

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 [51], 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 [42] and DeepEyes-47K [54]. First, we remove problems that re-

Algorithm 1 Faithfulness Evaluation Algorithm

Require: rollout \mathcal{R} , results \mathcal{A}

- 1: Load answer map $\text{Ans}[i] = (\hat{y}_i, y_i, \text{hit}_i)$ from \mathcal{A}
 - 2: Load entries $\{E_1, \dots, E_N\}$ from \mathcal{R}
 - 3: **for** each entry E_i **do**
 - 4: Extract q_i , original image path I_i , crop paths $C_i = \{c_{i1}, \dots, c_{iK_i}\}$
 - 5: Infer image index k from I_i and get hit_i from $\text{Ans}[k]$
 - 6: **for** each crop $c_{ij} \in C_i$ **do**
 - 7: Query GPT-4o with Template 8.1 and crop c_{ij}
 - 8: Obtain label $z_{ij} \in \{0, 1\}$ (contains_object)
 - 9: **end for**
 - 10: $\text{any_crop}_i \leftarrow \mathbb{I}(\exists j : z_{ij} = 1)$
 - 11: **end for**
 - 12: $\mathcal{I}_{\text{correct}} \leftarrow \{i : \text{hit}_i = 1\}$
 - 13: $\text{faithful} \leftarrow \frac{|\{i \in \mathcal{I}_{\text{correct}} : \text{any_crop}_i = 1\}|}{|\mathcal{I}_{\text{correct}}|}$
 - 14: **return** faithfulness score faithful
-

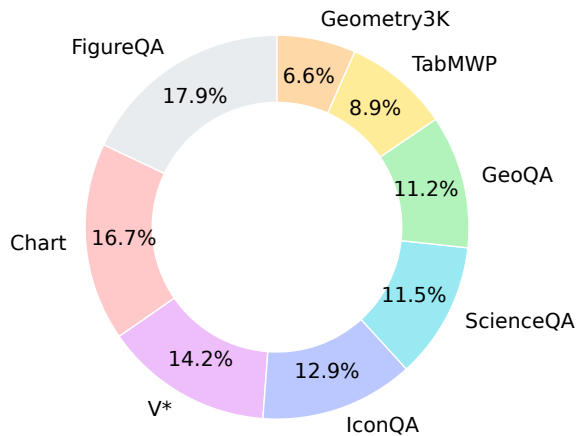


Figure 7. RL data distribution.

quire external knowledge (e.g., OK-VQA [26]), 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 Δ 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	<u>57.3</u>	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	<u>85.3</u>	53.0	53.1	49.2	<u>56.2</u>
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	<u>82.7</u>	76.8	<u>90.5</u>	62.0	<u>71.2</u>	86.2	57.1	51.9	<u>48.9</u>	55.9
CodeV-7B-SFT	34.4	76.4	71.5	84.8	58.3	64.5	77.8	51.3	–	–	–
CodeV-7B-RL	<u>46.6</u>	84.8	<u>76.1</u>	91.0	<u>61.3</u>	71.3	81.3	60.3	<u>52.6</u>	47.2	56.3
△ 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	CharXiv	MMMU	MathVerse	MathVision
		Reasoning	Description		Mini	Mini
Model without tool use						
GPT-4o	63.7	45.0	85.4	<u>68.7</u>	50.2	35.9
InternVL3-8B	70.4	37.8	72.2	70.1	40.1	26.3
ThinkLite-7B	<u>71.3</u>	38.4	<u>75.9</u>	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	<u>39.3</u>	72.1	59.3	<u>49.2</u>	<u>33.6</u>
△ 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 ablation 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 answer frequently remains unchanged even when the internal reasoning 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	CharXiv	MMMU	MathVerse	MathVision
		Reasoning	Description		Mini	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 %).

