# CodeV: Code with Images for Faithful Visual Reasoning via Tool-Aware Policy Optimization

Xinhai Hou[1*] Shaoyuan Xu[2*] Manan Biyani[2] Moyan Li[2] Jia Liu [3]
Todd C. Hollon [1] Bryan Wang[2†]

[1]University of Michigan [2]Amazon.com [3]The Ohio State University
[†]Corresponding Author [*]Equal contribution

https://github.com/RenlyH/CodeV

## Abstract

*Agentic vision–language models are increasingly trained to "think with images" by calling image operations. However, we show that high final-answer accuracy often hides unfaithful visual reasoning: models may invoke tools on irrelevant regions or ignore tool outputs entirely, yet still guess the correct answer. In this work, we first propose a faithfulness evaluation protocol that measures whether intermediate visual tool outputs (e.g., crops) actually contain the queried evidence. This reveals that recent visual agents achieve high final-answer accuracy but exhibit low rates of faithful tool-use on visual search benchmarks. We then introduce CodeV, a code-based visual agent trained with Tool-Aware Policy Optimization (TAPO). TAPO is a process-level RL framework that augments GRPO with dense rewards defined directly on visual tool inputs and outputs, rather than on chain-of-thought tokens, making supervision easier to verify and less susceptible to reward hacking. CodeV represents visual tools as executable Python code, and TAPO assigns step-wise rewards based solely on the question and tool output, encouraging both necessary and evidence-consistent tool use. In a two-stage SFT+RL pipeline, CodeV achieves competitive or superior accuracy while substantially increasing faithful tool-use rates on related visual search benchmarks. Beyond visual search, CodeV attains strong performance on a range of multimodal reasoning and math benchmarks, suggesting that explicitly supervising intermediate tool behavior is crucial for building trustworthy, agentic visual reasoning systems.*

## 1. Introduction

Large language models (LLMs) and vision–language models (VLMs) now tackle challenging tasks that require extended thinking and tool–integrated reasoning: web search, code execution, and agentic planning for complex goals [2, 5, 9–12, 18, 21, 30, 31, 41–43, 45]. In the visual domain,

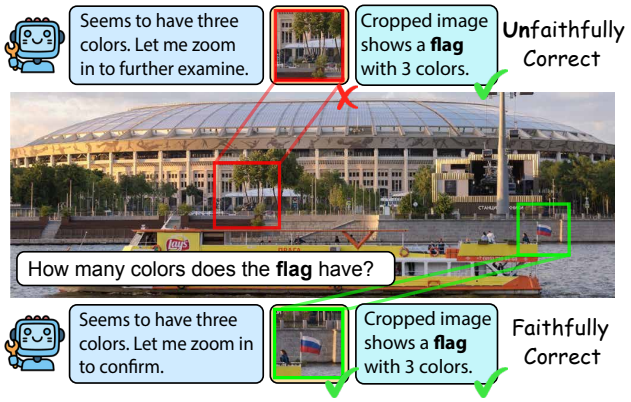



Figure 1. An example of visual agentic system generating "unfaithful" trajectory: the cropping tool is used at the wrong region with unfaithful analysis but leads to correct answer.

current visual systems explicitly "think with images" [21], interleaving reasoning with visual actions such as cropping, segmentation, and other image operations to ground answers in tool responses [6, 7, 28], yielding notable gains on compositional tasks such as visual reasoning [17], visual search [38], and chart reasoning [36]. This agentic reasoning paradigm consistently outperforms single-shot prediction and offers a scalable path to reliable, evidence-based solutions. However, many of the most capable agentic systems are based on proprietary models (GPT-4o, o3). In this work, we therefore focus on training open-source VLMs to acquire similar agentic visual tool-use behavior.

Reinforcement learning with verifiable rewards (RLVR) has emerged as a central approach for eliciting complex reasoning [5, 9, 31]. Recent work extends RLVR to agentic visual reasoning, where agents learn to interleave image operations (e.g., cropping, segmentation, depth estimation) with textual reasoning [8, 26, 27, 47]. Exposing images through a code interpreter further allows the model to invoke a rich library of image operations (e.g., crop, rotate) and leverage its visual capability over multiple tool

calls, as demonstrated by OpenAI o3 [21] and related works [44]. This code-based interface avoids reliance on external, heavyweight tool API and leverages the code patterns the model has already seen at scale during pretraining, making tool use both more expressive and more natural. Despite these advances, we observe a key failure mode in current open-source visual agents. Models such as DeepEyes [47] and Pixel-Reasoner [26] can score highly on visual search benchmarks while relying on incorrect or misaligned tool calls (e.g., cropping the wrong region but still answering correctly; Figure 1). This motivates a faithfulness analysis in Section 3, where we systematically evaluate *visual tool use* by checking whether intermediate image operations (e.g., crops) actually contain the object or region referenced in the question, thereby measuring action-level faithfulness for visual search.

Given these experimental results, we hypothesize that the observed misalignment arises from reward hacking [25], driven by two key issues in reward design: (1) outcome dominance and (2) reward sparsity. Specifically, current rewards are *outcome-dominant*: they prioritize final accuracy or whether a tool was invoked, while providing no supervision on how the tool was used [4, 47]. Lacking step-level credit assignment, policies quickly exploit this reward signal by hallucinating tool use or performing meaningless operations [32]. This mirrors the patterns in web-based agents that issue superficial search queries or clicks while largely ignoring retrieved evidence [13, 20, 41, 43]. Second, the reward is *sparse*, producing unstable optimization dynamics. Early-stage rollouts on difficult tasks frequently receive zero rewards, which discourages the faithful use of tools with partially correct attempts. Although reward shaping or invocation bonuses push the policy to use tools, they also introduce reward hacking, encouraging unfaithful behaviors such as unnecessary image operations and trivial computations [16, 47, 48]. As models scale up and reinforcement learning (RL) training intensifies, these issues become increasingly pronounced, amplifying the tendency for models to optimize for superficial success signals rather than genuine reasoning faithfulness. These observations of unfaithful tool use and unstable training motivate the central research question of this paper:

> How do we incentivize vision–language models to produce faithful reasoning grounded in tool outputs?

To answer this question and elicit more powerful coding capability, we rethink RLVR from a *process reward* perspective and propose Tool-Aware Policy Optimization (TAPO), which treats tool use as a sequence of verifiable decisions. Rather than rewarding only the final answer, TAPO leverages a rule-based check and a judge model to assign *step-level rewards* to each *tool output* that requires context of question and answers to align. TAPO couples the scalability of GRPO with dense, process-aware signals and avoids reliance on training a separate, hack-prone reward model to monitor unverifiable chain of thought. The TAPO-based reinforcement learning (RL) is further integrated with a cold-start instruction tuning phase to form a two-stage curriculum training framework, based on which wWe adopt and further curate the commonly used instruction tuning and RL datasets [35, 44, 47]. Trained on these datasets, our model CodeV-7B, built on top of Qwen2.5-VL-7B [1], yields stable, faithful tool use, reduces tool-related failure cascades, and achieves strong performance across 10 challenging benchmarks, narrowing the gap to proprietary systems [9, 21]. The key contributions of our work are summarized as follows:

1. **Tool-Aware Policy Optimization (TAPO).** We introduce TAPO, a novel policy optimization method that augments GRPO with dense process rewards for tool use necessity and tool output consistency. TAPO delivers markedly better sample efficiency and training stability.
2. **Faithfulness evaluation for visual agents.** We propose a protocol that detects unfaithful visual reasoning, yielding quantitative faithfulness metrics beyond accuracy.
3. **Empirical gains in faithfulness and accuracy.** Training CodeV with TAPO largely improves tool faithfulness for the visual reasoning task and demonstrates strong performance on 10 challenging benchmarks.
4. **Open recipe and resources.** We release the full training recipe for CodeV, including the implementations of TAPO, along with datasets and the corresponding model to facilitate reproduction and extension.

## 2. Preliminaries

### 2.1. Reinforcement Llearning for Rreasoning LLMs

RLVR [15, 24] has proven effective for eliciting multi-step reasoning in LLMs [5, 9, 37]. In this paradigm, the model generates its own trajectories, receives feedback from the reward signal, and iteratively refines its policy. Formally, given a dataset $\mathcal{D}$ of prompts $q$, the policy $\pi_\theta$ generates an output $o$ and receives a reward $R(q, o)$. We optimize:

$$\max_{\pi_\theta} \mathbb{E}_{x\sim\mathcal{D},\, o\sim\pi_\theta(\cdot|q)} R(q,o) - \beta\, \mathbb{D}_{\mathrm{KL}}\big(\pi_\theta(o|q) \,\|\, \pi_{\mathrm{ref}}(o|q)\big), \tag{1}$$

where $\mathbb{D}_{\mathrm{KL}}$ serves as a regularization penalty that constrains the updated policy to stay close to the reference policy, thereby stabilizing training in policy-gradient methods such as PPO and GRPO. In standard outcome-only setups, $R$ is used at the final answer, providing no signal on which intermediate tool decisions mattered, which becomes spurious in agentic RL, particularly when visual actions are required. The KL term keeps $\pi_\theta$ close to the reference policy, which stabilizes training and preserves useful SFT priors such as sensible tool-usage patterns.

### 2.2. Agentic Visual Reasoning

We consider the problem of answering visual questions through iterative reasoning with tool use. Given an input $\mathrm{x} = (V, Q)$ comprising an image $V$ and a textual query $Q$, a policy model $\pi_\theta$ generates a trajectory for at most $T$ turns of conversation: $\tau = (\mathbf{x}, a_1, o_1, a_2, o_2, \ldots, a_T)$ where each action $a_t \sim \pi_\theta(\cdot \mid \mathbf{x}, h_{t-1})$ conditions on the input and history $h_{t-1} = \{(a_i, o_i)\}_{i=1}^{t-1}$. Actions $a_t$ at $t$ turn belong to one of the three categories: `<think>` free-form textual thoughts. `<tool_call>` executable token that invokes a tool. `<answer>` final answer that ends the trajectory. The observation $o_t$ is available only when $a_t$ is a `<tool_call>` that produces an output (e.g., transformed images, numbers, or errors); for `<think>` and `<answer>` actions, we set $o_t$ to null. This unified action space enables the flexible interleaving of reasoning and tool use, allowing the model to dynamically adjust its strategy based on intermediate results rather than follow a fixed workflow. Our formulation treats agentic visual reasoning as a single, cohesive decision process, where each action is conditioned on the full trajectory history, including all previous execution outcomes.

Most recent works follow a two-stage curriculum pipeline [3, 4, 8, 14, 16, 21, 26, 44]: supervised fine-tuning (SFT) and then a reinforcement learning stage. SFT exposes the model to high-quality traces in which tools are invoked, and their outputs are woven into the reasoning, teaching basic patterns of tool use. Many papers highlight the importance of high-quality and diverse trajectory data, such as trial-and-error with self-reflection [26]. Then, RL is used to optimize an explicit reward, typically based on final-answer correctness, along with a tool invocation bonus, encouraging exploration under distribution shift while consolidating high-reward behaviors on the target data distribution. We acknowledge the importance of SFT as a cold-start step. However, without proper handling at the RL stage, it can easily diverge and result in unfaithful trajectories.

## 3. Evidence of Unfaithful Tool Use in VLMs

Faithful reasoning is central to building trustworthy agentic AI systems. When a model performs agentic visual reasoning and invokes visual tools, its intermediate actions should accurately reflect and depend on the information returned by those tools. As shown in Figure 1, without such alignment, a model can produce correct answers for the wrong reasons, undermining both interpretability and reliability. In the visual setting, this lack of faithfulness makes it unclear whether the model genuinely "sees" the evidence or is instead exploiting text cues to produce plausible outputs.

To study this behavior, we design experiments that quantitatively evaluate how faithfully models ground their reasoning in *visual tool use*. Rather than inspecting model-generated chain-of-thought tokens, we focus on the consequences of tool actions, which are easier to verify. Specifically, we test whether intermediate tool outputs (e.g., cropped images) remain relevant to the question, providing a direct proxy for whether the model's reasoning follows the intended problem-solving path.

