

CodeDance: A Dynamic Tool-integrated MLLM for Executable Visual Reasoning

Supplementary Material

Appendix

This appendix serves to enhance the reproducibility and transparency of our work by supplementing the main paper with critical technical details and experimental analyses. We provide additional implementation details (e.g., pipeline and examples of the trajectory construction, training setups, and hyperparameters), further descriptions and comparisons of our curated datasets and prompt templates, additional qualitative and quantitative results on various benchmarks, and extended discussions on design choices, broader impacts, limitations and future works.

Contents

1. Additional implementation details	1
1.1. Hyperparameters in training and inference . . .	1
1.2. Implementation of Code sandbox	1
1.3. Implementation of standard GRPO	2
2. Data engineering	2
2.1. Dataset curation	2
2.2. Prompt templates	3
3. Additional empirical studies	6
3.1. Visualization of sequence/turn-level rewards	6
3.2. Hyperparameters in our method	7
3.2.1 . Sensitivity of R_{BAT} Hyperparameters	7
3.2.2 . Effective range and bounds of $d(\mu_{acc})$	8
3.2.3 . Hyperparameter over (γ, δ)	8
3.3. Analysis of Reasoning Depth	9
4. Additional qualitative results	9
4.1. Failure cases observed in our experiments . . .	9
4.2. Additional novel reasoning trajectories	9
5. Discussion	11
6. Broader impact	13
7. Limitations and future work	13

1. Additional implementation details

1.1. Hyperparameters in training and inference

In Table 1, we present the additional hyperparameters used for training our model on the multimodal reasoning tasks.

We primarily adhere to the same settings as Qwen2.5-VL [1], and these parameters are mostly applied across other tasks.

Table 1. Hyperparameters used in training/inference.

	Param. Name	Value / Type
SFT	Batch size	128
	Learning rate	5e-5
	Warmup ratio	0.05
RL	Numerical precision	BF16
	Global batch size	256
	Rollout	8
	Total epochs	1
	Time	~2 Days
Inference & Eval	Deployment platform	vLLM [10]

1.2. Implementation of Code sandbox

To safely execute model-generated programs during training and evaluation, we run all Python code in an internal sandboxed environment. Each execution instance is assigned an isolated working directory and a dedicated namespace for variables, functions, and imports; for multimodal tasks, input images are stored in the corresponding temporary directory and are only visible within that instance. This design avoids interference across concurrent rollouts and confines state to a single trajectory.

Before running user code, the sandbox disables a small set of APIs that may compromise the stability (e.g., blocking user input, direct process termination), and treats each execution as an atomic transaction: if a run fails or is interrupted, partial updates are discarded and the logical state of the instance reverts to the last successful step. We further enforce a strict per-call wall-clock time limit (15s by default); when the limit is exceeded, the current execution is aborted and a timeout status is returned. Finally, the sandbox centralizes observability by capturing structured stdout/stderr logs and graphical outputs produced by common plotting libraries (e.g., `matplotlib`, `PIL`), which are then fed back to the model as textual and visual observations. An illustration of interaction with the code sandbox is shown in Figure 1.

