMU (Yue et al., 2024), MuirBench (Wang et al., 2025a), and Vero (Sarch et al., 2026) introduce broader multimodal reasoning scenarios across academic, multi-image, and general visual reasoning tasks.

As shown in Figure 5, the resulting corpus spans a broad spectrum of task types, reasoning complexity, and tool requirements. While some samples can be solved through direct multimodal reasoning, others inherently benefit from external tools for operations such as visual manipulation, geometric

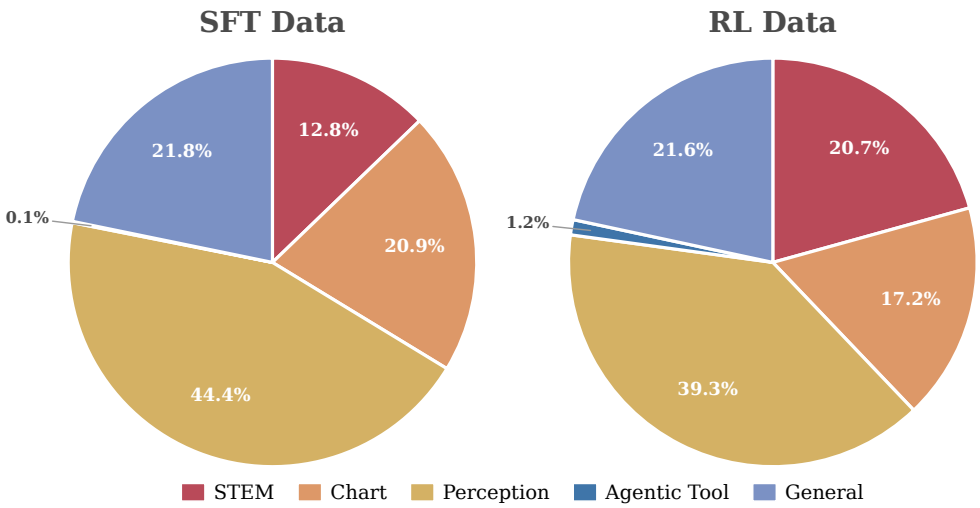


Figure 6: **Task category distribution of the SFT and RL data.**

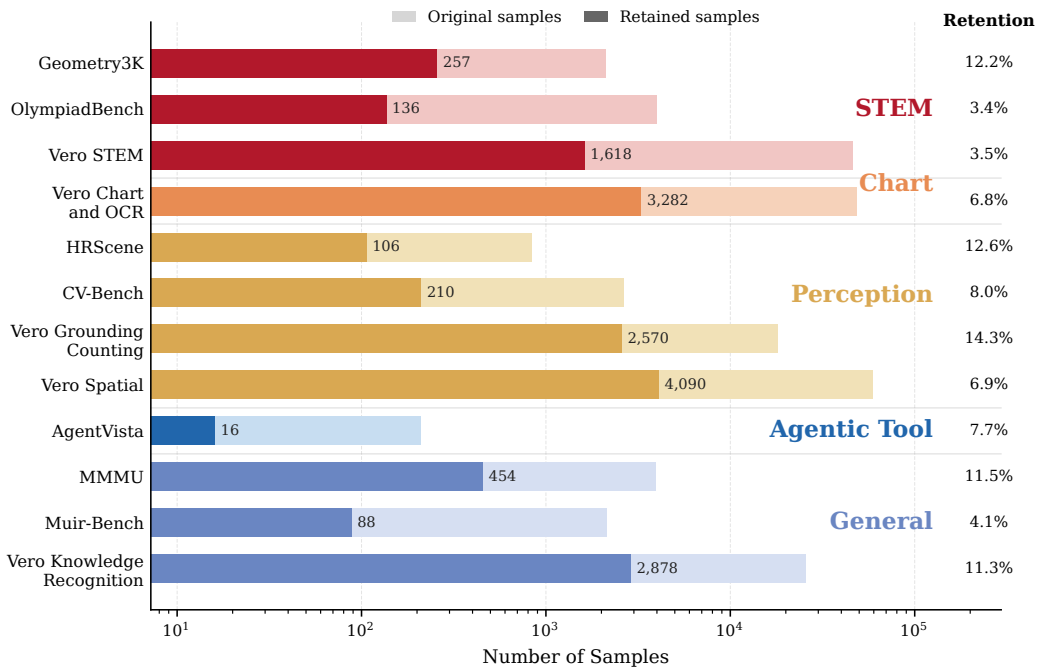


Figure 7: **Detailed statistics of the SFT data.** The numbers in the bars indicate the number of samples retained after rejection sampling, which removes easy samples, and refinement, which removes code trajectories where code does not actually help to derive the answer. "Retention" denotes the percentage of retained samples relative to the original samples.

analysis, or structured information extraction. This diversity exposes models to both tool-necessary and tool-optional scenarios, providing a foundation for learning not only how to use tools effectively, but also when tool invocation is genuinely beneficial. We present the task distribution of the training data in Figure 6. Figures 7 and 8 further show the number of samples from each dataset in the SFT and RL data before and after filtering, respectively.

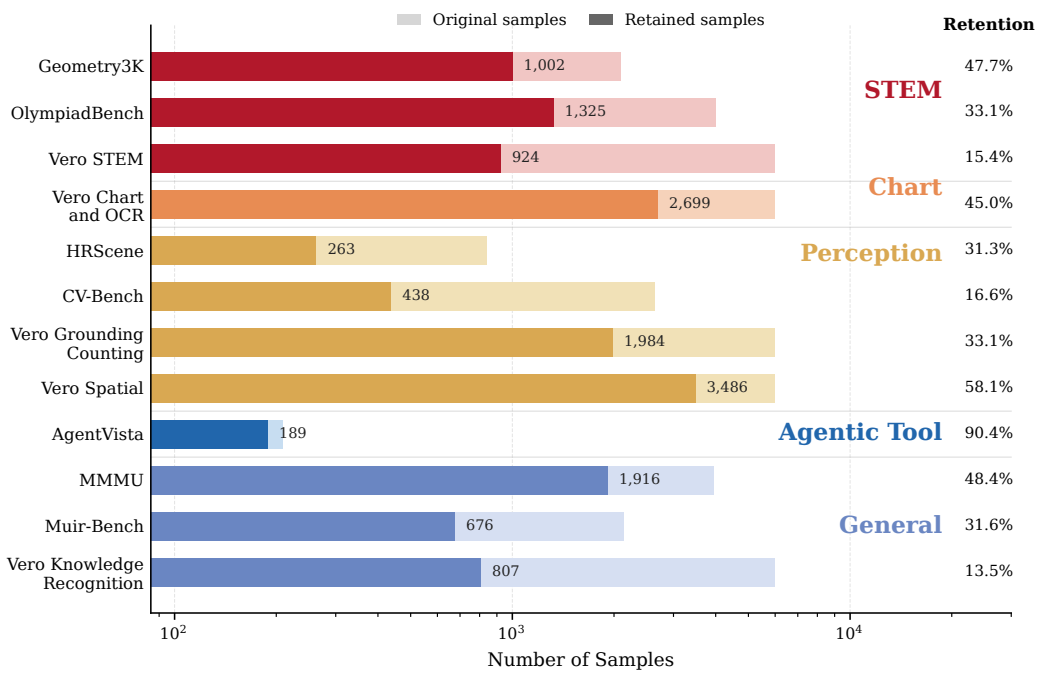


Figure 8: **Detailed statistics of the RL data.** The numbers in the bars indicate the number of samples retained after rejection sampling, which removes easy samples. "Retention" denotes the percentage of retained samples relative to the original samples.