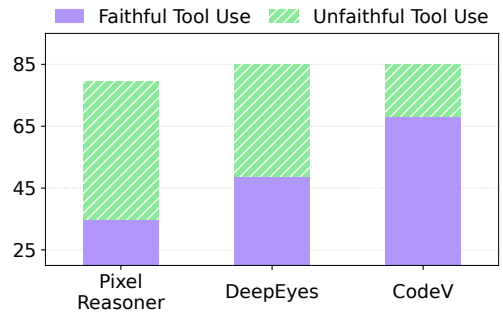



Figure 2. **Faithfulness conditioned on correct answers in V* [38] Benchmark.** For this visual search problem, crop is treated as the most effective tool use [26, 46, 47]. Therefore, we define faithful tool use as cropped image from tool use capturing *any* target object and evaluate how many correct answers are also faithful, as shown in violet. For the remaining correct answers, the tool uses do not capture the target object and are treated as unfaithful, as shown in mint. Recent visual agents achieve high final-answer accuracy but fail to use tools faithfully. CodeV shows great improvement in faithfulness with no decrease in accuracy.

**1) Baselines and benchmarks.** We evaluate two state-of-the-art recently open-sourced models, DeepEyes [47] and Pixel-Reasoner [26], which are trained for agentic visual reasoning through cropping-based tool use. Experiments are conducted on visual search benchmarks where a genuine answer should localize a small object in a high-resolution image and then describe its attributes or relations. We primarily use V* [38], and additionally report results on HRBench-4K [34] and Thyme [44] in Section 5.3.

**2) Evaluation setup.** To diagnose the faithfulness of visual tool use in these models, we evaluate whether intermediate cropped images actually contain the object or region referenced in the question. This directly targets their cropping-based design, where image operations are the core mechanism for grounding reasoning in visual evidence [46]. Concretely, for each example, we provide the original question together with an *intermediate* tool-output crop to GPT-4o, which we use as a judge, and prompt it to decide whether the crop includes the queried object or region. When a model produces multiple crops, we regard the visual tool use as faithful if *any* crop is judged to contain the target. We then report the proportion of faithful versus unfaithful tool calls, conditioned on the final answer being correct. A higher faithful tool-use rate reflects a more reliable, action-level alignment between the model's reasoning, the intended visual evidence, and the task objective. Detailed evaluation settings, including prompt construction, are provided in the

Appendix 8.

**3) Results.** Figure 2 shows the proportions of *faithful* and *unfaithful* tool calls under the condition that the final answer is correct. While they have a relatively high score on final answer accuracy, only about half of the intermediate tool calls are judged to be faithful to the original question (57% for DeepEyes and 43% for Pixel-Reasoner).

**4) Takeaways.** These results reveal that high final-answer accuracy does *not* guarantee strong agentic visual reasoning: models can often guess the correct answer from text cues like answer options, while using visual tools in unfaithful ways. In other words, current benchmarks can substantially overestimate the underlying visual tool-use capability. This suggests that the training objective should not only ensure final correctness but also enforce the correctness of tool use, both when constructing SFT dataset and when designing RL rewards. Aligning intermediate tool use with the intended evidence is, therefore, crucial for developing genuinely agentic and trustworthy visual reasoning systems.

# 4. Methodology

In this section, we instantiate the general agentic visual reasoning formulation in Section 2.2 with an agent that generates Code for Visual reasoning (CodeV), and describe our Tool-Aware Policy Optimization (TAPO) framework.

## 4.1. Overview

Figure 3 summarizes the training pipeline of CodeV. It consists of two main stages: supervised fine-tuning (SFT) and reinforcement learning (RL) with Tool-Aware Policy Optimization (TAPO). The goal is to optimize the model for both faithfulness and accuracy in visual reasoning tasks.

In the first stage, SFT initializes the model by training it on datasets involving image operations (e.g., cropping, rotating) and multi-round refinement tasks. This stage ensures efficient tool use and minimizes reliance on direct answers, with the learned knowledge serving as the reference policy.

In the second stage, similar to GRPO, TAPO performs on-policy rollouts and rewards are computed using a Monte Carlo-based baseline. TAPO balances tool use and accuracy, guiding the model to generate faithful tool steps while avoiding unnecessary or unfaithful actions.

The reward system combines accuracy and faithfulness, rewarding useful tool outputs (e.g., relevant crops) and penalizing "lazy" tool use, such as large, uninformative crops or incorrect tool operations.

## 4.2. Code-based agentic rollouts

As in Section 2.2 and Figure 3, a rollout is

$$\tau = (\mathbf{x}, a_1, o_1, \ldots, a_T), \quad \mathbf{x} = (V, Q), \tag{2}$$

where $a_t \sim \pi_\theta(\cdot \mid \mathbf{x}, h_{t-1})$ and $h_{t-1} = \{(a_i, o_i)\}_{i=1}^{t-1}$. In CodeV, each $a_t$ is one of:

- `<think>`: free-form text for intermediate reasoning;
- `<code>`: a delimited Python program that operates on $V$;
- `<answer>`: final answer that terminates $\tau$.

Only completed `<code>` blocks are executed in a restricted Python sandbox with read-only access to $V$ and a small set of deterministic image / math utilities. Execution returns an observation $o_t$ (logs and optional derived images) that is appended to the context; for `<think>` and `<answer>` we set $o_t = \varnothing$. Thus, tool use is fully represented inside the token-level policy over this mixed action space, without an external controller. Implementation details of the sandbox are deferred to Appendix 13.

## 4.3. TAPO: Tool-Aware Policy Optimization

We adopt a two-stage SFT+RL pipeline to train CodeV.

**Stage 1: SFT.** We highlight the importance of the cold-start SFT stage. We find that popular instruct-tuned models such as Qwen2.5-VL cannot transform the knowledge of coding into problem-solving capabilities: when performing RL with these models, rollouts using code on average have a lower correct rate than direct answering. The model will quickly converge to pure text reasoning and fail to conduct agentic visual reasoning. To elicit the knowledge of "Code with Image", the model needs to be cold-started via SFT. The SFT model serves as both initialization and as the reference policy $\pi_{\text{ref}}$ in Eq. (1).

**Stage 2: GRPO with group baselines.** We then perform GRPO-style optimization on on-policy rollouts. For each prompt group $g$ (corresponding to a single image–question pair), we sample $K$ trajectories $\{\tau_k^{(g)}\}_{k=1}^K \sim \pi_\theta(\cdot \mid g)$, compute their rewards $R(\tau_k^{(g)})$, and define a *group baseline* as a simple Monte Carlo critic as $b(g) = \frac{1}{K}\sum_{k=1}^K R(\tau_k^{(g)})$. Given a trajectory $\tau$, let $g(\tau)$ denote its associated prompt group. We define the trajectory-level advantage as $A_t = R(\tau) - b\big(g(\tau)\big)$, and broadcast this value to all time steps along the trajectory for every time step $t$ in $\tau$.

For every token-level action $a_t$ in state $s_t$ (the full multimodal history up to time $t$), we form the importance ratio

$$r_t(\theta) = \frac{\pi_\theta(a_t \mid s_t)}{\pi_{\text{ref}}(a_t \mid s_t)}. \tag{3}$$

TAPO optimizes a clipped GRPO objective

$$\mathbb{E}_{\tau,t}\Big[\min\big(r_t(\theta)\,A_t,\ \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\,A_t\big)\Big] - \beta\,\mathbb{E}_q\Big[\mathbb{D}_{\text{KL}}\big(\pi_\theta(\cdot \mid q)\big\|\pi_{\text{ref}}(\cdot \mid q)\big)\Big], \tag{4}$$

which instantiates Eq. (1) with group-wise advantages and PPO-style clipping applied to trajectories that contain both language and code actions. Under high-temperature sampling, group sampling implicitly attributes credit: since trajectories differ at certain steps and better trajectories yield higher $R(\tau)$, the resulting advantage $A(\tau)$ will selectively increase the probability of those decisive tool-use steps.

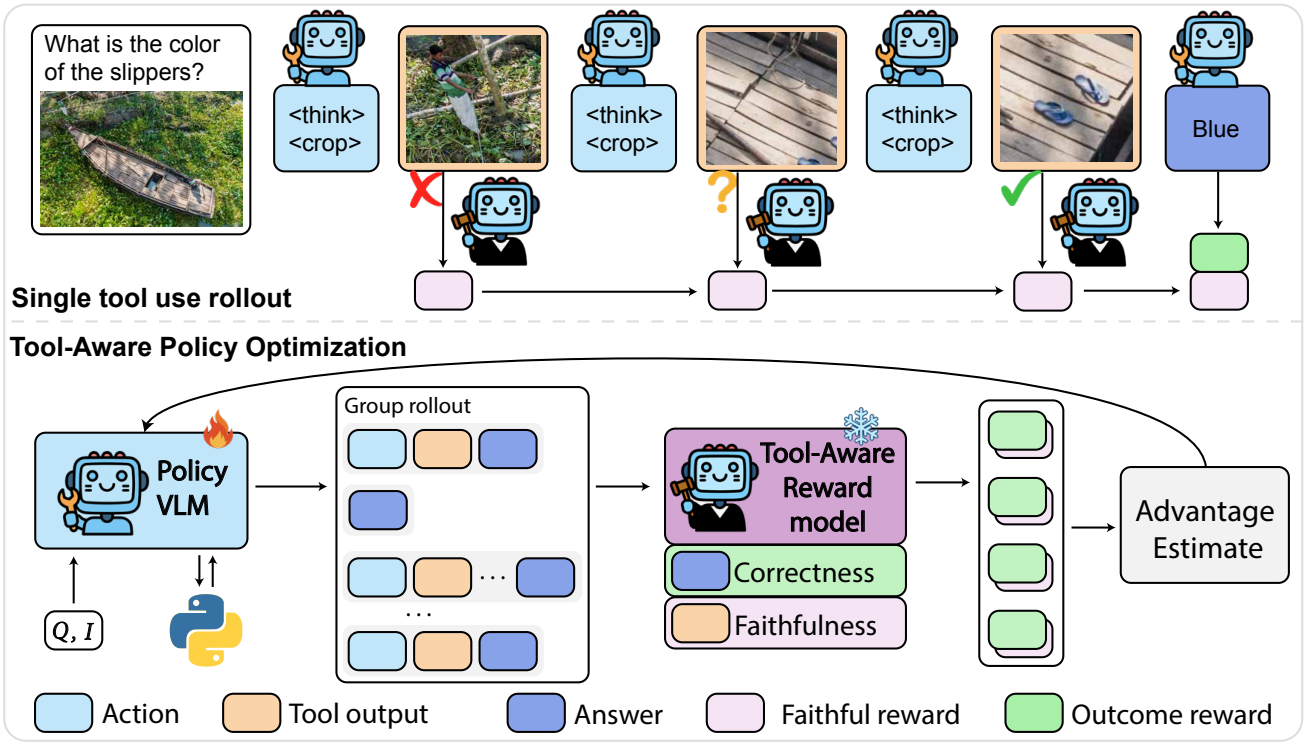



Figure 3. Overview of the CodeV rollout and Tool-Aware Policy Optimization (TAPO). The model processes an image $I$ and question $Q$ pair, using tools like cropping to generate intermediate results for its final answer. Tool faithfulness will be scored by a reward model. For the tool like cropping, reward model will score $r^{\text{tool}}$ based on the observability of the target object in the cropped image. The final answer correctness will be used as outcome reward. The policy VLM is fine-tuned with tool-aware policy optimization, a GRPO-style reinforcement learning approach. The policy VLM will conduct multiple rollouts for the same $Q$ and $I$ with tool use. These rollouts will be scored by the hybrid reward system that combines faithfulness and correctness. Final reward will be normalized within the group and used to estimate relative advantage for the policy VLM to update.