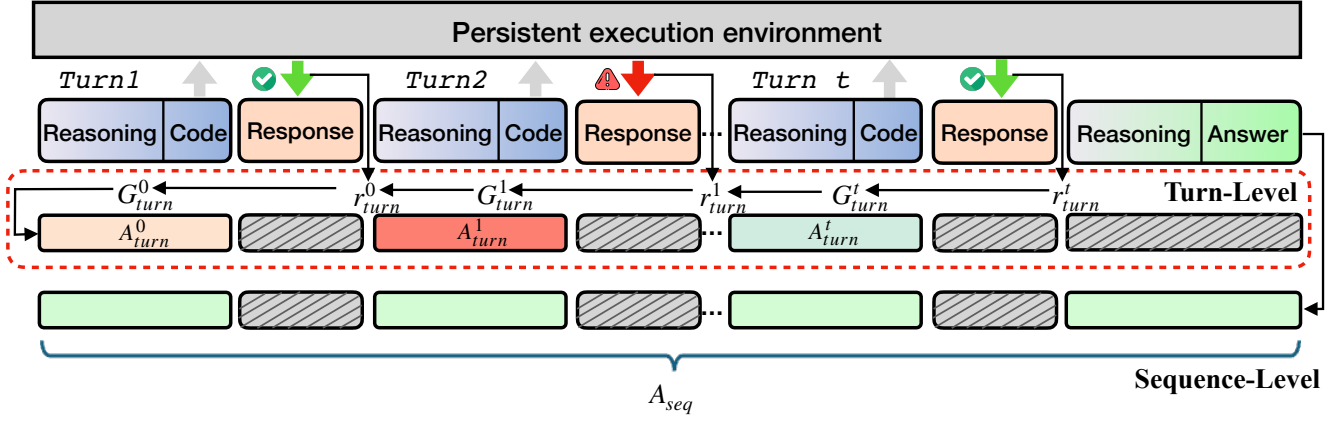


Figure 1. Interaction of our reward design R_{BAT} with the code sandbox (i.e., the execution environment).

1.3. Implementation of standard GRPO

Here, we reveal additional implementation details regarding the RL algorithm used in our work. Group Relative Policy Optimization (GRPO, Shao et al. [17]) has demonstrated strong effectiveness across diverse tasks, particularly in multi-turn tool call agents and “thinking with images” system [6, 7, 18, 23]. Unlike PPO [16], GRPO removes the need for a separate value network by directly computing advantages from the normalized rewards of G sampled solutions. Formally, let $\pi_{\theta_{\text{old}}}$ and π_{θ} denote the policy model (parameterized by θ) before and after the update, respectively, both defined over the action/token space at each position. For a question q sampled from a task dataset \mathcal{Q} , a group of G candidate solutions $\tau_i \sim \pi_{\theta_{\text{old}}}$ are rolled out and evaluated with a reward function $r(\cdot)$. Building on the clipped surrogate objective of PPO, we write the objective \mathcal{J} in an empirical expectation form:

$$\mathcal{J}_{\text{GRPO}}(\theta) = \mathbb{E}_{q \sim \mathcal{Q}, \{\tau_i\}_{i=1}^G \sim \pi_{\theta_{\text{old}}}(\cdot|q)} \left[\frac{1}{G} \sum_{i=1}^G \frac{1}{|\tau_i|} \sum_{t=1}^{|\tau_i|} \min \left(r_{i,t} A_i, \text{clip}(r_{i,t}, 1 - \varepsilon, 1 + \varepsilon) A_i \right) \right]. \quad (1)$$

where $r_{i,t} = \frac{\pi_{\theta}(\tau_{i,t}|q, \tau_{i,<t})}{\pi_{\theta_{\text{old}}}(\tau_{i,t}|q, \tau_{i,<t})}$, $\varepsilon = 0.2$ by default, and $\text{clip}(\cdot)$ denotes the clipping operator for stability. We omit the KL penalty here for simplicity. The normalized within-group reward then defines the advantage A_i of solution τ_i :

$$A_i = \frac{r(\tau_i) - \text{mean}(\{r(\tau_j)\}_{j=1}^G)}{\text{std}(\{r(\tau_j)\}_{j=1}^G)}. \quad (2)$$

We mostly followed the original implementation of GRPO [17] to compute outcome-driven advantage A_{seq} .

2. Data engineering

2.1. Dataset curation

Here, we firstly discuss several related works on data synthesis for MLLM training to complete the related works. Then, we show details of our dataset curation pipeline.