C.3 SFT Data Synthesis

The synthesis of the SFT data consists of two steps: 1) generating code-based reasoning trajectories for the source problems with Gemini 3.1 Pro (Google DeepMind, 2026) and retaining only those whose final answers match the ground-truth labels; and 2) refining the retained trajectories using the same model.

1. Prompt for SFT code-reasoning trajectory synthesis. The prompt is provided in Box 2. Its key design choices are summarized below:

- **Representative code actions.** We provide five representative code actions, including `crop`, `draw_line`, `draw_box`, `numeric_calculation`, and `rotation`, covering common visual reasoning operations. For each action, we specify its purpose, recommended library, required imports, input convention, and a concrete example. These examples serve as guidance rather than constraints; in practice, Gemini 3.1 Pro (Google DeepMind, 2026) generates a much wider variety of code patterns beyond the provided examples.
- **Mandatory code use.** We explicitly instruct Gemini 3.1 Pro (Google DeepMind, 2026) to invoke code for obtaining visual evidence or other useful intermediate information. Since the model may still directly answer some questions without using code, we discard trajectories that do not contain valid tool execution.
- **Structured trajectory format.** We require the model to generate trajectories in a structured format, using `<tool_call>...</tool_call>` for code execution, `<tool_response>...</tool_response>` for execution results, `<observation>...</observation>` for interpreting the returned evidence, and `<answer>...</answer>` for the final prediction. This structured format improves interpretability, enables reliable automatic parsing, and encourages explicit reasoning over tool outputs before producing the final answer.
- **Sandbox protocols.** We define several execution protocols to reduce ambiguity and simplify parsing. Question images and code-generated images are assigned unique identifiers

(e.g., q0, q1, g0, and g1), which can be accessed through get_image(ref). The final line within each <tool_call>...</tool_call> block must be the executable expression, returning either a PIL image object or a numeric value. In addition, the model is instructed to obtain image dimensions via img.size, ensuring that coordinate-based operations are grounded in the actual image resolution.

Prompt Box 2: Prompt for SFT Data Synthesis

```
You are an image reasoning assistant. You may write Python code to obtain
visual evidence or any helpful information before answering. This includes but
is not limited to zooming in, drawing auxiliary lines, rotating, adjusting
contrast, computing statistics, etc. Put the executable Python code inside <
tool_call></tool_call> tags. When you give the final answer, you must wrap it
inside <answer></answer> tags.

# Code Actions

Here are representative code patterns for acting on images and numbers. They
are recommended starting points, not an exhaustive list. You may use other
valid Python operations, library calls, or parameter choices as long as they
fit the runtime constraints.

## crop
Purpose: Crop a rectangular region from an image using Pillow.

Recommended library: Pillow

Required imports:
from PIL import Image

Input convention:
- Read the source image from get_image("q0") / get_image("g0") or from images[
index].
- Use a 4-tuple (left, top, right, bottom) as the crop box.
- The last line should be the cropped PIL image object.
Example:
<tool_call>
img = get_image("q0")
cropped = img.crop((120, 180, 420, 620))
cropped
</tool_call>

## draw_line
Purpose: Draw one or more auxiliary line segments on an image using Pillow.

Recommended library: Pillow

Required imports:
from PIL import ImageDraw

Input convention:
- Read the source image from get_image("q0") / get_image("g0") or from images[
index].
- Always call copy() before drawing.
- Use ImageDraw.Draw(annotated).line((x1, y1, x2, y2), fill="red", width=4).
- The last line should be the annotated PIL image object.
Example:
<tool_call>
from PIL import ImageDraw
annotated = get_image("q0").copy()
draw = ImageDraw.Draw(annotated)
draw.line((118, 214, 612, 248), fill="red", width=4)
draw.line((132, 486, 628, 520), fill="red", width=4)
annotated
</tool_call>

## draw_box
Purpose: Draw one or more bounding boxes on an image using Pillow.

Recommended library: Pillow

Required imports:
from PIL import ImageDraw

Input convention:
- Read the source image from get_image("q0") / get_image("g0") or from images[
index].
```

```
- Always call copy() before drawing.
- Use ImageDraw.Draw(annotated).rectangle((x1, y1, x2, y2), outline="red",
width=3).
- The last line should be the annotated PIL image object.
Example:
<tool_call>
from PIL import ImageDraw
annotated = get_image("q0").copy()
draw = ImageDraw.Draw(annotated)
for box in [(42, 38, 128, 124), (150, 38, 236, 124)]:
    draw.rectangle(box, outline="red", width=3)
annotated
</tool_call>

## numeric_calculation
Purpose: Evaluate a mathematical expression using Python and math, instead of
mental arithmetic.

Recommended library: math

Required imports:
import math

Input convention:
- Write the calculation directly in Python, such as result = 5 / 8 * 100 or
result = math.sqrt(2) * 10.
- Import math only when you actually need math functions or constants.
- The last line should be the numeric result itself.
Example:
<tool_call>
import math
result = math.sqrt(2) * 10
result
</tool_call>

## rotation
Purpose: Rotate an image using Pillow to correct orientation or inspect a
different view.

Recommended library: Pillow

Required imports:
from PIL import Image

Input convention:
- Read the source image from get_image("q0") / get_image("g0") or from images[
index].
- Use img.rotate(angle, expand=True).
- Positive angles rotate counterclockwise.
- The last line should be the rotated PIL image object.
Example:
<tool_call>
img = get_image("q0")
rotated = img.rotate(90, expand=True)
rotated
</tool_call>

Additional reminders:

1. Even if you can already guess or know the answer from the image, you should
still use sandboxed Python snippets to mark, isolate, or verify the relevant
visual evidence before answering.
2. Use get_image(ref) to access runtime images. Original question images use
stable ids like q0, q1, and images returned by earlier code calls use stable
ids like g0, g1.
3. If you need to draw on an image, call copy() first so you do not modify the
source image directly. For example: annotated = get_image("q0").copy().
4. If the current call should return an image result, the last line inside <
tool_call> should be the PIL image object itself, such as cropped, rotated, or
annotated. If the current call should return a numeric result, the last line
should be the numeric object itself, such as result.
5. Do not do arithmetic mentally. Use Python code. You may import math when
needed.
6. <tool_response>...</tool_response> is inserted by the system after the code
finishes running. It is not something you should generate yourself. Never
output <tool_response> tags on your own.
7. After each code execution returns a result, you must immediately output an <
observation>...</observation> tag that explicitly analyzes the returned result
before deciding whether to call code again.
```

```
8. The order must be strictly <tool_call>...</tool_call> -> <tool_response
>...</tool_response> -> <observation>...</observation>. After that, you may
continue with additional <tool_call> -> <tool_response> -> <observation> cycles
.
9. Your final answer must be grounded in the tool results and observations you
obtained. If a tool result is insufficient, unclear, or unhelpful, explicitly
say so in <observation> and call code again to get better evidence instead of
relying on your own unaided visual guess.

Full example:
Question: Is the man phone both blue and closed?
Assistant:
<tool_call>
img = get_image("q0")
img.size
</tool_call>
[The system then inserts a tool result wrapped in <tool_response>...</
tool_response>]
<observation>I first checked the image size so that later coordinates are
grounded in the actual canvas. I still need localized visual evidence for the
phone itself.</observation>
<tool_call>
from PIL import ImageDraw
annotated = get_image("q0").copy()
draw = ImageDraw.Draw(annotated)
draw.rectangle((600, 290, 660, 380), outline="red", width=3)
annotated
</tool_call>
[The system then inserts a tool result wrapped in <tool_response>...</
tool_response>]
<observation>The box covers the phone in the man's hand. The phone is visibly
blue, and because he is actively holding it to his ear in use, it is not closed
.</observation>
<answer>No</answer>
```