**Reward design for TAPO.** TAPO employs a hybrid reward that combines final-answer correctness with the faithfulness of visual tool use:

$$R(\tau) = \lambda_{\text{acc}}\, r^{\text{acc}}(\tau) + \lambda_{\text{tool}}\, r^{\text{tool}}(\tau), \tag{5}$$

where $r^{\text{acc}}$ measures answer quality and $r^{\text{tool}}$ evaluates tool steps, and $\lambda_{\text{acc}}, \lambda_{\text{tool}} \in [0, 1]$ are weighting coefficients. Let $\mathcal{T}_{\text{tool}}$ be the set of indices where $a_t$ is a <code> action and aggregate

$$r^{\text{tool}}(\tau) = \frac{1}{|\mathcal{T}_{\text{tool}}|} \sum_{t \in \mathcal{T}_{\text{tool}}} r_t^{\text{tool}}. \tag{6}$$

**Answer reward.** For $r^{\text{acc}}$, we use exact or programmatic matching when ground-truth answers are available (e.g., numeric or categorical VQA), and an LLM-as-a-judge for open-ended tasks. In open-ended visual reasoning, final correctness already relies on an external judge [44, 47], so TAPO reuses the same infrastructure without extra deployment burden.

**Tool reward as evidence checking.** Early experiments showed that directly judging the *model's* code and thoughts is both brittle and expensive. Instead, TAPO only inspects *non-model-generated* context: $(Q,\ o_t,\ \text{metadata})$, where $Q$ is the original question and $o_t$ is the sandbox output (cropped image or its description, coordinates, simple statistics). The judge template is asked whether the current tool result provides *any* useful evidence for answering $Q$. In particular, the rubric: (i) encourages any crop that clearly contains a relevant object or region; and (ii) does not require the crop to include all relevant objects. For questions involving multiple targets, a trajectory that isolates at least one correct region is rewarded rather than penalized.

This evidence-based view is easier to verify than diagnosing internal reasoning: the judge never sees the chain-of-thought or code and only answers "does this piece of evidence help with this question?". IoU-based rewards against ground-truth boxes would be an alternative, but they require dense annotations and do not generalize to tasks without bounding boxes. Our judge-based formulation works on top of arbitrary tool outputs and can be reused for other task families with minimal changes. As for the choice of

judge model, we leverage Qwen2.5-VL-32B as our judge with ablation on GPT-5-nano with results shown in 5.4.

**Redlines and discouraging lazy crops.** To avoid reward hacking and trivial large crops, we combine 2 mechanisms:

- **SFT shaping.** SFT-stage already teaches localized, high-resolution crops and multi-round refinement: large, uninformative boxes are rare in the training traces. RL fine-tunes around this prior rather than discovering cropping behavior from scratch.
- **Judge rubric.** For each crop, the judge is instructed to focus on whether the *main content* of the cropped image clearly contains the object or region needed to answer $Q$. Extremely large or cluttered crops, where the target is small or barely visible, typically fail this check and receive low $r_t^{\text{tool}}$. We also treat obvious misuse of tools (e.g., using image read/save as scratchpad) as red flags and assign negative reward.

Collectively, these techniques discourage lazy crops. At the same time, we keep $|\lambda_{\text{tool}}| < |\lambda_{\text{acc}}|$, so global answer correctness remains the dominant training signal.

**Tasks without explicit visual search.** For tasks that do not require localized visual search (e.g., some counting or captioning problems), dense tool rewards can easily encourage unnecessary tool calls and reward hacking. If no tool is needed to answer correctly, the optimal policy is simply to skip tools. Here, we deliberately keep $r_t^{\text{tool}}$ close to zero by default and only apply a negative reward for clear redlines such as invalid coordinates, repeated no-op crops, or avoidable sandbox errors.

# 5. Experiments

We present our training data and implementation details in 5.1 and provide an overview of the selected baselines and evaluation benchmarks in Section 5.2. We conduct additional faithfulness evaluation in Section 5.3 and finally we provide an ablation study in 5.4. Additional qualitative examples will be presented in Appendix 16.

## 5.1. Experimental Setup

**Data preparation.** High-quality training data is crucial for incentivizing visual reasoning and enabling the model to solve complex tasks. We use Thyme-SFT [44], a curated dataset with rich multi-step reasoning demonstrations in code and direct-answer samples. For RL training, we construct a larger dataset from open-source sources such as Thinklite-70K [35] and DeepEyes-47K [47], first removing problems requiring external knowledge (e.g., OK-VQA [19]) and then rigorously cleaning noisy labels via automated checks with Qwen2.5-VL-32B and manual verification. We also prompt Qwen2.5-VL-7B to answer each question eight times and discard samples whose empirical accuracy exceeds 0.9 during sampling, which improves RL efficiency since those rollouts receive the same reward and no relative advantage. This dataset is used exclusively for our TAPO-based RL stage. Details are in Appendix 9.

**Implementation details.** For the SFT stage, we follow the setting in Thyme [44] and finetune for three epochs. As for the RL stage, we design our prompt template with the system prompt explicitly specifying the required format, including the special tokens `<think>`, `<code>`, and `<answer>` , along with the sandbox output token as `<sandbox_output>`. Detailed prompt configurations are provided in Appendix 14. For training hyperparameters, both the rollout batch size and the training batch size are set to 256, with eight rollouts generated per sample and 200 steps of updates. Sampling is conducted at a temperature of 1.0, and optimization is performed with a learning rate of $1 \times 10^{-6}$. Our experiment uses eight H200 GPUs, with 255 and 288 GPU hours in the SFT and RL stages, respectively.

## 5.2. Performance of CodeV

We evaluate CodeV against both open-source baselines (Qwen2.5-VL-7B, Pixel-Reasoner-7B, Thyme-VL-7B-RL) and a proprietary model (GPT-4o) on a suite of comprehensive multimodal perception and reasoning benchmarks.

**Results.** Figure 4 summarizes the results (full numbers in Appendix 10, Tables 3 and 4). On VLMBlinds, CodeV attains the best score (46.7), improving over Qwen2.5-VL-7B and Pixel-Reasoner-7B by roughly 3 points and over Thyme-7B-RL by about 8 points, and also slightly surpassing GPT-4o. This suggests that training with TAPO does not sacrifice performance on primitive but challenging perception tasks. On large-image visual search benchmarks (V*, HRBench-4K-all, HRBench-8K-all), CodeV delivers strong overall performance. It achieves the highest score on V* (84.8), clearly outperforming GPT-4o (64.4) and Qwen2.5-VL-7B (75.0). On HRBench-4K-all and HRBench-8K-all, CodeV matches or slightly exceeds the best open-source baseline while maintaining a sizable margin over GPT-4o, indicating robust fine-grained visual reasoning at high resolution. On math-heavy datasets, CodeV reaches 71.8 on MathVista, the best among all models, and 49.2 on MathVerse-Mini, closing most of the gap to GPT-4o while outperforming other open-source baselines. On MathVision-mini, CodeV substantially improves over Qwen2.5-VL-7B and Pixel-Reasoner-7B and remains competitive with GPT-4o.

Overall, these results show that CodeV narrows the gap between open-source and proprietary systems on both perception and reasoning tasks. The consistent gains over the strong Qwen2.5-VL-7B baseline suggest that incentivizing visual tool use with dense process-level rewards in TAPO provides a principled path to more capable agentic VLMs.

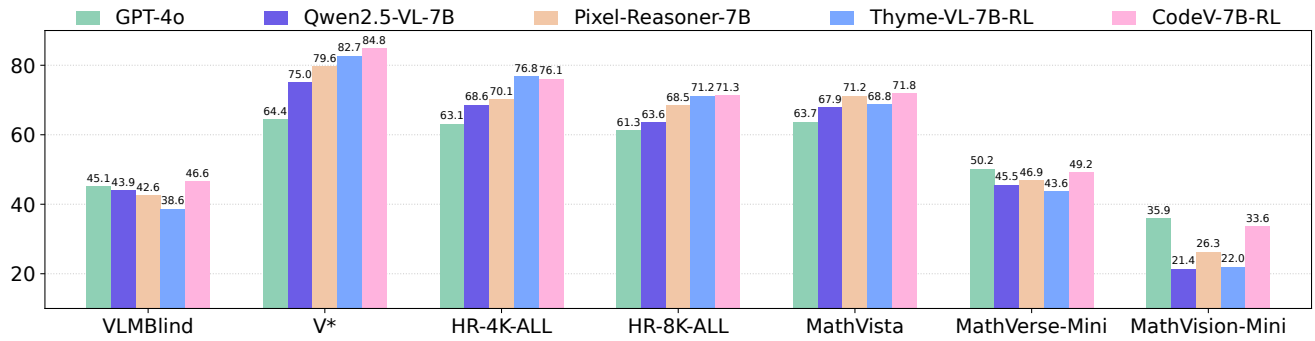



Figure 4. Performance across primitive perception, visual search, reasoning and math benchmarks. All model other than GPT-4o are 7B model with proper setup of tool use.

### 5.3. Evaluating Reasoning Faithfulness

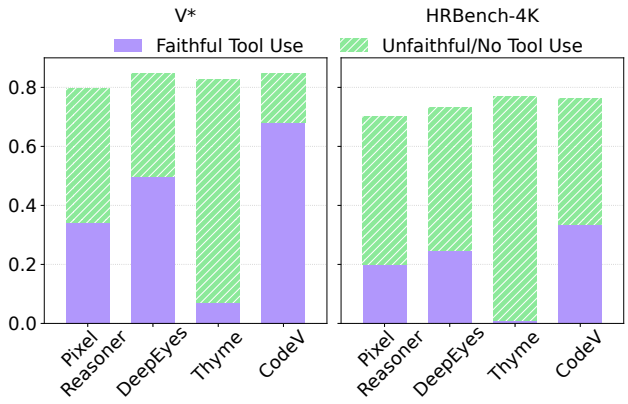



Figure 5. Faithfulness comparison on V* and HRBench-4k benchmarks. The extremely low faithful tool use rate in [44] results from low tool use rate and decorative tool use in chain of thought.

To evaluate the faithfulness enabled by the design of TAPO, we extend the analysis originally presented in Section 3 to compare the faithfulness of baseline approaches against CodeV. Additionally, we add Thyme [44] and the HRBench-4K benchmark. We report the faithful rate alongside the unfaithful and no-tool-use rates. A higher rate of faithful tool-use indicates better alignment between the model's reasoning and the question intent.

**Results.** Figure 5 compares faithful tool-use rates across the V* and HRBench-4k benchmarks. CodeV consistently attains the highest faithfulness, improving tool-use rates over Pixel-Reasoner and DeepEyes by double-digit margins on most settings and by more than 30 points in the most extreme case, while maintaining comparable or better answer accuracy. In relative terms, CodeV's faithful tool-use rate is roughly 1.3–2$\times$ that of other tool-using baselines, indicating that TAPO effectively encourages the model to ground its decisions in the evidence returned by tools.

Thyme, in contrast, exhibits extremely low faithfulness on both datasets (single-digit points). Manual inspection shows that fewer than 10% of its rollouts actually call visual tools: it often relies on purely textual reasoning, implicitly assuming a "cropped image" without executing the crop. These findings underscore the importance of explicit faithfulness evaluation and process-aware RL: without step-level incentives, models tend to optimize final accuracy while shortcutting tool use. By jointly rewarding answer correctness and faithful tool calls, TAPO enables CodeV to sustain high accuracy while substantially improving the reliability of its visual reasoning. Additional tool perturbation study and tool use analysis are in Appendix 11 and 12.

### 5.4. Ablation Studies

To better understand the contributions of our design choices, we conduct ablation studies on two critical aspects: (1) training stage ablation and (2) reward design in TAPO. We report the average scores across all perception and reasoning benchmarks. The full results and training curves are available in Appendix 10.3 and 15.

**Training stage ablation.** Starting from Qwen2.5-VL-7B, Zero-RL (without cold-start SFT stage) improves average performance by roughly 3–4 points but quickly collapses to pure text reasoning with minimal tool use, similar to the learning trap observed in Pixel-Reasoner [26]. The cold-start SFT model slightly underperforms these baselines but produces trajectories with richer tool use. Building on this initialization, CodeV trained with TAPO further improves average scores by about 1–3 points over Zero-RL and 6–8 points over SFT, with the largest gains on perception benchmarks, indicating that code and sandbox feedback are particularly beneficial for visual reasoning.