Background: Synthetic reasoning data for MLLM post-training. High-performance MLLMs require substantial instruction-following training data with detailed reasoning trajectories. Recent approaches include converting existing datasets using fixed templates [4, 20] or distilling knowledge from strong teacher models [2, 19, 22], with a focus on developing specific capabilities such as visual-centric reasoning [11] and mathematical problem-solving assisted by visual cues [3, 8]. However, several limitations persist in existing approaches: (i) tool-grounded verification mechanisms are often absent, and (ii) visual operations are typically limited to fixed schema such as cropping or zooming in [18, 23]. In contrast, we synthesize and curate training data with comprehensive reasoning trajectories and tool/code-assisted responses across a wide range of atomic visual operations, employing enhanced process supervision including multi-judge filtering and consistency validation. This leads to “thinking with images” reasoning capability [15] with competitive performance while requiring substantially less training data.

Synthesize high-quality cold-start trajectories for tool-integrated reasoning (TIR). An overview of the curation pipeline as a supplement of the main paper is shown in Figure 2 and a detailed example is in Figure 3. In addition, the dataset used for our trajectory synthesis is primarily composed of the following datasets:

- **Mathematical Reasoning:** MMK12 [14], Retool [6].
- **Table Data:** ChartQAPro [13], chartgemma [12].
- **Natural Images:** SA1B [9].

- **General Data:** Mulberry [21].

For the RL training, our data mainly comes from Deep-eyes [23], SA1B [9] and PixmoCount train [5].

2.2. Prompt templates

Prompt templates used in SFT data synthesis. To ensure the reliability and consistency of synthesized data, we design a set of standardized prompt templates tailored for different stages of the vision-language data pipeline. These templates serve complementary purposes: (i) **In Table 2:** assessing the informativeness of candidate images to guarantee sufficient visual complexity for fine-grained reasoning; (ii) **In Table 3:** labeling and locating the objects that most match the question. (iii) **In Table 4:** validating the quality of automatically generated visual question-answer pairs; (iv) **In Table 5:** enforcing a structured step-by-step reasoning process with explicit final answers; and (v) **In Table 6:** enhancing reasoning accuracy by incorporating code interpreter support for precise numerical or logical calculations. Together, these prompt templates provide a comprehensive and systematic framework for controlling data quality during synthesis, thereby improving the robustness and utility of the resulting multimodal datasets.

Table 2. Prompt template for assessing image informativeness.

You are an expert vision-language analyst.

Task

1. Observe the entire image.
2. Decide whether the picture meets **all Four conditions** below:
 - A. Diversity – Contains ≥ 4 different object categories **or** ≥ 6 individual objects.
 - B. Distinguishability – Includes at least one object that is mostly un-occluded, covers $\geq 30\%$ of the image area, and is not repeated by many visually identical copies.
 - C. Zoom-in Benefit – For that object (or another), some informative fine-grained detail (e.g., printed text, small logo, numerical value, subtle texture, or facial expression) would become noticeably clearer if the region were enlarged. In other words, a close-up view would materially help a downstream model answer a question about that object.
 - D. Is it suitable to come up with some VQA questions that require fine-grained understanding?
3. If **all** A, B, C, D are satisfied, Please respond with “True” or “False”.

Prompt templates used in RL training. We provide the RL training prompt template in **Table 7**. This template illustrates the input–output format and executable code constraints used during RL rollouts, offering additional transparency and reproducibility of our training setup.

Table 3. Prompt template for bbox generation.

Please detect the entire object that most matches the question in the image.

Question: {question}

If the target is part of an object, you need to give the bbox of the entire object.

For each object, return:

- 'label': the object name
- 'bbox_2d': the object’s bounding box coordinates as [x1, y1, x2, y2].

Respond in a **JSON array**, where each entry is a dictionary with 'label' and 'bbox_2d'.

Table 4. Prompt template for visual question validation.

You are a quality control assistant. Your task is to evaluate a visual question based on the provided image, question, and correct answer.