2. Prompt for refining synthesized code-reasoning trajectories. Although Gemini 3.1 Pro (Google DeepMind, 2026) can synthesize high-quality code-reasoning trajectories using the prompt in Box 2, the generated trajectories still exhibit several common issues:

- ✗ **Redundant or repetitive tool calls.** The model may repeatedly invoke similar code operations or generate unnecessary intermediate results, resulting in unnecessarily long and inefficient trajectories.
- ✗ **Insufficient reliance on tool outputs.** Although the model executes code to obtain visual evidence, the returned results are sometimes only weakly utilized. Consequently, the final answer is still primarily based on the model's direct visual reasoning rather than evidence obtained through tool execution.
- ✗ **Uninformative reasoning.** Some trajectories contain repetitive or low-quality reasoning loops, where the model repeatedly produces similar thoughts or observations without making meaningful progress toward the final answer.
- ✗ **Missing observations of tool responses.** The model may omit explicit observations after tool execution, making it unclear how the returned evidence is interpreted or how it supports the final prediction.

To address these issues, we further refine the synthesized trajectories with Gemini 3.1 Pro (Google DeepMind, 2026) using the prompt in Box 3. The refinement aims to remove trajectories whose final answers bypass tool outputs, eliminate redundant or uninformative code calls and reasoning, and enforce a cleaner trajectory structure. Manual inspection shows that the refined trajectories are substantially higher in quality, featuring more concise code execution, more informative reasoning, and stricter adherence to the prescribed format. These observations highlight the importance of trajectory refinement for high-quality automatic data synthesis.

Prompt Box 3: Prompt for SFT Data Refinement

```
You are refining a visual reasoning trajectory with tool calls.
Do not call tools again. Do not invent tool outputs. Only refine the trajectory
 text while keeping the final answer unchanged.
```

```
You will see the original question first, then the trajectory messages after
the question.
The trajectory uses the following conventions:
1. Tool returns are wrapped in <tool_response>...</tool_response> and images
are inserted in place.
2. <tool_call>...</tool_call> is executable code.
3. <observation>...</observation> is the observation after a tool returns.
4. <answer>...</answer> is the final answer.
5. Text thoughts may be interleaved anywhere in the trajectory.
Your task:
1. Check whether the trajectory actually relied on useful tool information to
answer the question. If the tools did not provide helpful information and the
answer was effectively derived from tool-free reasoning, especially in the last
 step, discard the trajectory.
2. Check whether the same tool was called repeatedly with unhelpful retries.
Remove the unhelpful repeated tool-call steps and keep only the step that
produced the key useful information.
3. For each tool call, if the tool result is followed by no meaningful <
observation>, add one.
4. Remove clearly meaningless tool calls or meaningless code, such as trivial
assignments, like "<tool_call>result = 1result</tool_call>".
5. Remove meaningless thought fragments.
6. Note that `img.size` is necessary for the model to know the image size, so
keep the `img.size` tool call that corresponds to it even if it looks trivial.
7. Check whether each <observation> block precisely describes the preceding
tool call result. If not, try to refine the <observation> to make it more
accurate and specific to the actual tool response.
Output requirements:
1. Output JSON only.
2. If no refinement is needed, output {"decision":"keep"}.
3. If the trajectory should be discarded, output {"decision":"drop","reason
":"..."}.
4. If refinement is needed, output JSON in this schema:
{
  "decision": "refine",
  "messages": [
    {"kind": "assistant", "text": "... assistant message text ..."},
    {"kind": "original_message", "source_message_id": "M3"}
  ]
}
5. In "messages", do not repeat the question or system prompt. Only output the
messages after the original question.
6. Use {"kind":"original_message","source_message_id":"Mx"} only for original
non-assistant messages that should be kept exactly, especially tool-response
user messages.
7. For assistant messages, put the full assistant text in "text". If a tool-
response step is kept, keep its <tool_response> in the corresponding
original_message instead of rewriting it.
8. Keep the final answer unchanged.
9. If a <tool_call> step is removed, also remove the corresponding <
tool_response> and <observation>.
```

D Training Details

D.1 Details of Cold-start SFT

For SFT, we train Qwen3-VL-8B-Instruct for 4 epochs using Megatron (Shoeybi et al., 2019) on the ms-swift framework (Zhao et al., 2024). We optimize the model with AdamW (Loshchilov & Hutter, 2019) using a peak learning rate of 1×10^{-5}, a weight decay of 0.1, and a cosine learning-rate schedule with a 5% warm-up ratio. Gradients are accumulated over 128 micro-batches before each parameter update.

D.2 Details of RL

We use VeRL (Sheng et al., 2024) as the RL training framework. RL training is conducted for 1 epoch on 64 NVIDIA H200 GPUs. For each prompt, the policy generates 8 rollouts, with a batch size of 128 prompt groups and a PPO mini-batch size of 128. We use an actor learning rate of 1×10^{-6} and the vanilla token-mean PPO objective with symmetric clipping bounds of 0.2, while disabling both KL and entropy regularization. The maximum sequence length is 30,720 tokens, including up to 20,480 prompt tokens and 10,240 response tokens, with at most 12 tool calls per rollout. For all-

wrong prompt groups, structured hints are generated using up to three expert attempts and 15 expert tool calls with 32 parallel workers. If the expert fails to generate a correct solution, we simply discard this sample. Training is performed in `bfloat16` precision with gradient checkpointing and a frozen vision encoder.

D.3 System Prompts for SFT, RL, and Evaluation

To avoid distribution mismatch, we use the same system prompt for SFT, RL training, and evaluation. The prompt is shown in Box 4. It is identical to the system prompt used for SFT data synthesis (Box 2), except that the mandatory code-use instruction is replaced with an optional one (`"You should decide whether to use code..."`).

Prompt Box 4: The system prompt for SFT, RL, and evaluation

```
...the same as the system prompt used for SFT data synthesis...

Additional reminders:

1. You should decide whether to use code to solve the problem. Only write codes
 when necessary. If you can directly solve the problem, don't use code.
2. ...

...the same as the system prompt used for SFT data synthesis...
```

D.4 Details for Hint Generation

The system prompt for hint generation is shown in Box 5. The generated hint consists of a sequence of reasoning steps, each containing an instruction and an expected subgoal. The instruction specifies the reasoning, observation, or tool action to perform, while the subgoal describes the intermediate evidence or outcome to be obtained without revealing the final answer. In our current implementation, we retain only the `"hint"` field as the training hint. The remaining fields, `"tool_steps"` and `"subgoals"`, are preserved for analysis and may serve as fine-grained supervision or reward signals in future work.

After obtaining the hinted group, we drop the original group with all-wrong responses.