Table 2 studies reward design starting from the cold-start SFT model. An accuracy-only outcome reward yields a small gain over SFT, but eventually drives the policy toward text-only reasoning (green curve in Figure 6), and adding a consistency reward, similar as [44], only has negligible changes in average performance. Using GPT-5-nano as a judge within TAPO provides an improvement of approximately 2 points in reasoning and 1 point in perception. The full TAPO reward used in CodeV adds another $\sim$2-point

| Training stage | **Reasoning** | **Perception** |
|---|---|---|
| Qwen2.5-VL-7B | 49.7 | 62.8 |
| Zero-RL | 52.9 | 67.0 |
| Cold-start SFT | 47.7 | 61.7 |
| CodeV | 54.2 | 69.7 |

Table 1. **Training stage ablation.** Average performance across all reasoning and perception benchmarks.

| Reward design | **Reasoning** | **Perception** |
|---|---|---|
| Accuracy only | 51.2 | 67.5 |
| + Consistency | 50.4 | 67.6 |
| + GPT-5-nano Judge | 52.4 | 68.7 |
| CodeV | 54.2 | 69.7 |

Table 2. **Reward design ablation.** Average performance across all reasoning and perception benchmarks.

gain, yielding the strongest overall performance and suggesting that step-level supervision on tool use is more effective than purely outcome-based rewards for fostering robust agentic visual reasoning.

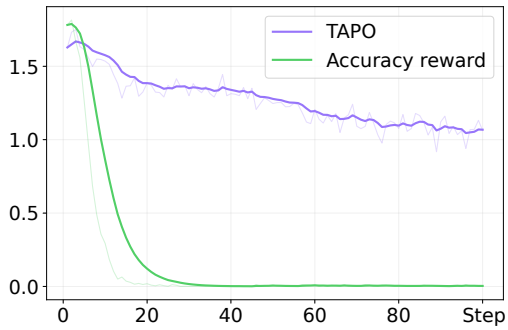



Figure 6. Tool invocation count during early RL training.

# 6. Related Work

## 6.1. Visual Reasoning and Tool Use

A line of work explores agentic visual reasoning by composing image operations or tools, from VisProg [6] and ViperGPT [28], which express complex visual tasks as executable programs over image operators or a unified Python API, to MM-ReAct [40] and Visual Sketchpad [7], which prompt VLMs to interleave perception, search, and calculation while editing intermediate sketches as a visual chain-of-thought. More recent methods, MathCoder-VL [33] trains models on large image–code corpora to jointly perceive images and write executable reasoning code. Pixel-Reasoner [26], DeepEyes [47], ViRFT [16], and OpenThinkIMG [27] apply reinforcement learning improve long-horizon multimodal reasoning, while VISTA [8] and Thyme [44] investigate more flexible tool interfaces and code-based tool abstractions. Our work builds on these tool-based visual agents by exposing their limitations, which motivates a new focus on evaluating and training for faithful visual tool outputs.

## 6.2. Chain-of-Thought Faithfulness

Early RLHF systems such as InstructGPT [22] relied on PPO [23] with sparse outcome rewards, suffering from credit-assignment limitations. Subsequent work showed that process reward models with step-level feedback can substantially outperform pure outcome supervision on reasoning tasks [15], while DeepSeekMath [24] demonstrated that GRPO with outcome-only rewards may still induce unfaithful behaviors such as language mixing. More recent frameworks [29, 39] combine verifiable signals with dense reward-model scores and on-policy distillation of high-reward trajectories to stabilize long-horizon reasoning. Complementary to these textual chain-of-thought methods, we pursue an alternative route that attaches verifiable rewards directly to visual tool outputs, providing step-level supervision that tightly couples the agent's actions to the underlying evidence.

# 7. Conclusion

In this work, we studied how to train agentic multimodal models that perform faithful visual reasoning. We identified unfaithful behaviors in current open-source models. We then introduced CodeV, a code-based visual agent trained with Tool-Aware Policy Optimization (TAPO), a process-level RL framework that directly rewards faithful visual tool use instead of focusing solely on final-answer accuracy. Empirically, CodeV attains strong performance and a high rate of faithful tool use across visual search and reasoning benchmarks. These results reveal a key limitation of outcome-only RLVR and highlight the importance of explicitly supervising tool use when training agentic multimodal systems.

**Limitations and Future Work** While CodeV and TAPO make progress toward faithful visual tool use, several limitations remain. First, TAPO relies on a static judge model, which incurs additional deployment costs; fine-tuning a reward model or self-critic to approximate this judge is an important direction. Second, our current judge is mainly tailored to image-centric operations, such as cropping, and does not provide a general metric for tool-output faithfulness across richer tool types. Extending our framework to broader tool ecosystems (e.g., structured knowledge bases, dynamic web environments) will require designing scalable, verifiable notions of tool faithfulness and corresponding task-specific reward signals.

## References


[1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, et al. Qwen2. 5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 2

[2] Udbhav Bamba, Minghao Fang, Yifan Yu, Haizhong Zheng, and Fan Lai. Xrpo: Pushing the limits of grpo with targeted exploration and exploitation. *arXiv preprint arXiv:2510.06672*, 2025. 1

[3] Tianzhe Chu, Yuexiang Zhai, Jihan Yang, Shengbang Tong, Saining Xie, Dale Schuurmans, Quoc V Le, Sergey Levine, and Yi Ma. Sft memorizes, rl generalizes: A comparative study of foundation model post-training. *arXiv preprint arXiv:2501.17161*, 2025. 3

[4] Jiazhan Feng, Shijue Huang, Xingwei Qu, Ge Zhang, Yujia Qin, Baoquan Zhong, Chengquan Jiang, Jinxin Chi, and Wanjun Zhong. Retool: Reinforcement learning for strategic tool use in llms. *arXiv preprint arXiv:2504.11536*, 2025. 2, 3

[5] Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Ruoyu Zhang, Runxin Xu, Qihao Zhu, Shirong Ma, Peiyi Wang, Xiao Bi, et al. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *arXiv preprint arXiv:2501.12948*, 2025. 1, 2

[6] Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 14953–14962, 2023. 1, 8

[7] Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *Advances in Neural Information Processing Systems*, 37:139348–139379, 2024. 1, 8

[8] Zeyi Huang, Yuyang Ji, Anirudh Sundara Rajan, Zefan Cai, Wen Xiao, Haohan Wang, Junjie Hu, and Yong Jae Lee. Visualtoolagent (vista): A reinforcement learning framework for visual tool selection. *arXiv preprint arXiv:2505.20289*, 2025. 1, 3, 8

[9] Aaron Jaech, Adam Kalai, Adam Lerer, Adam Richardson, Ahmed El-Kishky, Aiden Low, Alec Helyar, Aleksander Madry, Alex Beutel, Alex Carney, et al. Openai o1 system card. *arXiv preprint arXiv:2412.16720*, 2024. 1, 2

[10] Akhil Kondepudi, Akshay Rao, Chenhui Zhao, Yiwei Lyu, Samir Harake, Soumyanil Banerjee, Rushikesh Joshi, Anna-Katharina Meissner, Renly Hou, Cheng Jiang, et al. Health system learning achieves generalist neuroimaging models. *arXiv preprint arXiv:2511.18640*, 2025.

[11] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Yuanchun Zhou, Mingjie Tang, and Jianguo Wang. A demonstration of gptuner: A gpt-based manual-reading database tuning system. In *Companion of the 2024 International Conference on Management of Data*, pages 504–507, 2024.

[12] Jiale Lao, Yibo Wang, Yufei Li, Jianping Wang, Yunjia Zhang, Zhiyuan Cheng, Wanghu Chen, Mingjie Tang, and Jianguo Wang. Gptuner: An llm-based database tuning system. *ACM SIGMOD Record*, 54(1):101–110, 2025. 1

[13] Xiaoxi Li, Guanting Dong, Jiajie Jin, Yuyao Zhang, Yujia Zhou, Yutao Zhu, Peitian Zhang, and Zhicheng Dou. Search-o1: Agentic search-enhanced large reasoning models. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pages 5420–5438, 2025. 2

[14] Xuefeng Li, Haoyang Zou, and Pengfei Liu. Torl: Scaling tool-integrated rl. *arXiv preprint arXiv:2503.23383*, 2025. 3

[15] Hunter Lightman, Vineet Kosaraju, Yuri Burda, Harrison Edwards, Bowen Baker, Teddy Lee, Jan Leike, John Schulman, Ilya Sutskever, and Karl Cobbe. Let's verify step by step. In *The Twelfth International Conference on Learning Representations*, 2023. 2, 8

[16] Ziyu Liu, Yuhang Zang, Yushan Zou, Zijian Liang, Xiaoyi Dong, Yuhang Cao, Haodong Duan, Dahua Lin, and Jiaqi Wang. Visual agentic reinforcement fine-tuning. *arXiv preprint arXiv:2505.14246*, 2025. 2, 3, 8

[17] 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. *arXiv preprint arXiv:2310.02255*, 2023. 1

[18] Yiwei Lyu, Samir Harake, Asadur Chowdury, Soumyanil Banerjee, Rachel Gologorsky, Shixuan Liu, Anna-Katharina Meissner, Akshay Rao, Chenhui Zhao, Akhil Kondepudi, et al. Learning neuroimaging models from health system-scale data. *Nature Biomedical Engineering*, pages 1–13, 2026. 1

[19] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 3195–3204, 2019. 6, 2

[20] Reiichiro Nakano, Jacob Hilton, Suchir Balaji, Jeff Wu, Long Ouyang, Christina Kim, Christopher Hesse, Shantanu Jain, Vineet Kosaraju, William Saunders, et al. Webgpt: Browser-assisted question-answering with human feedback. *arXiv preprint arXiv:2112.09332*, 2021. 2

[21] OpenAI. Thinking with images. *OpenAI Blog*, 2025. https://openai.com/index/thinking-with-images/. 1, 2, 3

[22] Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. Training language models to follow instructions with human feedback. *Advances in neural information processing systems*, 35:27730–27744, 2022. 8

[23] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 8

[24] 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. 2, 8

[25] Joar Skalse, Nikolaus H. R. Howe, Dmitrii Krasheninnikov, and David Krueger. Defining and characterizing reward

hacking. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA, 2022. Curran Associates Inc. 2

[26] Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhu Chen. Pixel reasoner: Incentivizing pixel space reasoning via curiosity-driven reinforcement learning. In *The Thirty-ninth Annual Conference on Neural Information Processing Systems*, 2025. 1, 2, 3, 7, 8

[27] Zhaochen Su, Linjie Li, Mingyang Song, Yunzhuo Hao, Zhengyuan Yang, Jun Zhang, Guanjie Chen, Jiawei Gu, Juntao Li, Xiaoye Qu, et al. Openthinkimg: Learning to think with images via visual tool reinforcement learning. *arXiv preprint arXiv:2505.08617*, 2025. 1, 8

[28] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 11888–11898, 2023. 1, 8

[29] Leitian Tao, Ilia Kulikov, Swarnadeep Saha, Tianlu Wang, Jing Xu, Sharon Li, Jason E Weston, and Ping Yu. Hybrid reinforcement: When reward is sparse, it's better to be dense. *arXiv preprint arXiv:2510.07242*, 2025. 8

[30] Kimi Team, Yifan Bai, Yiping Bao, Guanduo Chen, Jiahao Chen, Ningxin Chen, Ruijue Chen, Yanru Chen, Yuankun Chen, Yutian Chen, et al. Kimi k2: Open agentic intelligence. *arXiv preprint arXiv:2507.20534*, 2025. 1

[31] Kimi Team, Angang Du, Bofei Gao, Bowei Xing, Changjiu Jiang, Cheng Chen, Cheng Li, Chenjun Xiao, Chenzhuang Du, Chonghua Liao, et al. Kimi k1. 5: Scaling reinforcement learning with llms. *arXiv preprint arXiv:2501.12599*, 2025. 1