Image: [Image is attached]

Question: {question}

Provided Correct Answer: {correct_answer}

Evaluation Criteria:

1. **Correctness:** Is the provided “Correct Answer” truly the correct answer based on the image?
2. **Difficulty:** Is the question non-trivial? It should require careful observation of details and not be something overly simple or obvious (e.g., “What color is the sky?”).

Your Response:

Respond with “GOOD” if the question meets BOTH criteria.

Respond with “BAD” if the question fails one or both criteria. Do not provide any other explanation or text.

Table 5. Prompt template for step-by-step solving with answer tag.

Solve the following problem step by step and then provide the final answer.

The final answer **MUST BE** enclosed within <answer> </answer> tags.

Question: {question}

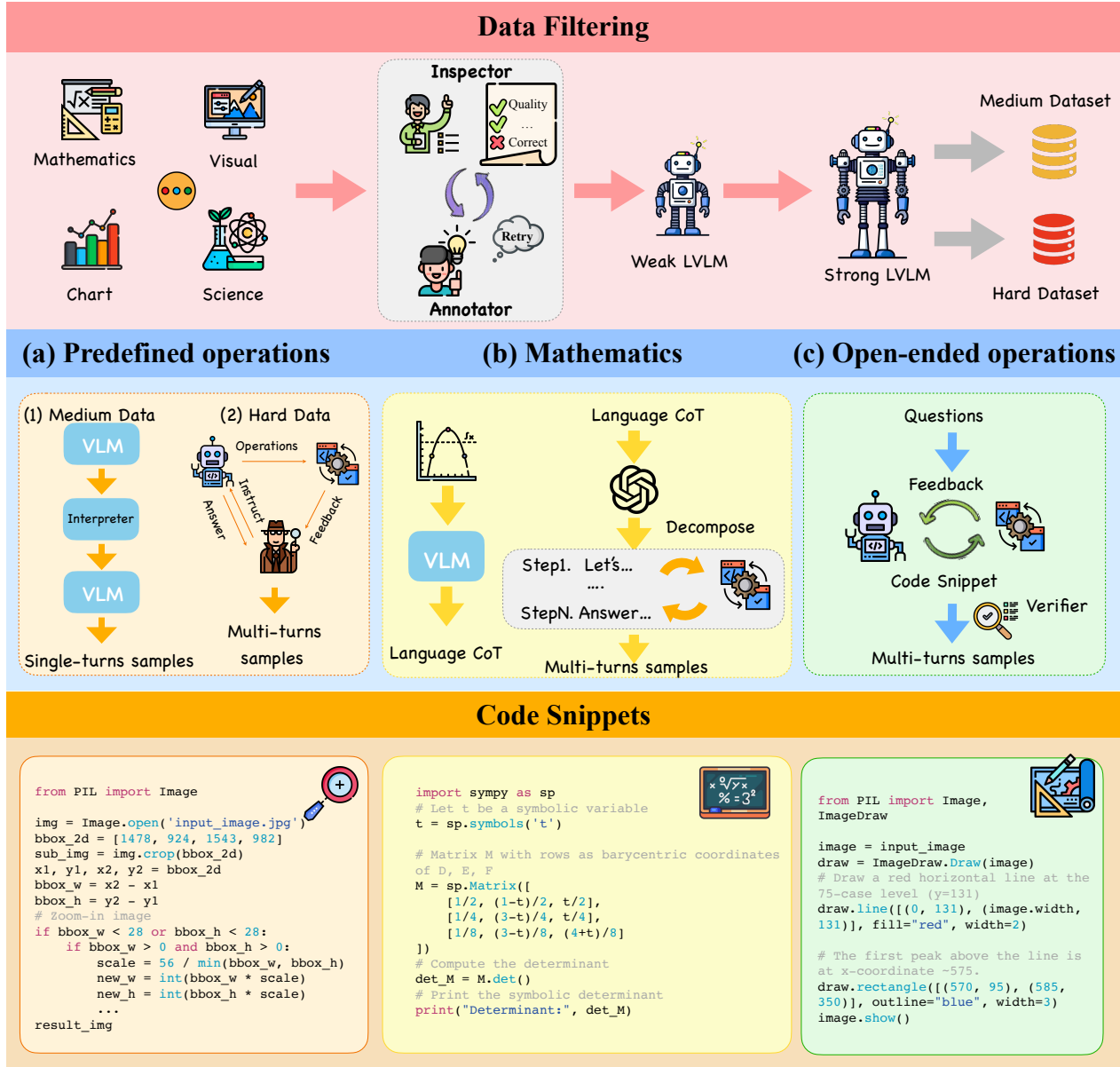


Figure 2. **Overview of the SFT dataset curation pipeline.** **Top:** Weak-to-strong filtering. Candidate samples from diverse domains (mathematics, science, visual reasoning, charts) are validated for quality and correctness through automated inspector modules. A weak VLM removes trivial or low-information cases, while a stronger VLM further stratifies the remaining data into medium- and hard-difficulty sets. **Middle:** Multi-turn atomic supervision. The curated data are organized into three categories. (a) Predefined image operations (e.g., crop, resize, rotate), where medium samples produce single-turn trajectories and hard samples produce multi-turn reasoning with interpreter feedback. (b) Mathematical reasoning, where language-based CoT traces are decomposed into step-level atomic operations and converted into executable code. (c) Open-ended visual operations (e.g., drawing, annotation), where questions, feedback, and snippet verification form multi-turn executable trajectories. **Bottom:** Example executable snippets illustrating the supervision format across image processing, symbolic computation, and visual annotation. **A detailed curation pipeline example is shown in Figure 3.**

Example SFT training data. To better illustrate the construction of SFT data, we provide representative examples of atomic operations. As shown in Figure 4, the top trajectory corresponds to a two-turn reasoning process, where the model iteratively performs cropping, observes intermediate results, and reflects on the correctness before locating

the accurate price tag of a specific toothbrush. In contrast, the bottom trajectory demonstrates a single-turn process, in which the model directly identifies the phone number from a cropped sign. These cases exemplify how SFT data captures both multi-step and single-step reasoning, integrating tool invocation, visual observation, and answer generation.