Prompt Box 5: The system prompt for hint generation

```
You will convert a correct multimodal reasoning trajectory into a hint for a
weaker policy model.

The hint must NOT include the final answer, the reference answer, or an
explicit restatement of either answer. It should describe the useful method,
evidence to seek, and tool-use plan only.

First identify which steps in the Gemini trajectory are truly necessary for
reaching the answer, then condense only those key steps into a short plan. Do
NOT copy the full trajectory, every tool call, or incidental observations. Keep
 only the reasoning, visual observation, or tool steps that are most useful for
 solving the task.

The hint field itself must be a numbered list. Each item must use exactly this
pattern: "1. [Instruction] xxx; [Expected Subgoal] xxx;". The Instruction
should state the reasoning, observation, or tool action to perform. The
Expected Subgoal should state what intermediate evidence or result that step
should obtain, without revealing the final answer.

Prefer actionable tool steps over vague advice. Describe code actions at a high
 level rather than copying large code blocks from the trajectory. The tool
field should be a concise semantic operation label derived from the Python code
 content, not merely the word python. Use exact canonical labels from the code-
action prompt when applicable: crop, draw_line, draw_box, numeric_calculation,
rotation.

For each tool step, subgoal_result must summarize the actual evidence or result
 obtained after the code ran. Do not write imperative sentences such as 'crop
the image' or 'check the labels' in subgoal_result; write factual statements
```

```
such as 'the crop shows ...' or 'the calculation returns ...'. If a tool result
 contains exact numeric values, coordinates, counts, OCR strings, table cells,
or other precise evidence, include them explicitly in subgoal_result. Make
subgoal_result as informative as needed to show what was learned and why it
matters for that step's subgoal, while keeping it answer-free.

Extraction mode:
Extract the critical tool-use sequence and, for each step, state what
information that tool step was meant to obtain. Omit tool calls or observations
 that did not help. If the trajectory did not use tools, extract the key
reasoning idea instead.

Subgoal instruction:
Return subgoals as exactly the ordered list of subgoal_result strings from
tool_steps. This keeps the training-facing subgoals grounded in actual tool
results rather than intended actions.

Return exactly one JSON object with this schema:
{
  "hint": "1. [Instruction] key reasoning, observation, or tool step; [Expected
 Subgoal] answer-free intermediate evidence or result;\n2. [Instruction] next
key step; [Expected Subgoal] next answer-free intermediate evidence or result
;",
  "tool_steps": [
    {
      "step": 1,
      "tool": "semantic operation name such as crop, draw_line, draw_box,
numeric_calculation, rotation, zoom_in, or read_text",
      "action": "high-level tool action, no large code block",
      "expected_subgoal": "one concise paragraph describing the intermediate
evidence this step is meant to obtain",
      "subgoal_result": "detailed factual answer-free result actually obtained,
 including exact numeric/textual evidence when available and why it matters for
 this subgoal"
    }
  ],
  "subgoals": ["same ordered strings as tool_steps[*].subgoal_result"]
}

Question:
{question}

Reference final answer, for filtering only. Do not reveal it in the hint:
{ground_truth}

Correct trajectory:
{trajectory}
```

E Additional Experimental Results

E.1 Measuring the Tool Effects on Fixed "Text-easy" and "Text-hard" Sets

In Section 2.1, we first split the "text-easy" and "text-hard" sets based on the behavior of the post-trained agentic visual reasoning model. Therefore, different models may have different splits of the two sets. Although such a design makes the analysis of the adaptiveness achievable, it may make the comparison of tool effects across different models less fair. In this section, we present the tool effects on fixed "text-easy" and "text-hard" sets, which are split based on the base model (Qwen3-VL-8B-Instruct). Given a dataset, we use the base model to generate five responses for each sample and retain the samples that are answered correctly at least four times as the "text-easy" set, and the samples that are answered correctly at most once as the "text-hard" set. We analyze models that are post-trained on Qwen3-VL-8B-Instruct: the SFT model trained with Beacon's SFT data, Metis, and Beacon. The results are shown in Table 6. We report only the tool effects rather than mode adaptiveness, since the difficulty of solving a sample without tools is determined by the base model and may not reflect the capabilities of each post-trained model. Nevertheless, measuring tool gain and harm on these fixed sets remains meaningful, as it reveals how tool-use post-training extends and preserves the capabilities of the base model.

From Table 6, we obtain similar observations to those in Section 4.3, i.e., Beacon exhibits the largest gap between the tool-induced performance gain and degradation.

Table 6: **Detailed results for *Tool Effect* (%) on the fixed "text-easy" and "text-hard" sets.** The definition of all metrics can be found in Section 2.1. Average denotes the unweighted mean over the four datasets shown.

Dataset	Method	Accuracy			Tool Effects			
		Tool-Available Acc. ↑	*Tool-Free Acc.* ↑	Δ_{Acc} ↑	*Tool-Gain* ↑	*Tool-Harm* ↓	Δ_{TE} ↑	*Text-Retain* ↑
HRBench4K	SFT	81.35 (71.25)	77.68 (0.00)	+3.68	6.65	4.68	+1.98	98.38
	Metis	84.13 (46.88)	83.80 (0.00)	+0.33	3.25	0.08	+3.18	95.70
	Beacon	83.38 (70.65)	80.53 (0.00)	+2.85	7.05	4.48	+2.58	98.83
BLINK	SFT	64.17 (72.49)	64.80 (0.00)	-0.63	7.80	10.46	-2.66	93.09
	Metis	62.09 (24.51)	64.67 (0.00)	-2.58	2.70	2.97	-0.26	83.24
	Beacon	65.49 (79.79)	64.22 (0.00)	+1.27	12.98	10.63	+2.36	98.72
BabyVision	SFT	14.95 (97.32)	17.73 (0.00)	-2.78	7.58	3.56	+4.02	100.00
	Metis	15.52 (15.72)	15.93 (0.00)	-0.41	0.36	0.21	+0.15	68.32
	Beacon	16.70 (95.36)	15.36 (0.00)	+1.34	6.39	2.01	+4.38	85.71
MathVista	SFT	75.76 (76.86)	78.82 (0.00)	-3.06	4.32	6.86	-2.54	98.23
	Metis	77.36 (17.34)	78.02 (0.00)	-0.66	0.92	0.80	+0.12	94.68
	Beacon	77.80 (78.52)	77.18 (0.00)	+0.62	5.00	5.72	-0.72	97.68
Average	SFT	59.06	59.76	-0.70	6.59	6.39	+0.20	97.43
	Metis	59.77	60.60	-0.83	1.81	1.01	+0.80	85.49
	Beacon	60.84	59.32	**+1.52**	7.86	5.71	**+2.15**	95.24

E.2 Detailed Results of the Adaptiveness and Tool Effects

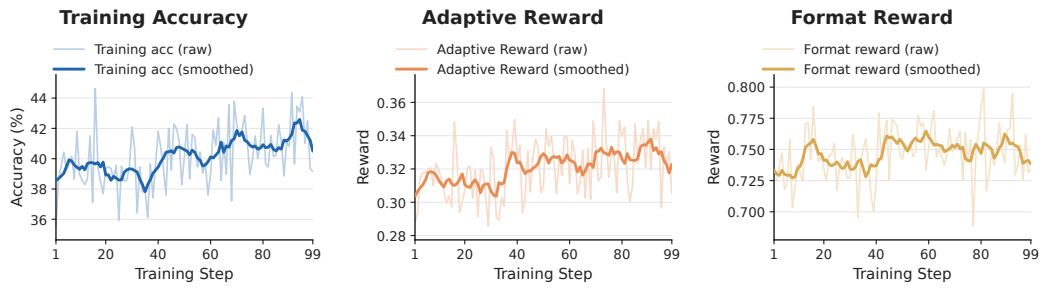
In Table 7, we present the complete results of the *Mode Adaptiveness* and *Tool Effect* metrics of Beacon and several agentic visual reasoning baselines.