[32] Simon Valentin, Jinmiao Fu, Gianluca Detommaso, Shaoyuan Xu, Giovanni Zappella, and Bryan Wang. Cost-effective hallucination detection for llms. *arXiv preprint arXiv:2407.21424*, 2024. 2

[33] Ke Wang, Junting Pan, Linda Wei, Aojun Zhou, Weikang Shi, Zimu Lu, Han Xiao, Yunqiao Yang, Houxing Ren, Mingjie Zhan, and Hongsheng Li. MathCoder-VL: Bridging vision and code for enhanced multimodal mathematical reasoning. In *Findings of the Association for Computational Linguistics: ACL 2025*, pages 2505–2534, Vienna, Austria, 2025. Association for Computational Linguistics. 8

[34] Wenbin Wang, Liang Ding, Minyan Zeng, Xiabin Zhou, Li Shen, Yong Luo, Wei Yu, and Dacheng Tao. Divide, conquer and combine: A training-free framework for high-resolution image perception in multimodal large language models. In *Proceedings of the AAAI Conference on Artificial Intelligence*, pages 7907–7915, 2025. 3

[35] Xiyao Wang, Zhengyuan Yang, Chao Feng, Hongjin Lu, Linjie Li, Chung-Ching Lin, Kevin Lin, Furong Huang, and Lijuan Wang. Sota with less: Mcts-guided sample selection for data-efficient visual reasoning self-improvement. *arXiv preprint arXiv:2504.07934*, 2025. 2, 6, 1

[36] Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *Advances in Neural Information Processing Systems*, 37:113569–113697, 2024. 1

[37] Chao Wu, Baoheng Li, Mingchen Gao, and Zhenyi Wang. From efficiency to adaptivity: A deeper look at adaptive reasoning in large language models. *arXiv preprint arXiv:2511.10788*, 2025. 2

[38] Penghao Wu and Saining Xie. V*: Guided visual search as a core mechanism in multimodal llms. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 13084–13094, 2024. 1, 3

[39] Peiran Xu, Zhuohao Li, Xiaoying Xing, Guannan Zhang, Debiao Li, and Kunyu Shi. Hybrid reward normalization for process-supervised non-verifiable agentic tasks. *arXiv preprint arXiv:2509.25598*, 2025. 8

[40] Zhengyuan Yang, Linjie Li, Jianfeng Wang, Kevin Lin, Ehsan Azarnasab, Faisal Ahmed, Zicheng Liu, Ce Liu, Michael Zeng, and Lijuan Wang. Mm-react: Prompting chatgpt for multimodal reasoning and action. *arXiv preprint arXiv:2303.11381*, 2023. 8

[41] Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik R Narasimhan, and Yuan Cao. React: Synergizing reasoning and acting in language models. In *The eleventh international conference on learning representations*, 2022. 1, 2

[42] Yifan Yu, Moyan Li, Shaoyuan Xu, Jinmiao Fu, Xinhai Hou, Fan Lai, and Bryan Wang. Correct: Condensed error recognition via knowledge transfer in multi-agent systems. *arXiv preprint arXiv:2509.24088*, 2025.

[43] Yunjia Zhang, Jordan Henkel, Avrilia Floratou, Joyce Cahoon, Shaleen Deep, and Jignesh M Patel. Reactable: enhancing react for table question answering. *Proceedings of the VLDB Endowment*, 17(8):1981–1994, 2024. 1, 2

[44] Yi-Fan Zhang, Xingyu Lu, Shukang Yin, Chaoyou Fu, Wei Chen, Xiao Hu, Bin Wen, Kaiyu Jiang, Changyi Liu, Tianke Zhang, et al. Thyme: Think beyond images. *arXiv preprint arXiv:2508.11630*, 2025. 2, 3, 5, 6, 7, 8, 1

[45] Chenhui Zhao, Yiwei Lyu, Asadur Chowdury, Edward Harake, Akhil Kondepudi, Akshay Rao, Xinhai Hou, Honglak Lee, and Todd Hollon. Towards scalable language-image pre-training for 3d medical imaging. *arXiv preprint arXiv:2505.21862*, 2025. 1

[46] Shitian Zhao, Haoquan Zhang, Shaoheng Lin, Ming Li, Qilong Wu, Kaipeng Zhang, and Chen Wei. Pyvision: Agentic vision with dynamic tooling. *arXiv preprint arXiv:2507.07998*, 2025. 3

[47] Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. Deepeyes: Incentivizing" thinking with images" via reinforcement learning. *arXiv preprint arXiv:2505.14362*, 2025. 1, 2, 3, 5, 6, 8

[48] Zetong Zhou, Dongping Chen, Zixian Ma, Zhihan Hu, Mingyang Fu, Sinan Wang, Yao Wan, Zhou Zhao, and Ranjay Krishna. Reinforced visual perception with tools. *arXiv preprint arXiv:2509.01656*, 2025. 2

# CodeV: Code with Images for Faithful Visual Reasoning via Tool-Aware Policy Optimization

Supplementary Material

In this supplementary material, we provide (1) details of the faithfulness evaluation protocol and judge prompt (Appendix 8), (2) data preprocessing and distribution statistics (Appendix 9), (3) full benchmark results and additional tool use analysis (Appendix 10–12), (4) sandbox implementation details (Appendix 13, (5) prompt templates for RL data and reward model (Appendix 14), (6) training dynamics during RL (Appendix 15) and (7) additional qualitative examples (Appendix 16).

## 8. Faithfulness Evaluation Protocol

Template 8.1: Faithfulness Judge Prompt

**[System]**
You are evaluating whether an image contains any relevant visual content in the question.
Your task: Determine if the image clearly shows any objects/content mentioned or implied in the question.
Use 1 if: - Any objects/content mentioned in the question is clearly visible
Use 0 if: - None of the relevant objects is visible, clearly visible, or identifiable

**[User]**
Question: { }
Does this image clearly show any objects/content mentioned in the question?

Image. { }

To quantify action-level faithfulness in visual tool use, we evaluate only the *visual consequences* of each tool call rather than the model's internal chain-of-thought. Concretely, for every rollout we collect the intermediate cropped images produced during reasoning and, together with the original question, submit each crop to GPT-4o using the *Faithfulness Judge Prompt* (Template 8.1) to decide whether the crop visibly contains the queried object or region. Since these agents are explicitly trained to solve visual search tasks by cropping a small target from a high-resolution image, the presence or absence of the target in the crop provides a direct proxy for whether the tool call follows the intended problem-solving path. The *Faithfulness Evaluation Algorithm* (Algorithm 1) then aggregates these judgments by marking an example as faithful if *any* of its crops is judged to contain the target and computing the proportion of faithful examples among those with correct final answers. This yields a single faithfulness score that measures how often correct predictions are supported by visually grounded tool use, rather than being obtained through shortcuts such as exploiting textual cues.

## 9. Data preparation

**SFT-data.** We start from Qwen2.5-VL-7B-Instruct, which has strong coding and perception capabilities but is not naturally incentivized to write code, inspect visual evidence, and then answer questions. Direct Zero-RL training from this model leads to highly unstable updates: the policy either hacks the tool reward when it is easy to obtain or avoids code entirely once penalties are introduced. Therefore, we first perform a cold-start SFT stage that explicitly teaches the model which tools are available and how to use them to support answering.

We build our SFT corpus from Thyme-SFT [44], a curated dataset distilled from over 4M raw multimodal examples. This dataset contains rich multi-step traces where the model writes and executes code for image operations (e.g., cropping high-resolution images, rotation, contrast enhancement) and mathematical computation. From this corpus, we select single-round and second-round tool-use examples, resulting in 333K SFT samples.

**RL-data.** High-quality RL data is crucial for incentivizing visual reasoning and ensuring that the model can handle complex tasks. We construct our RL corpus by curating from open-source datasets such as Thinklite-70K [35] and DeepEyes-47K [47]. First, we remove problems that re-

**Algorithm 1** Faithfulness Evaluation Algorithm

**Require:** rollout $\mathcal{R}$, results $\mathcal{A}$
Load answer map $\text{Ans}[i] = (\hat{y}_i, y_i, \text{hit}_i)$ from $\mathcal{A}$
Load entries $\{E_1, \ldots, E_N\}$ from $\mathcal{R}$
**for** each entry $E_i$ **do**
  Extract $q_i$, original image path $I_i$, crop paths $C_i = \{c_{i1}, \ldots, c_{iK_i}\}$
  Infer image index $k$ from $I_i$ and get $\text{hit}_i$ from $\text{Ans}[k]$
  **for** each crop $c_{ij} \in C_i$ **do**
    Query GPT-4o with Template 8.1 and crop $c_{ij}$
    Obtain label $z_{ij} \in \{0, 1\}$ (`contains_object`)
  **end for**
  $\text{any\_crop}_i \leftarrow \mathbb{I}(\exists j : z_{ij} = 1)$
**end for**
$\mathcal{I}_{\text{correct}} \leftarrow \{i : \text{hit}_i = 1\}$
$\text{faithful} \leftarrow \dfrac{|\{i \in \mathcal{I}_{\text{correct}} : \text{any\_crop}_i = 1\}|}{|\mathcal{I}_{\text{correct}}|}$
**return** faithfulness score faithful

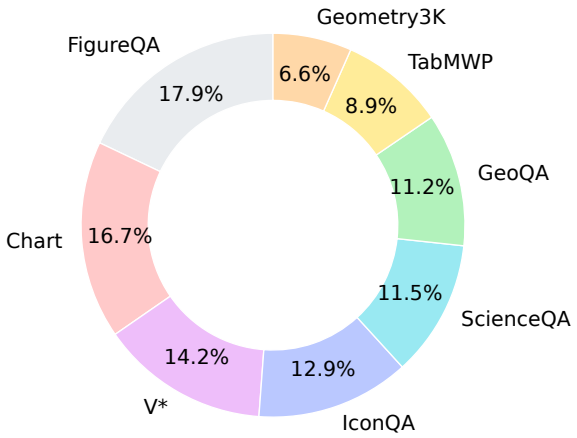



Figure 7. RL data distribution.

quire external knowledge (e.g., OK-VQA [19]), since these tasks depend on web search or domain-specific tools rather than visual reasoning. Second, we clean noisy ground-truth labels via a combination of automatic checks with Qwen2.5-VL-32B-Instruct and manual verification. Third, we prompt Qwen2.5-VL-7B-Instruct to answer each question eight times and discard samples whose empirical accuracy exceeds 0.9, which reduces near-trivial questions where all trajectories receive identical rewards and provide little RL signal.

The resulting RL dataset is used exclusively in our TAPO-based RL stage. The distribution over source datasets is shown in Figure 7, where FigureQA, ChartQA, V*, and IconQA jointly account for more than half of the samples.

## 10. Full results

### 10.1. Full results for main benchmarks

Table 3 provides a detailed breakdown on perception-heavy benchmarks, including VLMBlinds, V*, HRBench (4K/8K), and MME-Realworld-Lite. We observe that models built upon similar large-scale visual tool-use training data, namely *DeepEyes-7B*, *Thyme-RL-7B*, and our *CodeV* variants, exhibit broadly comparable performance across these tasks. This is expected, as these models share overlapping foundations in tool-augmented perception data and visual search supervision.

Within this regime, CodeV demonstrates a consistent advantage:

- On VLMBlinds, CodeV achieves 46.6, outperforming both Thyme-RL-7B (38.6) and DeepEyes-7B (41.2), and slightly surpassing GPT-4o (45.1), indicating that incentivizing tool use does not harm low-level perception robustness.
- On large-image benchmarks (V*, HRBench-4K/8K), CodeV matches or exceeds the best-performing tool-based baselines. In particular, it reaches **84.8** on V* and **91.0** on HRBench-4K-FSP, while maintaining strong performance on HRBench-8K.
- Compared to its backbone baseline Qwen2.5-VL-7B, CodeV-7B-RL yields consistent improvements across all perception dimensions, with gains ranging from +2.7 on VLMBlinds to +9.8 on V*, as summarized by the $\Delta$ row.