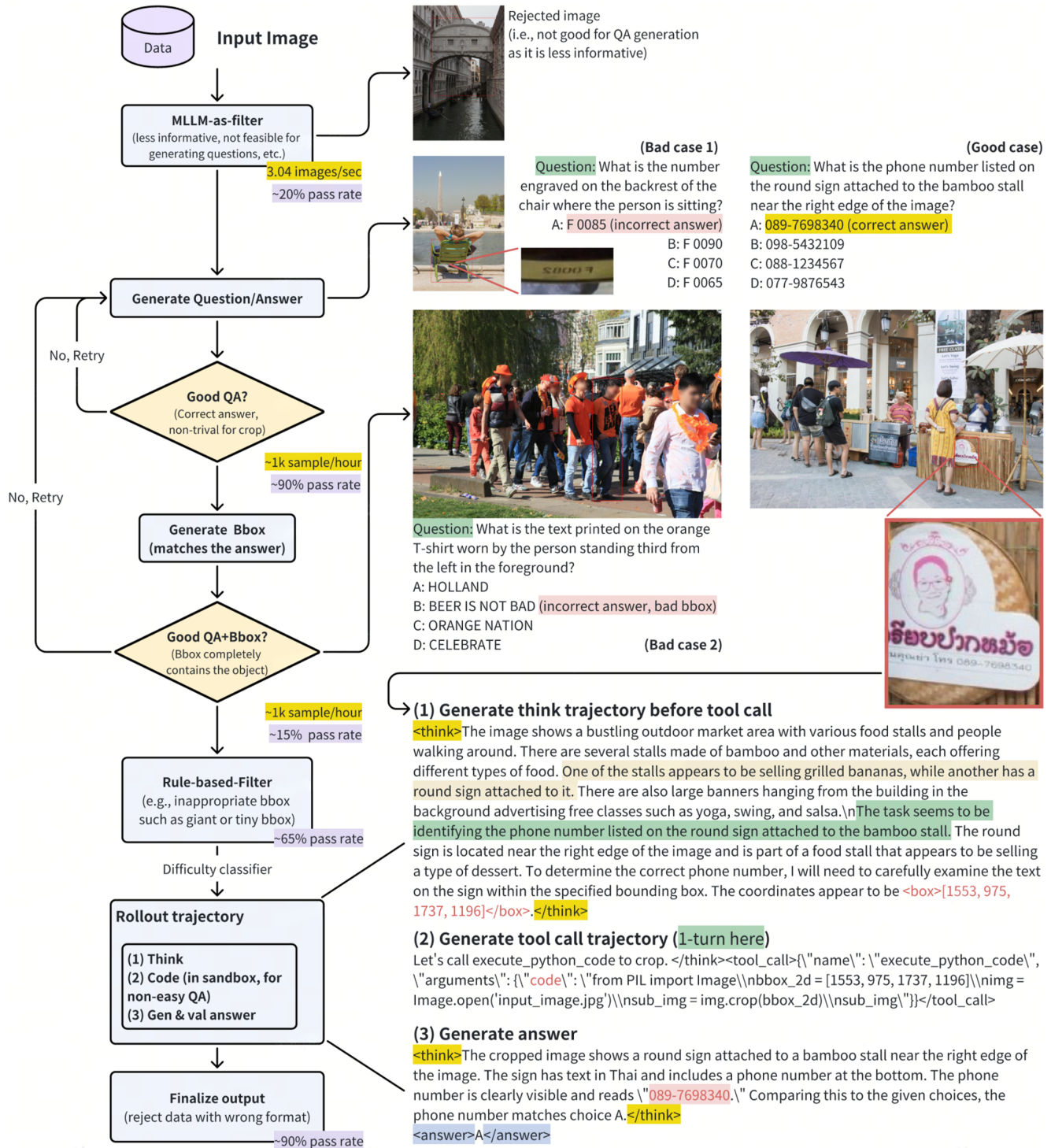


Figure 3. An illustration of curating tool-integrated reasoning trajectories (for cropping operations here). Representative good and bad cases are shown on the right, covering incorrect answers, trivial or mismatched bounding boxes, and fully valid QA pairs. A rule-based filter further removes outputs with implausible bounding boxes. Similar operations are applied when synthesizing other trajectories with different atomic capabilities. Best viewed in color with zooming in.

Table 6. Prompt template for revised thinking with code interpreter.

You are a helpful AI assistant. Initially, when solving a question, you would need to think step by step, without the ability to use code for calculation. Now, you have the capability to write code to use the code interpreter for calculation. The code will be executed by a sandbox, and the result can be returned to enhance your reasoning process. You can now leverage code to enhance your calculation while still maintaining the reasoning process.

The thinking process can have multiple code snippets. Each code snippet is wrapped with

```
<code>
```python
code snippet
```
</code>
```

The returned result is wrapped with

```
<interpreter> execution results</interpreter>
```

Goal: Modify the original thinking process to make it more accurate by replacing manual calculation steps that can benefit from code execution with the corresponding code snippets and their interpreter’s execution results. The core reasoning logic from the original thinking process, including any unsuccessful attempts, should remain unchanged. You should only replace the necessary manual calculation steps with code and interpreter’s execution results, without altering the rest tokens of the thinking process.