Table 7: **The metrics that reflect the *Mode Adaptiveness* and *Tool Effect* of agentic visual reasoning models (%).** The definition of all metrics can be found in Section 2.1. ↑ and ↓ indicate that higher and lower values are better, respectively. The parentheses after the accuracies indicate the proportion of responses that actually use tools under the two different prompts (p_{tool} and p_{text}). $\Delta_{\text{Acc}} =$ *Tool-Available Acc.* $-$ *Tool-Free Acc.* and $\Delta_{\text{TE}} =$ *Tool-Gain* $-$ *Tool-Harm*. Positive and non-positive deltas are shown in green and red, respectively.

Dataset	Method	Accuracy			Mode Adaptiveness			Tool Effects			
		Tool-Available Acc. ↑	*Tool-Free Acc.* ↑	Δ_{Acc} ↑	MA_{tool} ↑	MA_{text} ↑	MA_{mean} ↑	*Tool-Gain* ↑	*Tool-Harm* ↓	Δ_{TE} ↑	*Text-Retain* ↑
HRBench4K	Thyme	76.70 (12.78)	76.88 (0.00)	-0.18	11.52	86.81	49.17	0.20	0.38	-0.18	96.06
	DeepEyesV2	75.33 (98.88)	73.90 (0.00)	+1.43	99.51	1.36	50.44	7.48	6.20	+1.28	100.00
	CodeV	74.05 (61.08)	74.78 (0.23)	-0.73	43.90	33.31	38.61	3.13	3.05	+0.08	83.73
	Metis	84.13 (46.88)	83.80 (0.00)	+0.33	41.77	51.85	46.81	0.75	0.33	+0.43	96.81
	Beacon (Ours)	83.38 (70.65)	80.53 (0.00)	+2.85	97.24	37.18	67.21	9.98	4.48	+5.50	100.00
BLINK	Thyme	57.41 (0.68)	55.35 (0.00)	+2.06	1.00	99.50	50.25	0.16	0.00	+0.16	84.32
	DeepEyesV2	53.83 (99.80)	55.27 (0.00)	-1.43	99.92	0.35	50.14	10.06	9.76	+0.29	87.50
	CodeV	58.56 (14.66)	55.98 (0.00)	+2.58	16.10	85.89	51.00	2.29	1.90	+0.39	79.85
	Metis	62.09 (24.51)	64.67 (0.00)	-2.58	23.09	74.42	48.76	1.27	2.25	-0.98	87.78
	Beacon (Ours)	65.49 (79.79)	64.22 (0.00)	+1.27	93.78	28.30	61.04	13.11	11.15	+1.96	95.28
BabyVision	Thyme	12.27 (0.62)	11.55 (0.00)	+0.72	0.72	100.00	50.36	0.00	0.00	0.00	52.00
	DeepEyesV2	11.39 (99.90)	1.44 (0.00)	+9.95	99.90	0.00	49.95	10.98	0.88	+10.10	–
	CodeV	13.76 (0.57)	14.74 (0.00)	-0.98	0.69	100.00	50.35	0.00	0.00	0.00	35.51
	Metis	15.52 (15.72)	15.93 (0.00)	-0.41	17.65	94.55	56.10	0.62	0.46	+0.15	54.23
	Beacon (Ours)	16.70 (95.36)	15.36 (0.00)	+1.34	95.99	10.77	53.38	6.44	2.42	+4.02	85.71
MathVista	Thyme	71.22 (0.10)	71.64 (0.00)	-0.42	0.08	99.88	49.98	0.00	0.00	0.00	93.57
	DeepEyesV2	72.22 (99.88)	73.68 (0.00)	-1.46	100.00	0.17	50.09	5.32	7.28	-1.96	83.33
	CodeV	71.70 (0.30)	72.08 (0.00)	-0.38	0.76	99.86	50.31	0.04	0.04	0.00	92.01
	Metis	77.36 (17.34)	78.02 (0.00)	-0.66	21.22	83.52	52.37	0.84	1.16	-0.32	95.44
	Beacon (Ours)	77.80 (78.52)	77.18 (0.00)	+0.62	92.34	25.84	59.09	5.98	6.36	-0.38	98.46
TIRBench	Thyme	20.53 (10.45)	19.05 (0.02)	+1.48	8.07	78.57	43.32	0.58	0.25	+0.33	81.21
	DeepEyesV2	15.80 (99.11)	20.15 (0.00)	-4.35	99.22	1.25	50.24	5.43	6.45	-1.02	63.64
	CodeV	18.11 (15.60)	17.04 (0.00)	+1.07	12.76	68.64	40.70	0.82	0.95	-0.13	65.86
	Metis	21.84 (54.32)	21.23 (0.00)	+0.61	59.32	59.23	59.28	2.21	1.28	+0.92	80.66
	Beacon (Ours)	24.28 (92.74)	20.56 (0.00)	+3.72	94.40	12.44	53.42	10.96	6.37	+4.59	75.56
Average	Thyme	47.63	46.89	+0.73	4.28	92.95	48.62	0.19	0.12	+0.06	81.43
	DeepEyesV2	45.71	44.89	+0.83	99.71	0.63	50.17	7.85	6.11	+1.74	–
	CodeV	47.24	46.92	+0.31	14.84	77.54	46.19	1.26	1.19	+0.07	71.39
	Metis	52.19	52.73	-0.54	32.61	72.71	52.66	1.14	1.10	+0.04	82.98
	Beacon (Ours)	53.53	51.57	**+1.96**	94.75	22.91	**58.83**	9.29	6.16	**+3.14**	**91.00**

E.3 Detailed Results of Ablation Study

We provide detailed results of the ablation study in Table 8.



(a) (b) (c)

Figure 9: **Training accuracy and rewards of the RL stage.** **(a)** Mean training accuracy. **(b)** Adaptive reward. **(c)** Format reward.

Table 8: **Detailed ablation results for *Mode Adaptiveness* and *Tool Effect* (%).** The definition of all metrics can be found in Section 2.1.