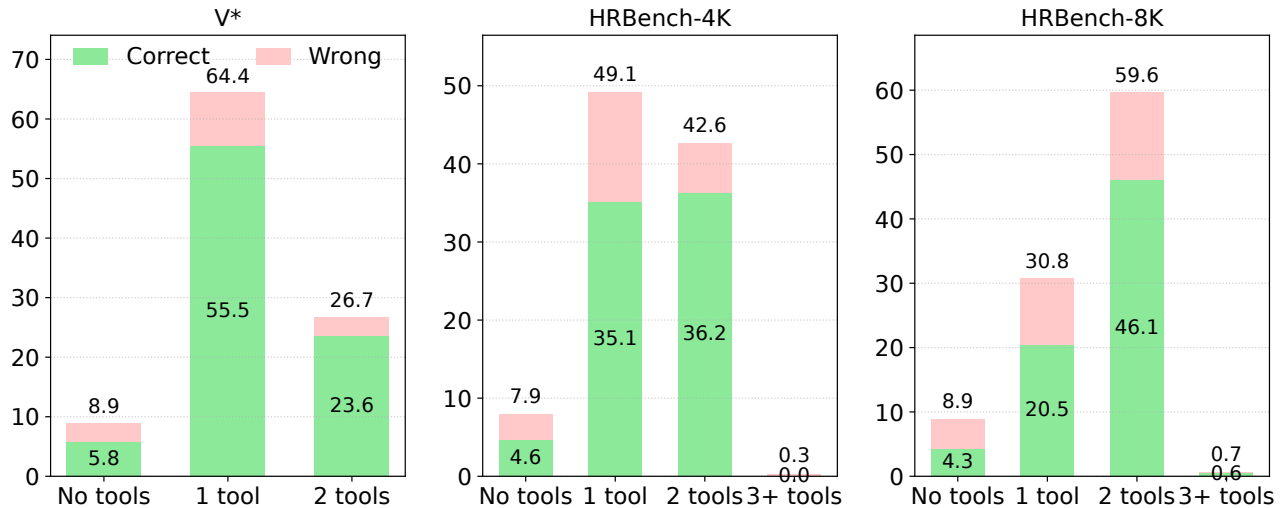


Figure 8. **Tool-use frequency and accuracy of CodeV.** For each dataset (V*, HRBench-4K, HRBench-8K), we bucket examples by the number of tool calls in a trajectory and plot the fraction of all test examples in that bucket, split into correct (green) and incorrect (red) answers. Numbers above each bar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

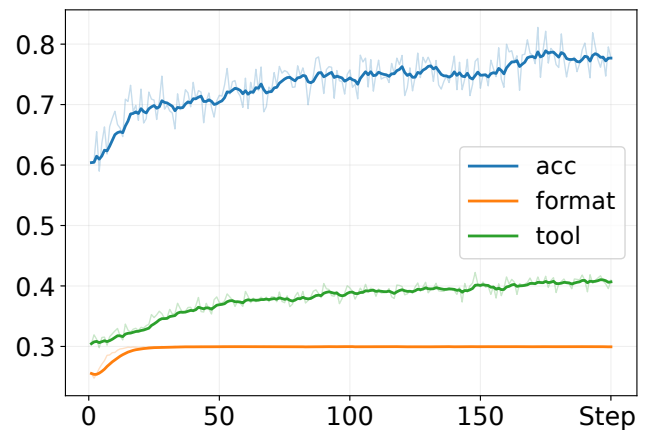


Figure 9. Reward score for accuracy, format and tool use.