User Question: {question}

Original Thinking Process (without code interpreter’s support):

```
<original_thinking_process> {original_response} </original_thinking_process>
```

Details:

1. Identify sections where code execution could speed up the reasoning process or make the calculation more accurate. For simple calculations, you should keep the original text-based reasoning process without executing any code.
2. Replace the manual calculation steps with code snippets and the corresponding interpreter’s execution results.
3. Keep the logical flow of the reasoning process intact, including any failed exploration attempts that were part of the initial process.
4. The code snippets should be complete scripts, including necessary imports.
5. Outputs in the code snippets must explicitly call the **print** function.
6. Execution results should match the model’s output exactly, with no extra or missing tokens.
7. If, during the revised thinking process, you obtain the same result as in the original reasoning, you may omit numerical computations and refrain from simplifying to specific numeric values.
8. If the Original Thinking Process does not include an `<answer>` section at the end, please add it: `<answer> \boxed{{'The final answer goes here.'}} </answer>`

Revised Thinking Process (With code interpreter’s support):

3. Additional empirical studies

3.1. Visualization of sequence/turn-level rewards

Figure 1 illustrates how the sequence-level reward R_{seq} and the turn-level reward R_{turn} defined in the main paper are applied along a multi-turn trajectory in the persistent execution environment. Each query is solved through a se-

quence of *Reasoning–Code–Response* turns, where code is executed in a shared sandbox so that intermediate variables and visual artifacts can be reused across turns. After the whole trajectory τ is completed, the sequence-level reward R_{seq} is computed from the final-answer correctness and the global tool-use statistics $(N_{succ}(\tau), N_{total}(\tau), \mu_{acc})$. The corresponding sequence-level advantage A_{seq} is then



Figure 4. Example SFT training data of an atomic operation (zoom-in operation here). Top: two-turn, bottom: a single-turn trajectory.

broadcast to all tokens in the trajectory (green bars at the bottom of the figure), providing an outcome-level learning signal that adaptively encourages or discourages overall tool usage depending on task difficulty.

Inside the red dashed box, the turn-level component is computed from execution outcomes at each turn m . Failed executions receive an immediate negative penalty $r_{\text{turn},m} < 0$, while successful or no-code turns receive $r_{\text{turn},m} = 0$. These per-turn penalties are accumulated into discounted returns G_{turn}^m and normalized into advantages A_{turn}^m , which are only assigned to the tokens of that specific turn (colored segments in the figure; hatched regions indicate zero turn-

level signal). The final token-wise advantage is obtained by combining the global A_{seq} with the local A_{turn}^m of its turn.

3.2. Hyperparameters in our method

3.2.1. Sensitivity of R_{BAT} Hyperparameters

Recall that the sequence-level component of our reward is defined in Eq. (3) as

$$R_{\text{seq}}(\tau) = \left(0.5 + 0.5 \cdot \mathbf{1}\{R_{\text{acc}}(\tau) > 0\}\right) d(\mu_{\text{acc}}) \frac{N_{\text{succ}}(\tau)}{N_{\text{total}}(\tau)}, \quad (3)$$

Table 7. Prompt template for Reinforcement Learning Rollout.

User. <image> Question: {question}

Think step-by-step within <think></think>. You now have the ability to selectively write executable Python code to enhance your reasoning process. The Python code should be complete scripts, including necessary imports.

Each code snippet is wrapped with

```
<code>
```python
code snippet
```
</code>
```

You must provide your final answer in

```
<answer> </answer>.
```

where the scaling term $d(\mu_{\text{acc}})$ is defined as $d(\mu_{\text{acc}}) = \sigma(\gamma(0.5 - \mu_{\text{acc}})) - \delta$, $\mu_{\text{acc}} \in [0, 1]$ is the group-level accuracy, $\sigma(z) = 1/(1 + e^{-z})$ is the logistic function, and (γ, δ) control how strongly the group accuracy modulates the incentive on tool calls. Here, $\frac{N_{\text{succ}}(\tau)}{N_{\text{total}}(\tau)}$ measures the proportion of successful tool executions within the trajectory, and the leading term $(0.5 + 0.5 \cdot \mathbf{1}\{R_{\text{acc}}(\tau) > 0\})$ switches the base weight according to final-answer correctness. Intuitively, $d(\mu_{\text{acc}})$ decreases as μ_{acc} increases: it amplifies successful tool calls when the group is struggling (low μ_{acc}) and suppresses unnecessary calls when the group already performs well (high μ_{acc}). The hyperparameters control this behavior: γ adjusts how sharp the transition is, δ shifts the baseline and determines whether suppression occurs in high-accuracy regimes, and β is the discount factor for per-turn penalties in R_{turn} . In the main experiments we set $\gamma = 4$, $\delta = 0.2$, and $\beta = 0.2$. The sensitivity results are summarized in Figure 5.