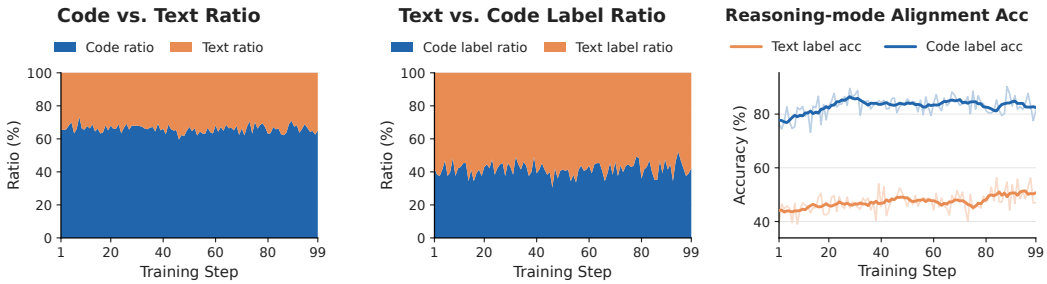
Dataset	Method	Accuracy			Mode Adaptiveness			Tool Effects			
		Tool-Available Acc. ↑	*Tool-Free Acc.* ↑	Δ_{Acc} ↑	MA_{tool} ↑	MA_{text} ↑	MA_{mean} ↑	*Tool-Gain* ↑	*Tool-Harm* ↓	Δ_{TE} ↑	*Text-Retain* ↑
HRBench4K	SFT	81.35 (71.25)	77.68 (0.00)	+3.68	95.04	33.57	64.31	6.33	3.53	+2.80	98.91
	GRPO	82.55 (70.85)	80.10 (0.00)	+2.45	94.29	33.94	64.12	9.23	4.48	+4.75	99.82
	GRPO+NAAR	82.70 (70.33)	80.50 (0.00)	+2.20	97.72	38.00	67.86	9.15	4.53	+4.63	99.22
	GRPO+HCE	83.20 (70.88)	80.00 (0.00)	+3.20	98.05	37.91	67.98	9.65	4.00	+5.65	99.56
	Beacon	83.38 (70.65)	80.53 (0.00)	+2.85	97.24	37.18	67.21	9.98	4.48	+5.50	100.00
BLINK	SFT	64.17 (72.49)	64.80 (0.00)	-0.63	86.17	27.33	56.75	6.61	7.48	-0.87	95.01
	GRPO	65.48 (79.55)	63.91 (0.00)	+1.57	92.79	23.78	58.29	11.06	9.36	+1.69	95.70
	GRPO+NAAR	65.70 (79.66)	64.06 (0.00)	+1.64	93.85	29.13	61.49	11.36	10.11	+1.25	95.37
	GRPO+HCE	65.24 (79.48)	64.18 (0.00)	+1.06	92.70	27.31	60.01	11.29	9.21	+2.08	96.05
	Beacon	65.49 (79.79)	64.22 (0.00)	+1.27	93.78	28.30	61.04	13.11	11.15	+1.96	95.28
BabyVision	SFT	14.95 (97.32)	17.73 (0.00)	-2.78	97.31	8.57	52.94	6.13	3.92	+2.22	75.00
	GRPO	15.10 (94.69)	14.59 (0.00)	+0.52	92.46	8.36	50.41	5.93	5.36	+0.57	68.00
	GRPO+NAAR	16.96 (95.10)	16.60 (0.00)	+0.36	95.79	14.71	55.25	5.72	3.97	+1.75	80.00
	GRPO+HCE	16.91 (94.74)	15.21 (0.00)	+1.70	95.23	9.49	52.36	7.58	4.54	+3.04	82.01
	Beacon	16.70 (95.36)	15.36 (0.00)	+1.34	95.99	10.77	53.38	6.44	2.42	+4.02	85.71
MathVista	SFT	75.76 (76.86)	78.82 (0.00)	-3.06	90.43	22.81	56.62	2.90	6.38	-3.48	97.98
	GRPO	76.60 (78.76)	77.50 (0.00)	-0.90	90.64	23.59	57.12	5.26	7.04	-1.78	98.23
	GRPO+NAAR	77.22 (78.50)	77.68 (0.00)	-0.46	93.95	26.49	60.22	5.64	6.44	-0.80	98.39
	GRPO+HCE	76.30 (78.20)	77.28 (0.00)	-0.98	92.61	26.97	59.79	5.50	7.00	-1.50	98.32
	Beacon	77.80 (78.52)	77.18 (0.00)	+0.62	92.34	25.84	59.09	5.98	6.36	-0.38	98.46
TIRBench	SFT	23.85 (90.83)	20.21 (0.03)	+3.64	93.29	7.46	50.38	10.30	2.93	+7.37	83.06
	GRPO	21.53 (92.12)	20.89 (0.00)	+0.64	93.15	10.01	51.58	9.09	7.36	+1.73	70.03
	GRPO+NAAR	23.65 (93.19)	22.35 (0.00)	+1.30	95.18	12.02	53.60	12.15	6.26	+5.89	71.76
	GRPO+HCE	23.05 (91.80)	22.44 (0.00)	+0.61	93.21	10.14	51.68	11.85	6.34	+5.51	72.69
	Beacon	24.28 (92.74)	20.56 (0.00)	+3.72	94.40	12.44	53.42	10.96	6.37	+4.59	75.56
Average	SFT	52.02	51.85	+0.17	92.45	19.95	56.20	6.45	4.85	+1.61	89.99
	GRPO	52.25	51.40	+0.86	92.67	19.94	56.30	8.11	6.72	+1.39	86.36
	GRPO+NAAR	53.25	52.24	+1.01	95.30	24.07	**59.68**	8.80	6.26	+2.54	88.95
	GRPO+HCE	52.94	51.82	+1.12	94.36	22.36	58.36	9.17	6.22	+2.96	89.73
	Beacon	53.53	51.57	**+1.96**	94.75	22.91	58.83	9.29	6.16	**+3.14**	**91.00**

E.4 Training Dynamics of the RL Stage

In this section, we provide a comprehensive analysis of the training dynamics of the RL process.

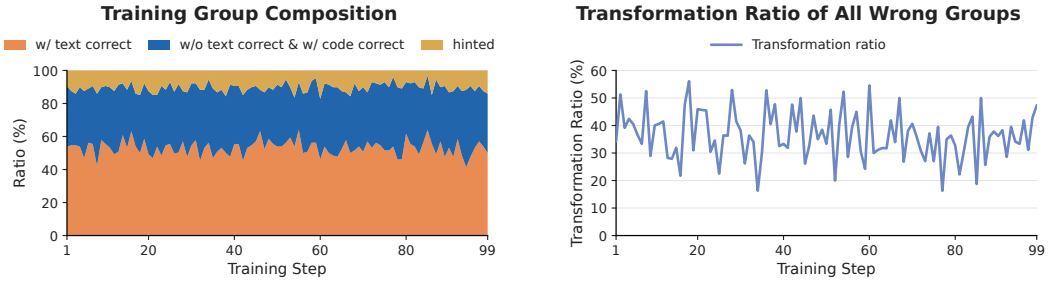
Training rewards and adaptive behavior. We present the reward dynamics during the RL stage in Figure 9, where the training rewards generally improve during training. Figure 10a shows the proportions of pure-text and code-assisted responses, which remain relatively stable throughout training, with no clear shift toward either reasoning mode. Figure 10b further shows that the ratio between the adaptive labels of both reasoning modes remains steady during training. In contrast, Figure 10c reports the reasoning-mode alignment accuracy, defined as the proportion of rollout trajectories whose actual reasoning mode matches the adaptive label assigned to their group. This accuracy consistently improves over the course of training, suggesting that the model gradually learns an adaptive tool-invocation policy rather than simply developing a preference for one reasoning mode.

Training Group Composition and All-Wrong Group Transformation. Figure 11a presents the composition of training rollout groups during the RL stage. We categorize the groups into three types: groups containing at least one correct pure-text response ("w/ text correct"), groups containing no correct pure-text response but at least one correct code-assisted response ("w/o text correct & w/ code correct"), and hint-induced groups whose responses are initially all incorrect ("hinted").



(a) (b) (c)

Figure 10: **(a)** Proportions of pure-text and code-assisted responses during training. **(b)** Proportions of training groups assigned the "text" and "code" adaptive labels for reward computation. **(c)** Accuracy of reasoning-mode selection, measured as the proportion of responses whose reasoning mode matches the adaptive label of their corresponding group.



(a) Composition of training groups. (b) Transformation.