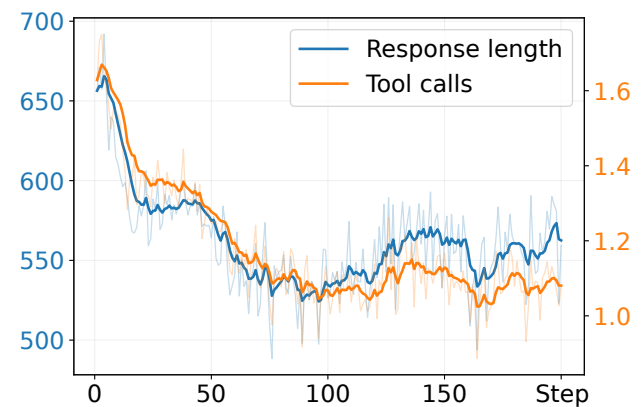


Figure 10. Response length and tool calls during TAPO.

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]

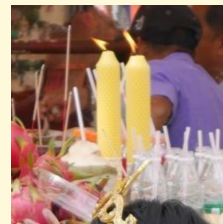
# Apply zoom (resize the cropped image to a
    larger size)
zoom_factor = 2
zoomed_image = cv2.resize(cropped_image, None,
    fx=zoom_factor, fy=zoom_factor,
    interpolation=cv2.INTER_LINEAR)

# 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.

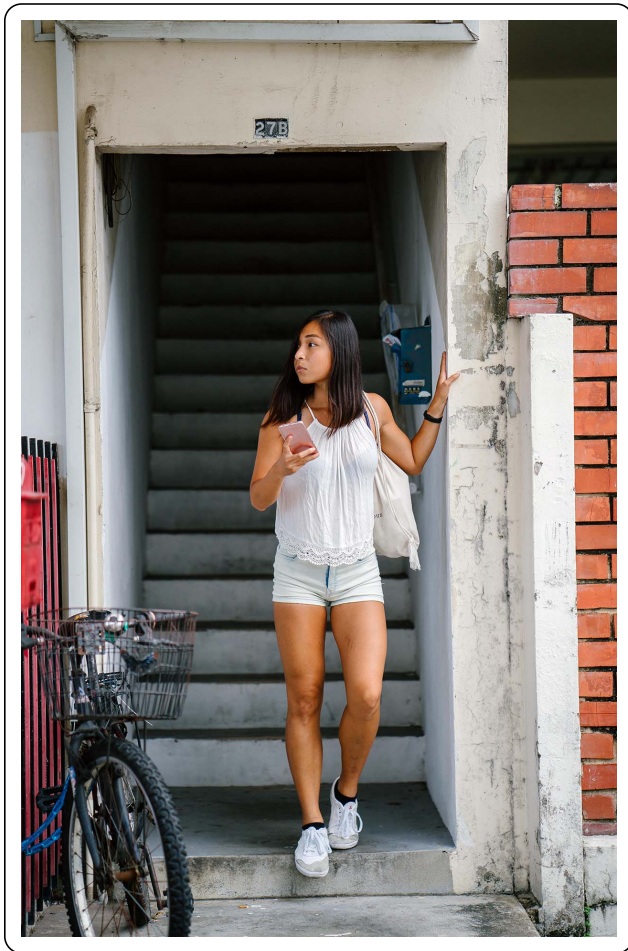
User Image Path: "10.jpg"

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

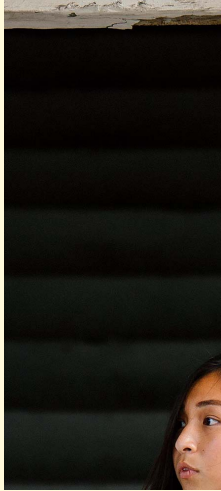
# Crop the image
cropped_image = image.crop((x1, y1, x2, y2))

# 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)
```

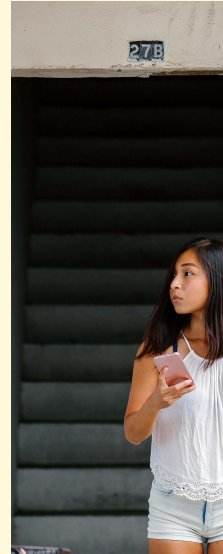
</code> </think>

Sandbox Output



```
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
```

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>**