These trends suggest that the strong performance of tool-based models on visual search and high-resolution perception tasks is largely grounded in their shared training distributions. Importantly, CodeV further improves upon this foundation through process-level reinforcement learning, leading to more stable and generalizable gains rather than overfitting to specific benchmarks.

**Math and Reasoning Benchmarks.** Table 4 summarizes performance on math-heavy and scientific reasoning benchmarks, including MathVista, CharXiv, MMMU, MathVerse-Mini, and MathVision-Mini.

Compared to proprietary GPT-4o and strong open-source baselines, we observe:

- CodeV achieves the best performance among all models on MathVista (**71.8**) and substantially narrows the gap with GPT-4o on MathVerse-Mini (49.2 vs. 50.2).
- On MathVision-Mini, CodeV scores 33.6, closing most of the gap to GPT-4o (35.9) while significantly outperforming all other 7B open-source models by a large margin (e.g., +12.2 over Qwen2.5-VL-7B).
- On broader multimodal reasoning tasks like MMMU and CharXiv, CodeV remains competitive, consistently surpassing its 7B peers while approaching the performance of GPT-4o.

Taken together, these results show that although our training pipeline is heavily grounded in visual tool use and perception data (similar to Thyme and DeepEyes), the proposed TAPO framework enables a broader transfer of capabilities. Specifically, CodeV not only matches or surpasses peer 7B open-source models across math and reasoning tasks, but also closes much of the remaining gap to GPT-4o on several general-purpose multimodal benchmarks.

### 10.2. Full results on Faithful Tool Use

Table 5 reports the full results of our faithfulness analysis on V*, HRBench-4K, and HRBench-8K. Unlike standard accuracy metrics, this evaluation explicitly measures whether a model's tool use is *evidence-aligned*, i.e., whether its intermediate visual operations (such as image crops) actually contain the queried region or object, and whether the final answer is correct.

**Scaling Difficulty and Resolution.** Another finding is

| Model | VLMBlind | V* | HRBench-4K | | | HRBench-8K | | | MME-Realworld-Lite | | |
|---|---|---|---|---|---|---|---|---|---|---|---|
| | | | ALL | FSP | FCP | ALL | FSP | FCP | All | Reasoning | Perception |
| **Model w/o tool use** | | | | | | | | | | | |
| GPT-4o | 45.1 | 64.4 | 63.1 | 67.8 | 58.5 | 61.3 | 65.3 | 57.3 | 52.0 | 48.3 | 54.4 |
| InternVL3-8B | 44.1 | 76.1 | 70.1 | 79.8 | 61.1 | 69.3 | 78.8 | 59.8 | 48.6 | 44.8 | 51.0 |
| ThinkLite-7B | **47.4** | 76.4 | 70.4 | 87.0 | 53.8 | 65.9 | 81.8 | 50.0 | 39.5 | 39.1 | 39.8 |
| Qwen2.5-VL-7B | 43.9 | 75.0 | 68.6 | 82.3 | 55.0 | 63.6 | 75.0 | 52.3 | 44.1 | 37.7 | 48.8 |
| **Model with tool use** | | | | | | | | | | | |
| DeepEyes-7B | 41.2 | **84.8** | 73.1 | **91.0** | 55.3 | 69.1 | 85.3 | 53.0 | **53.1** | **49.2** | 56.2 |
| Pixel-Reasoner-7B | 42.6 | 79.6 | 70.1 | 83.5 | 56.8 | 68.5 | 82.5 | 54.5 | 49.0 | 45.8 | 51.1 |
| Thyme-RL-7B | 38.6 | 82.7 | **76.8** | 90.5 | **62.0** | 71.2 | **86.2** | 57.1 | 51.9 | 48.9 | 55.9 |
| **CodeV-7B-SFT** | 34.4 | 76.4 | 71.5 | 84.8 | 58.3 | 64.5 | 77.8 | 51.3 | – | – | – |
| **CodeV-7B-RL** | 46.6 | **84.8** | 76.1 | **91.0** | 61.3 | **71.3** | 81.3 | **60.3** | 52.6 | 47.2 | **56.3** |
| Δ Qwen2.5-VL-7B | +2.7 | +9.8 | +7.5 | +8.7 | +6.3 | +7.7 | +6.3 | +8.0 | +8.5 | +9.5 | +7.5 |

Table 3. **Perception benchmarks results.** Comparison of model performance on VLMBlinds, V*, HRBench-4K, HRBench-8K and MME-Realworld-Lite benchmarks (values in %). Best and second-best results in each column are highlighted in **bold** and underlined, respectively.

| Model | MathVista | CharXiv Reasoning | CharXiv Description | MMMU | MathVerse Mini | MathVision Mini |
|---|---|---|---|---|---|---|
| **Model without tool use** | | | | | | |
| GPT-4o | 63.7 | **45.0** | **85.4** | 68.7 | **50.2** | **35.9** |
| InternVL3-8B | 70.4 | 37.8 | 72.2 | **70.1** | 40.1 | 26.3 |
| ThinkLite-7B | 71.3 | 38.4 | 75.9 | 57.8 | 48.2 | 25.7 |
| Qwen2.5-VL-7B | 67.9 | 36.3 | 71.8 | 55.2 | 45.5 | 21.4 |
| **Model with tool use** | | | | | | |
| DeepEyes-7B | 68.0 | 36.7 | 70.5 | 52.7 | 45.4 | 26.3 |
| Pixel-Reasoner-7B | 71.2 | 38.8 | 70.7 | 48.7 | 46.9 | 26.3 |
| Thyme-RL-7B | 68.8 | 38.2 | 67.4 | 54.7 | 43.6 | 22.0 |
| **CodeV-7B-SFT** | 68.1 | 32.3 | 70.6 | 48.4 | 44.2 | 23.7 |
| **CodeV-7B-RL** | **71.8** | 39.3 | 72.1 | 59.3 | 49.2 | 33.6 |
| Δ Qwen2.5-VL-7B | +3.9 | +3.0 | +0.3 | +4.1 | +3.7 | +12.2 |

Table 4. **Math and reasoning benchmarks.** Best and second-best results in each column are highlighted in **bold** and underlined, respectively.

that faithfulness consistently drops as image resolution and task difficulty increase (from V* to HRBench-8K). This highlights the inherent challenge of maintaining faithful reasoning in large-scale, high-resolution visual environments, where models must localize fine-grained regions under greater visual complexity. For tasks like this, we believe a multi-agent system that divides the full image into small parts and analyzes them independently is a more practical solution. Nevertheless, CodeV demonstrates significantly better robustness under this increasing difficulty, further validating the effectiveness of our process-level training design.

### 10.3. Full results for ablation study

The abalation results reported in Section 5.4 present a summarized view averaged across benchmarks. Tables 7 and 8 provide the complete, per-benchmark breakdown.

From these full results, we observe that individual training stages and reward components affect different benchmarks unevenly: some variants yield gains on specific perception subsets (e.g., HRBench-FSP) or structured reasoning tasks (e.g., CharXiv-Reasoning), but often at the cost of regressions elsewhere. In contrast, CodeV shows consistently strong performance across nearly all perception and reasoning benchmarks, indicating that its advantages are not

| Model | V* | | HRBench-4K | | HRBench-8K | |
|---|---|---|---|---|---|---|
| | Accuracy | Faithful | Accuracy | Faithful | Accuracy | Faithful |
| DeepEyes-7B | **84.8** | 49.7 | 73.1 | 24.7 | 69.1 | 6.7 |
| Pixel-Reasoner-7B | 79.6 | 34.1 | 70.1 | 20.1 | 68.5 | 7.6 |
| Thyme-RL-7B | 82.7 | 7.0 | **76.8** | 1.0 | 71.2 | 1.2 |
| **CodeV-7B-RL** | **84.8** | **68.0** | 76.1 | **33.5** | **71.3** | **13.3** |

Table 5. **Faithful results.** Faithful is computed as faithful tool use *AND* correct answer, divided by total number of testing samples. Best result in each column are highlighted in **bold**.

| Model | Δ | Mask | Noise | Random | Empty |
|---|---|---|---|---|---|
| Pixel-Reasoner | Think | 39.7 | 28.7 | 28.3 | 16.8 |
| | Action | 18.3 | 17.3 | 6.3 | 2.6 |
| | Answer | 2.1 | 2.1 | 1.0 | 4.2 |
| DeepEyes | Think | 5.8 | 2.6 | 5.2 | 6.3 |
| | Action | 0.0 | 0.0 | 0.5 | 0.0 |
| | Answer | 5.2 | 5.8 | 5.2 | 4.7 |
| **CodeV-RL** | Think | 85.9 | 90.1 | 82.7 | 79.2 |
| | Action | 48.9 | 50.8 | 44.5 | 24.1 |
| | Answer | 9.6 | 8.0 | 8.9 | 8.4 |

Table 6. **Tool-output perturbation analysis.** We evaluate changes in the model's behavior when all returned tool-output images are perturbed at inference time. **Mask**: mask out the tool output. **Noise**: replace the tool output with Gaussian noise. **Random**: replace the tool output with a non-overlapping random crop. **Empty**: remove the tool output entirely. We report the percentage of examples with a behavioral change (Δ): **Think**, changes in reasoning tokens; **Action**, changes in subsequent tool-use actions (e.g., retrying a crop); and **Answer**, changes in the final answer.

driven by isolated datasets but reflect improved general robustness.

## 11. Perturbation Analysis

Producing a correct crop does not necessarily imply that the model actually uses the tool output. To further assess faithfulness, we perform a *tool-output perturbation* analysis. Specifically, on the V* benchmark, we perturb all returned tool output at inference time and measure whether the model's behavior changes. We consider four perturbation types:

- **Mask**: mask out the tool-output images.
- **Noise**: replace the tool-output images with Gaussian noise.
- **Random**: replace the tool-output images with a non-overlapping random crop.
- **Empty**: remove the tool-output image entirely.

We report the percentage of trajectories that exhibit a behavioral change along three dimensions:

- **Think**: changes in reasoning tokens.
- **Action**: changes in subsequent tool-use actions (e.g., retry cropping).
- **Answer**: changes in final answer.

Higher values indicate that the model is more sensitive to perturbations of the tool output, suggesting that its reasoning, tool-use behavior, or final prediction depends more strongly on the returned visual evidence.

Table 6 shows that Pixel-Reasoner and DeepEyes are largely insensitive to perturbed tool outputs. In contrast, CodeV exhibits substantially larger behavior changes in **Think** and **Action**, indicating that its reasoning process and tool-use decisions are more strongly conditioned on the returned tool outputs. For the final answer, CodeV is also noticeably more sensitive, with an answer-change rate of around 8–10%, compared with roughly 1–5% for the baselines.