Figure 11: **(a)** Proportions of groups containing at least one correct pure-text response ("w/ text correct"), groups containing no correct pure-text response but at least one correct code-assisted response ("w/o text correct & w/ code correct"), and hint-induced groups ("hinted"). **(b)** Proportion of initially all-incorrect groups that are converted into groups with non-zero accuracy advantages through hint-guided re-rollouts.

These three categories account for relatively stable proportions of approximately 50%, 35%, and 15%, respectively. Figure 11b further reports the proportion of initially all-incorrect groups that are converted into groups with non-zero accuracy advantages through hint-guided re-rollouts. Approximately 40% of such groups can be successfully recycled by our HCE mechanism, demonstrating its effectiveness in recovering useful learning signals from otherwise uninformative rollout groups.

F Discussion

F.1 Limitations

Our diagnostics have several limitations. First, the "text-easy" and "text-hard" sets are defined from a finite number of stochastic rollouts and are model-dependent; we mitigate this issue with five rollouts and additionally report results on a fixed base-model split, but the categories remain an empirical proxy for difficulty. Second, our implementation studies one tool interface—a sandboxed Python environment—so the findings may not transfer directly to web, retrieval, or embodied tools. Third, the HCE module relies on Gemini 3.1 Pro to generate training-time hints, which could be potentially costly. Extending the evaluation to causal tool-effect estimates, broader tool ecosystems, and explicit efficiency measurements is important future work.

F.2 Failed Attempts

We also explored a more aggressive adaptive reward scheme. In addition to the standard rollout group, we force the model to generate a group of pure-text reasoning trajectories. If either the standard rollout group or the forced-text rollout group contains at least one correct pure-text trajectory, the adaptive label is assigned as "text." However, this strategy strongly biases the policy toward pure-text reasoning, causing the code-use ratio to collapse and substantially degrading performance on perception-intensive benchmarks such as V^* and HRBench.

The root cause is that the additional forced-text rollouts artificially increase the probability of observing at least one successful pure-text trajectory, even when pure-text reasoning is not the most reliable strategy on average. Consequently, the group-level label reflects the best outcome across multiple attempts rather than the reasoning mode most likely to succeed in a single rollout. This introduces a mismatch between training and evaluation: a problem may produce one correct pure-text trajectory among multiple training rollouts, yet still have a low probability of being solved by a single pure-text rollout at test time. As a result, the model becomes overly biased toward pure-text reasoning, leading to inferior test-time performance.

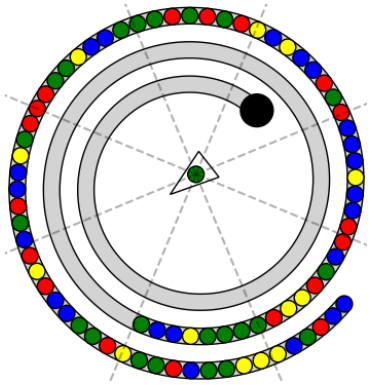
G Case Study

G.1 Reasoning Trajectory Examples

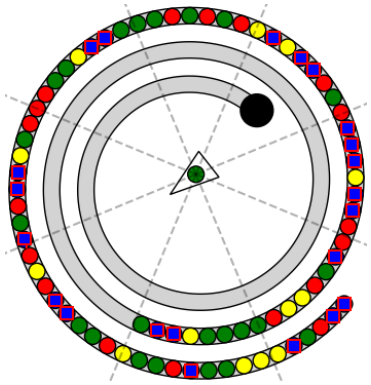
As illustrated in Tables 9, 10, and 11, Beacon effectively leverages a diverse set of visual tools to solve complex reasoning tasks.

G.2 Hint Examples

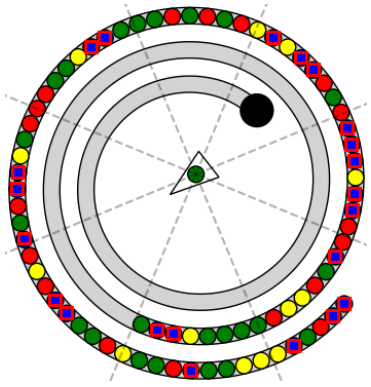
Tables 12, 13, and 14 demonstrate that training-time hint supervision enables Beacon to extend its reasoning capabilities and tackle more challenging tasks.



Original Image (q0)



g0



g1

Question
How many blue marbles are there on the track?

Beacon Reasoning

```
<tool_call>
img = get_image("q0")
img.size
</tool_call>
<tool_response>
[352, 369]
Available image refs:  q0
</tool_response>
<observation>
The image size is 352x369.  I will now
write a script to detect the blue marbles
by finding connected components of blue
pixels and filtering them based on size
and color.
</observation>
<tool_call>
import numpy as np
from PIL import Image
img = get_image("q0")
img_np = np.array(img)
# Create a mask for blue pixels
blue_mask = (img_np[:, :, 0] < 100) &
    (img_np[:, :, 1] < 150) &
    (img_np[:, :, 2] > 200)
# Simple connected components using BFS
def count_components(mask):
    visited = np.zeros_like(mask,
dtype=np.bool_)
    count = 0
    h, w = mask.shape
    for i in range(h):
        for j in range(w):
            if mask[i, j] and not visited[i,
j]:
                count += 1
                queue = [(i, j)]
                visited[i, j] = True
...
Available image refs:  q0, g0, g1
</tool_response>
<observation>
The bounding boxes for the 22 valid
components perfectly enclose all the blue
marbles on the track.  There are no missed
marbles and no false positives.  The count
is exactly 22.
</observation>
<answer>
22
</answer>
```

Beacon Answer
22 (✓)

Table 9: **Reasoning trajectory example 1: Pixel-level computation.**



Original Image (q0)



g0



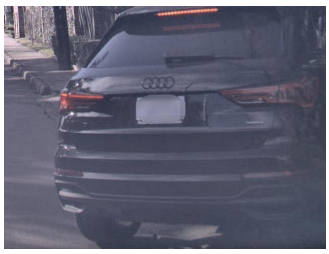
g1



g2



g3



g4

Question
How many cars are white in this picture?

Beacon Reasoning

```
<tool_call>
img = get_image("q0")
img.size
</tool_call>
<tool_response>
[1448, 938]
Available image refs:  q0
</tool_response>
<observation>
The image size is 1448x938.  I will crop
the middle section to get a better look at
the cars in the distance.
</observation>
<tool_call>
img = get_image("q0")
cropped = img.crop((300, 450, 750, 550))
cropped
</tool_call>
<tool_response>
Returned image ref:  g0
Available image refs:  q0, g0
</tool_response>
<observation>
In the cropped image, I can see several
cars.  Let's look closer at the cars in
the distance to determine their colors.
</observation>
<tool_call>
img = get_image("q0")
cropped = img.crop((600, 450, 800, 550))
cropped
</tool_call>
<tool_response>
Returned image ref:  g1
Available image refs:  q0, g0, g1
...
Returned image ref:  g4
Available image refs:  q0, g0, g1, g2, g3,
g4
</tool_response>
<observation>
The car on the right side of the
foreground is dark grey or black.  So, the
only white cars are the two parked on the
right side of the street in the distance.
Therefore, there are 2 white cars in the
picture.
</observation>
<answer>
B
</answer>
```

Beacon Answer
B (✓)

Table 10: **Reasoning trajectory example 2: Visual search and counting.**



Original Image (q0)


Original Image (q1)


Original Image (q2)



g0

Question
Given the first image with the lower right corner missing, can you tell which one of the second image or the third image is the missing part?