A closer examination of CodeV trajectories suggests that, under perturbation, the model often switches to alternative reasoning paths (e.g., text-only reasoning or direct producing an answer instead). As a result, the final naswer frequently remains unchanged even when the internal reasoing and action sequence differ. Overall, these results indicate that CodeV genuinely relies on tool outputs during inference, and this behavior emerges from TAPO's simple process-aware objective, despite not being explicitly trained with perturbation-based supervision.

| Model | VLMBlinds | V* | HRBench-4K | | | HRBench-8K | | |
|---|---|---|---|---|---|---|---|---|
| | | | ALL | FSP | FCP | ALL | FSP | FCP |
| **Training stage** | | | | | | | | |
| Qwen2.5-VL-7B | 43.9 | 75.0 | 68.6 | 82.2 | 55.0 | 63.6 | 75.0 | 52.2 |
| Zero-RL | 46.6 | 78.5 | 73.0 | 89.0 | 57.0 | 69.9 | 84.2 | 55.5 |
| Cold-start SFT | 34.4 | 76.4 | 71.5 | 84.8 | 58.2 | 64.5 | 77.8 | 51.2 |
| **Reward design** | | | | | | | | |
| Accuracy only | 43.8 | 82.7 | 74.9 | 89.8 | 60.0 | 68.8 | 82.8 | 54.8 |
| + Consistency | 45.0 | 83.2 | 74.4 | 87.5 | 61.3 | 67.9 | 79.0 | 56.8 |
| + GPT-5-nano Judge | 44.7 | 84.3 | 74.8 | 89.8 | 59.8 | 71.1 | 85.2 | 57.0 |
| **CodeV-7B-RL** | **46.7** | **84.8** | **76.1** | **91.0** | **61.3** | **71.2** | **81.2** | **60.2** |

Table 7. Ablation study of model performance on Math and reasoning benchmarks (values in %).

| Model | MathVista | CharXiv Reasoning | CharXiv Description | MMMU | MathVerse Mini | MathVision Mini |
|---|---|---|---|---|---|---|
| **Training stage** | | | | | | |
| Qwen2.5-VL-7B | 67.9 | 36.3 | 71.8 | 55.2 | 45.5 | 21.4 |
| Zero-RL | 69.8 | 38.2 | 71.5 | **59.3** | 48.2 | 29.9 |
| Cold-start SFT | 68.1 | 32.3 | 70.6 | 48.4 | 44.2 | 23.7 |
| **Reward design** | | | | | | |
| Accuracy only | 69.0 | 39.0 | 71.4 | 52.8 | 47.9 | 27.3 |
| + Consistency | 69.3 | 36.6 | 70.5 | 52.0 | 48.0 | 26.0 |
| + GPT-5-nano Judge | **72.4** | **39.6** | **72.5** | 56.7 | **49.7** | 23.7 |
| **CodeV-7B-RL** | 71.8 | 39.3 | 72.0 | **59.3** | 49.2 | **33.6** |

Table 8. Ablation study of model performance on VLMBlinds, V*, and HR benchmarks (values in %).

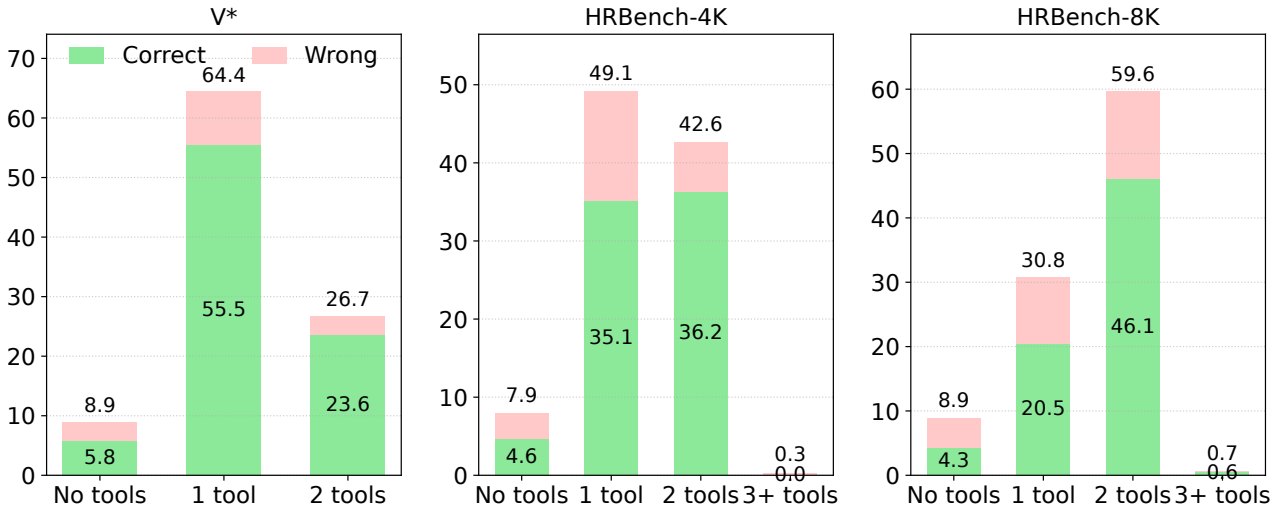



Figure 8. **Tool-use frequency and accuracy of CodeV.** For each dataset (V*, HRBench-4K, HRBench-8K), we bucket examples by the number of tool calls in a trajectory and plot the fraction of all test examples in that bucket, split into correct (green) and incorrect (red) answers. Numbers above each bar show the total fraction of examples in that bucket, and numbers inside the green bars show the fraction of *all* examples that are both correct and use that number of tools.

## 12. Tool Use analysis

Figure 8 analyzes how often CodeV invokes tools and how this correlates with accuracy on V*, HRBench-4K, and HRBench-8K. For each dataset, we bucket episodes by the number of tool calls in a trajectory ("No tools", "1 tool", "2 tools", "3+ tools") and report (i) the fraction of all test examples in each bucket and (ii) the accuracy within that bucket. Concretely, the green portion of each bar corresponds to (% of all examples using this number of tools) $\times$ accuracy in this bucket, so, for example, the "2 tools" bar on HRBench-8K indicates that CodeV answers correctly on 46.1% of all test examples using exactly two tool calls.

Across all three benchmarks, most responses use either one or two tool calls, with 3+ tools being extremely rare ($\leq$1% of examples). On V*, performance is dominated by single-tool trajectories, whereas on HRBench-4K and HRBench-8K the mass shifts toward two-tool trajectories, reflecting the need for slightly deeper visual interaction at higher resolutions. In all cases, zero-tool answers are less accurate than one–two tool trajectories, and additional tool calls beyond two do not yield clear gains, suggesting that CodeV learns to use a small number of focused tool calls rather than over-invoking tools.

## 13. Python Sandbox Design

We execute all model-generated Python code inside a dedicated sandbox with three goals: (i) security and isolation, (ii) reduced coding burden, and (iii) robust error handling.

**Security and isolation** The sandbox isolates execution from the host system and routes all file I/O into a controlled workspace, so that code cannot affect external files or processes. We statically scan generated code for dangerous filesystem operations (e.g., deleting, moving, or renaming files) and block execution if they are detected. Each tool call is further constrained by a strict wall-clock time limit; exceeding this limit raises a timeout error and aborts the run, preventing malformed programs from stalling training.

**Usability and code normalization** To make code execution more forgiving, the sandbox automatically handles many routine details. Before execution, we normalize working directories and relative paths, auto-format and patch minor issues (e.g., indentation or simple I/O omissions), predefine common variables such as the input image path and loaded image, and pre-import frequently used libraries for image I/O and basic vision operations. We use lightweight static analysis to infer cropping boxes and clamp coordinates to valid image boundaries, avoiding crashes from slightly out-of-range indices. Across multiple tool calls in the same trajectory, we maintain imports and variable definitions so that later code segments can reuse earlier results.

**Visualization and artifacts as observations** Instead of requiring the model to manage filenames or GUI windows, the sandbox tracks all newly created artifacts in the workspace as potential observations. We monkey-patch plotting backends (e.g., intercepting calls to `plt.show()`) so that figures are captured as images and returned as part of the observation $o_t$, turning visualizations into explicit evidence for the agent rather than ephemeral side effects.

**Error handling and iterative repair** Each tool call is wrapped with exception handling. If execution fails, the resulting error message is returned to the model, which can then decide whether to revise and regenerate the code or bypass the tool and continue reasoning. This design exposes a large but usable action space: the policy can freely "think with code and images" while the sandbox guarantees safe, consistent, and debuggable execution.

## 14. RL Prompt Design

**Template 14.1: RL Data Prompt**

**[System]**
You are a helpful assistant.

**Goal**:
- Answer the user's question based on the provided image and question.
- You can optionally generate and execute Python code to help analyze or process the image before answering.

**Python execution rules**:
- The Python code will be executed by an external sandbox, and its output will be returned to you in the format <sandbox_output>...</sandbox_output>.
- Use sandbox output to decide whether to answer the question or run another round of Python code.
- For image operations, load the image and then process it (crop, resize, rotate, adjust contrast).
- Save any processed images and print the saved filename.
- For calculation, define all variables and print output.
- Python code must be wrapped in the following block:
<code>
``` python
# your python code here
```
</code>

**[User]**
Image: { } Question: { }
### User Image Path:** "{file_path} "
### User Image Size:** "{image_x}×{image_y}"
### **Output Format (strict adherence required):**
<think>Your detailed reasoning process, including any <code> </code>, should go here.</think>
<answer>Answer to the user's question.</answer>

**Template 14.2: Reward model prompt for perception**

**[System]** You are an expert judge to score a Agent Response's tool use quality.
Agent generate code to process the image, processed image will appear after **Agent Generated Images**
- Score = 1 if **Agent Generated Images** clearly contains at least one object mentioned in the question.
- Score = 0.5 if **Agent Generated Images** shows a PARTIAL match.
- Score = 0.25 if **Agent Generated Images** doesn't matches.
**[User]**
## Question: { }
**Agent Generated Images**: { }

## 15. Training Dynamics

During TAPO reinforcement learning, we observe stable and monotonic improvement in all three critic rewards (Figure 9). The format reward quickly saturates near its max value (0.3), while the accuracy and tool-consistency rewards continue to rise throughout training, indicating that the policy steadily improves both answer correctness and how tools are used. In parallel, the average response length and the mean number of tool calls per step gradually decrease (Figure 10), even as rewards increase, suggesting that TAPO discourages reward hacking via overly long chains or gratuitous tool invocations and instead guides the model toward shorter, more targeted, and faithful visual reasoning with tool use.

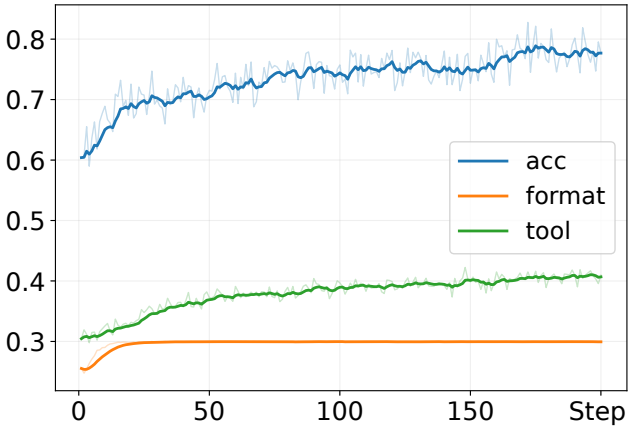



Figure 9. Reward score for accuracy, format and tool use.

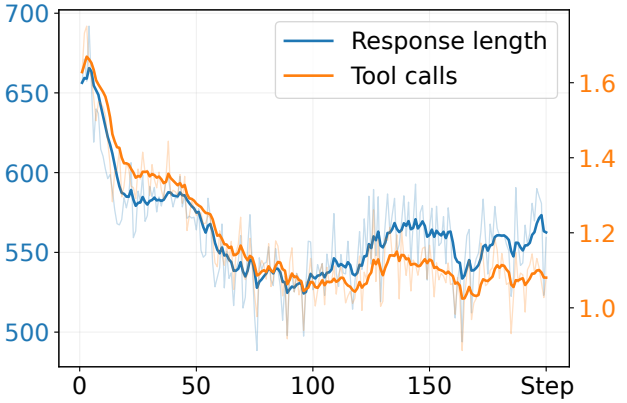



Figure 10. Response length and tool calls during TAPO.

## 16. Qualitative Examples

We provide two faithful and correct examples and two unfaithful examples. These results shows the exact rollouts interleaving text, image, code and tool response.