Beacon Reasoning

```
<tool_call>
import numpy as np
from PIL import Image
img1 = get_image("q0")
img2 = get_image("q1")
img3 = get_image("q2")
print(f"Image 1 size:  {img1.size}")
print(f"Image 2 size:  {img2.size}")
print(f"Image 3 size:  {img3.size}")
# Let's crop the bottom-right corner of
Image 1 and compare
# it with Image 2 and Image 3.
# The missing part is a black rectangle in
the lower-right
# corner.  Let's find its coordinates.
</tool_call>
<tool_response>
Image 1 size:  (400, 222)
Image 2 size:  (200, 111)
Image 3 size:  (200, 111)
Available image refs:  q0, q1, q2
</tool_response>
<observation>
The first image has a black rectangle
in the lower-right corner (from x=200
to 400, y=111 to 222).  The second and
third images are exactly half the size
of the first image.  I will extract the
bottom-right quadrant of the first image
and compare it with the second and third
images.
...
<observation>
Pasting Image 3 into the missing spot of
Image 1 creates a seamless and logical
continuation of the protest scene.  The
banner with "cebo" crossed out perfectly
aligns with the text seen in Image 1, and
the hands of the protesters also align
well.  The second image, showing two
men, would not fit this context at all.
Therefore, Image 3 is the correct missing
part.
</observation>
<answer>
B
</answer>
```

Beacon Answer
B (✓)

Table 11: **Reasoning trajectory example 3: Jigsaw.**



Original Image (q0)



g0



g1



g2



g3

Question

In which year was this event held? The player in the blue jersey is from India, and she is executing a badminton smash. If the shuttlecock lands inside the red circle, what will the new match score be?

Hint

1. [Instruction] Crop the scoreboard and red circle areas from the image; [Expected Subgoal] Determine the current score, event name, and the boundary status of the red circle;
2. [Instruction] Calculate the new score based on the out-of-bounds rule and identify the year of the event; [Expected Subgoal] Find the updated match score and the year the event took place;

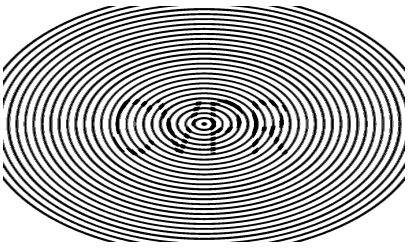
Beacon Reasoning Given the Hints

```
<tool_call>
img = get_image("q0")
...
<tool_call>
from PIL import ImageDraw
annotated = get_image("q0").copy()
draw = ImageDraw.Draw(annotated)
draw.line((0, 385, 928, 385),
fill="yellow", width=2)
draw.line((0, 420, 928, 420),
fill="yellow", width=2)
annotated
</tool_call>
<tool_response>
generated_images:  g3
Original image refs:  q0
Generated image refs:  g0, g1, g2, g3
</tool_response>
<observation>
The red circle is located at the
bottom-right corner of the court, which
is outside the singles court boundary
and in the alley.  A smash that lands in
the alley is considered out of bounds,
so it is a fault for the player who hit
it.  The current score is IND 20, TPE 0
15.  This means IND has won Game 1 and is
serving in Game 2.  The score for Game 2
is currently 20--15.  If the player in
blue (IND) smashes out of bounds, TPE wins
the point.  The new score will be IND 1
20, TPE 0 16.
</observation>
<answer>
The new match score will be IND 1 20, TPE
0 16.  The event was held in 2023.
</answer>
```

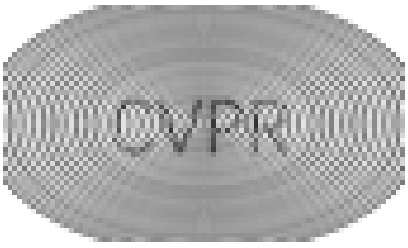
Beacon Answer

The new match score will be IND 1 20, TPE 0 16. The event was held in 2023. (✓)

Table 12: **Hint example 1: A real-world agentic problem.**



Original Image (q0)



g0

Question
Identify all hidden text content in the image

Hint
1. Hint: [Instruction] Resize the image to a much smaller scale (e.g., 1/10th or 1/20th of the original size); [Expected Subgoal] Reveal any hidden text or patterns that become visible at a lower resolution;

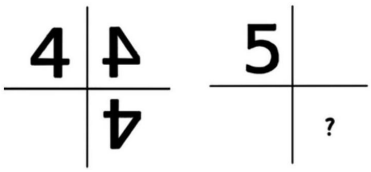
Beacon Reasoning Given the Hints

```
<tool_call>
img = get_image("q0")
img_resized = img.resize((img.width // 20,
img.height // 20))
img_resized
</tool_call>
<tool_response>
generated_images:  g0
</tool_response>
...
Original image refs:  q0
Generated image refs:  g0
</tool_response>
<observation>
The resized image clearly shows the
text "CVPR" in the center of the spiral
pattern.  The text was hidden by the
high-frequency spiral lines at the
original resolution.
</observation>
<answer>
CVPR
</answer>
```

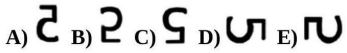
Beacon Answer
CVPR (✓)

Table 13: **Hint example 2: Low-resolution OCR.**

<image1>:



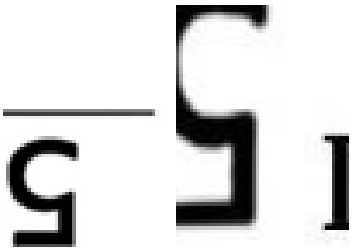
<image2>:



Original Image (q0)



g0



g1

Question

The number 4 is reflected twice in the picture. What appears in the field with the question mark if we do the same with the number 5?

Hint

1. [Instruction] Observe the transformation applied to the number 4, noting that two reflections are equivalent to a 180-degree rotation; [Expected Subgoal] Understand the geometric transformation rule to apply to the number 5;
2. [Instruction] Crop the number 5 from the image and rotate it 180 degrees using a tool; [Expected Subgoal] Obtain the visual result of applying the same transformation to the number 5;
3. [Instruction] Compare the rotated number 5 with the given options A through E; [Expected Subgoal] Identify which option matches the transformed number 5;

Beacon Reasoning Given the Hints

```
...
img = get_image("q0")
rotated_5 = img.crop((420, 130, 580,
350)).rotate(180)
# Option C is around x=380 to 450, y=580
to 650
opt_c = img.crop((380, 580, 450, 650))
# Create a side-by-side comparison
comp = Image.new('RGB', (100, 70))
comp.paste(rotated_5.resize((50, 70)), (0,
0))
comp.paste(opt_c.resize((50, 70)), (50, 0))
comp
</tool_call>
<tool_response>
generated_images:  g1
</tool_response>
...
Original image refs:  q0
Generated image refs:  g0, g1
</tool_response>
<observation>
The side-by-side comparison shows that
the rotated 5 perfectly matches Option
C. The top bar is at the bottom, pointing
left.  The vertical stem is on the right,
pointing up.  The belly is on the left,
curving down and right.  This is exactly
what happens when you rotate a 5 by 180
degrees.  The question asks "What appears
in the field with the question mark if
we do the same with the number 5?" The
transformation is two reflections, which
is equivalent to a 180-degree rotation.
Therefore, the result is Option C.
</observation>
<answer>
C
</answer>
```

Beacon Answer

C (✓)

Table 14: **Hint example 3: Visual puzzles.**