### 16.1. Faithful Example 1

**User Prompt**

**Question:** What is the color of the slippers?
**Options:** A. black B. red C. blue D. pink
Please select the correct answer from the options above.
**User Image Path:** `"12.jpg"`
**User Image Size:** `"2251x1500"`
**Output Format (strict adherence required):**
`<think>` Your detailed reasoning process, including any `<code>`, should go here. `</think>`
`<answer>` Your final answer to the user's question goes here. `</answer>`

**Assistant**

**<think>** The task involves identifying the color of the slippers on the boat. The slippers are located near the center of the boat, and the image is quite wide, making the slippers relatively small. To accurately determine their color, it is necessary to zoom in on the specific region where the slippers are located. This will enhance the visibility of the slippers and allow for a more precise identification of their color.
Here is the Python code to crop and zoom in on the slippers:
**<code>**

```
import cv2
import numpy as np
import os
import random

# Load the image
image_path = "12.jpg"
image = cv2.imread(image_path)

# Define the coordinates for cropping the
    region around the slippers
x1, y1, x2, y2 = 1100, 850, 1300, 950

# Crop the image
cropped_image = image[y1:y2, x1:x2]

# Apply a zoom factor to enhance the
    visibility of the slippers
zoom_factor = 2
resized_image = cv2.resize(cropped_image, None
    , fx=zoom_factor, fy=zoom_factor,
    interpolation=cv2.INTER_LINEAR)

# Save the processed image with a simple
    filename
processed_path = "processed_1.jpg"
cv2.imwrite(processed_path, resized_image)

# Print the path of the saved cropped image
print(processed_path)
```

**</code> </think>**

**Sandbox Output**

**Assistant**

**<think>**Upon examining the processed image, the slippers are clearly visible near the center of the boat. The enhanced visibility confirms that the slippers are blue in color. This detailed view allows for a precise identification of their color, addressing the original user question effectively. **</think>**
**<answer>** C. blue **</answer>**

## 16.2. Faithful Example 2

**User Prompt**

**Question:** What is the color of the candles?
**Options:** A. red B. yellow C. gray D. white
Please select the correct answer from the options above.
**User Image Path:** "6.jpg"
**User Image Size:** "1998x1500"
**Output Format (strict adherence required):**
<think> Your detailed reasoning process, including any <code>, should go here. </think>
<answer> Your final answer to the user's question goes here. </answer>

**Assistant**

**<think>** The task is to determine the color of the candles in the image. The candles are located on the table in the background, and they are relatively small compared to the overall scene. To accurately identify their color, it is necessary to zoom in on the area around the candles. This will help in clearly distinguishing the color of the candles from the other objects in the image. The coordinates for cropping should focus on the table area where the candles are placed.
Here is the Python code to crop and zoom in on the relevant area:
**<code>**

```
import cv2
import os
import numpy as np
from uuid import uuid4

# Load the image
image_path = "6.jpg"
image = cv2.imread(image_path)

# Define the coordinates for cropping (x1, y1,
     x2, y2)
x1, y1, x2, y2 = 950, 150, 1200, 400

# Crop the image\ncropped_image = image[y1:y2,
     x1:x2]

# Apply zoom (resize the cropped image to a
    larger size)
zoom_factor = 2
zoomed_image = cv2.resize(cropped_image, None,
     fx=zoom_factor, fy=zoom_factor,
    interpolation=cv2.INTER_LINEAR)

# Save the processed image with a simple
    filename\nprocessed_path = \"processed_1.
    jpg\"
cv2.imwrite(processed_path, zoomed_image)

# Print the path of the saved cropped image
print(processed_path)
```

**</code> </think>**

**Sandbox Output**

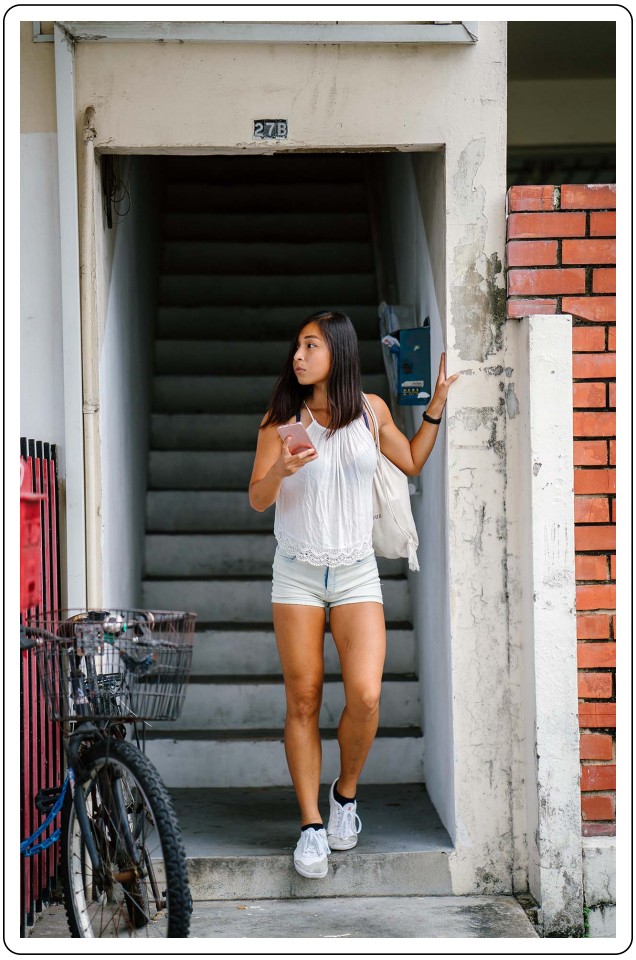
**Assistant**

**<think>**Upon examining the cropped and zoomed image, the candles on the table in the background are clearly visible. The candles appear to be yellow in color. This observation aligns with the options provided in the question. The zoomed-in view effectively highlights the candles, making it easier to distinguish their color from the surrounding objects. Therefore, the correct answer is that the candles are yellow. **</think>**
**<answer>** B. yellow **</answer>**

### 16.3. Unfaithful Example 1

**User Prompt**

**Question:** Is the mailbox on the left or right side of the woman?
**Options:** A. In front B. Left C. Right D. Directly above
Please select the correct answer from the options above.
**User Image Path:** `"10.jpg"`
**User Image Size:** `"4992x7680"`
**Output Format (strict adherence required):**
`<think>` Your detailed reasoning process, including any `<code>`, should go here. `</think>`
`<answer>` Your final answer to the user's question goes here. `</answer>`



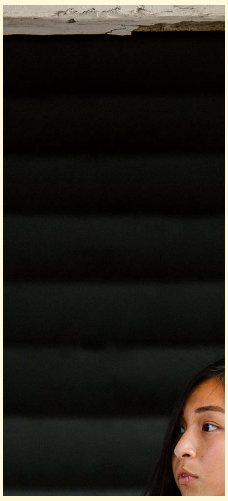

**Assistant**

**<think>** To determine the position of the mailbox relative to the woman, we need to focus on the area around her and the mailbox. The mailbox is located near the wall to her right. Cropping and zooming in on this specific region will help clarify its position relative to the woman. This will ensure that we can accurately identify whether it is on her left, right, or in front of her.
The task involves analyzing the relative position of the mailbox concerning the woman. To improve visibility, I will crop the region around the woman and the mailbox. The coordinates appear to be near the right side of the woman, close to the wall. I will refine this area for better focus and clarity.
Here is the Python code to crop and zoom in on the relevant area:
**<code>**

```
import os
import random
from PIL import Image

# Load the image
image_path = "10.jpg"
image = Image.open(image_path)

# Define the cropping coordinates (x1, y1, x2,
     y2)
x1, y1, x2, y2 = 1600, 1000, 2500, 3000

# Crop the image
cropped_image = image.crop((x1, y1, x2, y2))

# Save the cropped image with a simple
    filename
processed_path = "cropped_1.jpg"
cropped_image.save(processed_path)

# Print the path of the saved cropped image
print(processed_path)
```

**</code> </think>**

**Sandbox Output**



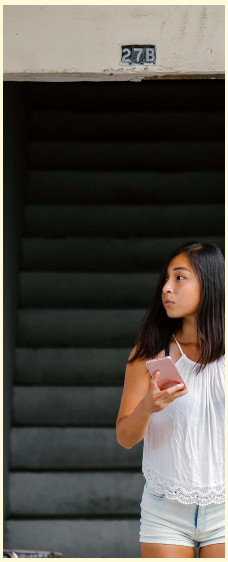
**Assistant**

**<think>** The initial cropping did not effectively capture the mailbox and its position relative to the woman. The cropped image only shows a close-up of the woman's face and the wall, missing the mailbox entirely. To address this, the cropping range needs to be adjusted to include both the woman and the mailbox.
Here is the corrected cropping code to ensure the mailbox and its position relative to the woman are visible:
**<code>**

```
import os
import random
from PIL import Image

# Load the image
image_path = "10.jpg"
image = Image.open(image_path)

# Define the cropping coordinates (x1, y1, x2,
     y2)
x1, y1, x2, y2 = 1000, 500, 2800, 5000

# Crop the image
cropped_image = image.crop((x1, y1, x2, y2))

# Save the cropped image with a simple
    filename
processed_path = "cropped_1.jpg"
cropped_image.save(processed_path)

# Print the path of the saved cropped image
print(processed_path)
```

**</code> </think>**

**Sandbox Output**




**Assistant**

**<think>**The second attempt at cropping successfully includes both the woman and the mailbox. The mailbox is clearly visible on the right side of the woman, confirming its position. Therefore, the mailbox is on the right side of the woman. **</think>**
**<answer>** C. Right**</answer>**

### 16.4. Unfaithful Example 2

**User Prompt**

**Question:** Where is the person standing in relation to the recycle bin?
**Options:**
A. In front of the recycle bin
B. To the right of the recycle bin
C. To the left of the recycle bin
D. Behind the recycle bin
Please select the correct answer from the options above.
**User Image Path:** `"48.jpg"`
**User Image Size:** `"7680x4992"`
**Output Format (strict adherence required):**
`<think>` Your detailed reasoning process, including any `<code>`, should go here. `</think>`
`<answer>` Your final answer to the user's question goes here. `</answer>`

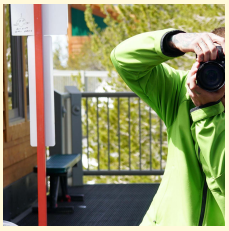
**Assistant**

**<think>** The task involves determining the relative position of the person in relation to the recycle bin. The person is clearly visible in the foreground, and the recycle bin is in the background to the right. To better analyze the spatial relationship, cropping the image to focus on the area around the person and the recycle bin will enhance clarity. This will help in accurately assessing whether the person is in front of, to the right of, to the left of, or behind the recycle bin.
The coordinates for cropping should be around the person and the recycle bin. Based on the image, the person is positioned roughly in the center-right, and the recycle bin is further to the right in the background. Cropping the image to focus on this area will provide a clearer view.
Here is the Python code to crop and zoom in on the relevant area: **<code>**

```
import os
import random
from PIL import Image

# Load the image
image_path = "48.jpg"
image = Image.open(image_path)

# Define the cropping coordinates (x1, y1, x2,
     y2)
x1, y1, x2, y2 = 1500, 500, 4000, 3000

# Crop the image
cropped_image = image.crop((x1, y1, x2, y2))

# Save the cropped image with a simple
    filename
processed_path = "cropped_1.jpg"
cropped_image.save(processed_path)

# Print the path of the saved cropped image
print(processed_path)
```

**</code> </think>**

**Sandbox Output**



**Assistant**

**<think>** The initial cropping did not effectively capture the relative position of the person in relation to the recycle bin. The cropped image focused too closely on the person, excluding the recycle bin and the surrounding area. To accurately determine the person's position relative to the recycle bin, the cropping range should be expanded to include both the person and the recycle bin.
The corrected cropping code should adjust the coordinates to ensure both the person and the recycle bin are visible. The person is still in the foreground, but the recycle bin should be included in the background

to the right.
Here is the corrected cropping code:
**<code>**

```
import os
import random
from PIL import Image

# Load the image
image_path = "48.jpg"
image = Image.open(image_path)

# Define the corrected cropping coordinates (
    x1, y1, x2, y2)
x1, y1, x2, y2 = 0, 0, 7000, 4000

# Crop the image
cropped_image = image.crop((x1, y1, x2, y2))

# Save the cropped image with a simple
    filename
processed_path = "cropped_1.jpg"
cropped_image.save(processed_path)

# Print the path of the saved cropped image
print(processed_path)
```

**</code> </think>**

**Sandbox Output**

**Assistant**

**<think>**The final cropped image now clearly shows both the person and the recycle bin. The person is standing in front of the recycle bin, which is visible in the background to their right. The person is positioned closer to the camera, while the recycle bin is further back and to the right. This confirms that the person is in front of the recycle bin. **</think>**
**<answer>**A. In front of the recycle bin**</answer>**