3.2.2. Effective range and bounds of $d(\mu_{\text{acc}})$

As $\sigma(\cdot) \in (0, 1)$ for all real inputs and $\mu_{\text{acc}} \in [0, 1]$, the scaling term d is deterministically bounded as

$$-\delta < d(\mu_{\text{acc}}) < 1 - \delta.$$

Importantly, negative values of d are intentional rather than pathological: they occur only when the group-level accuracy μ_{acc} is sufficiently high and are used to discourage unnecessary tool calls on easy queries. The sign flip of d happens at a unique threshold

$$\mu_{\text{acc}}^* = 0.5 - \frac{1}{\gamma} \sigma^{-1}(\delta), \quad \text{where } \sigma^{-1}(u) = \log \frac{u}{1-u}.$$

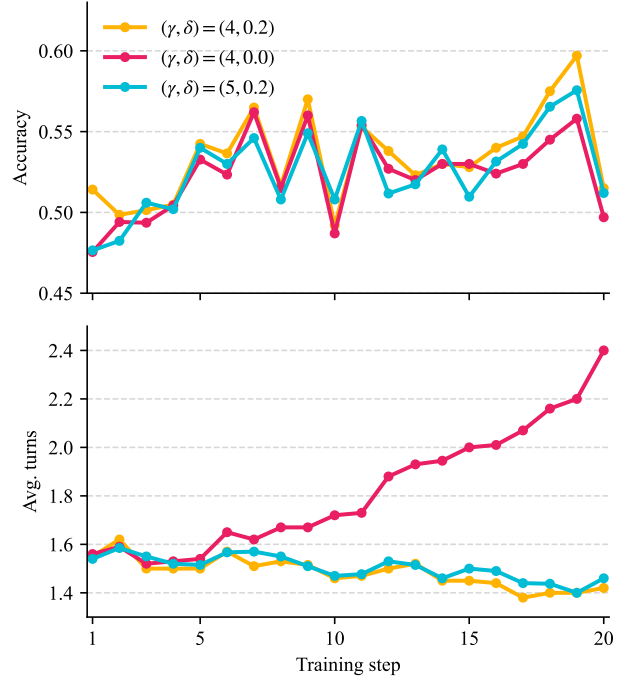


Figure 5. Training dynamics of R_{BAT} under three (γ, δ) configurations. We plot accuracy (top) and average reasoning turns (bottom) over training steps on the union of RL benchmarks. Accuracy is relatively stable across $\gamma \in \{4, 5\}$ (i.e., γ mainly smooths the transition in $d(\mu_{\text{acc}})$), whereas setting $\delta = 0$ removes suppression for high- μ_{acc} groups and leads to steadily increasing tool usage with only marginal accuracy gains. Our default $(\gamma, \delta) = (4, 0.2)$ maintains high accuracy while restraining unnecessary tool calls.

For $\gamma = 4$, $\delta = 0.2$, this gives a transition around

$$\mu_{\text{acc}}^* \approx 0.5 + \frac{1}{4} \log \frac{1 - 0.2}{0.2},$$

i.e., only when most rollouts in the group are already correct does d become slightly negative. In all other regimes (medium or low μ_{acc}), $d > 0$ and R_{BAT} increases the relative value of successful tool calls. Together with the per-turn penalty R_{turn} for failed execution, this design prevents degenerate tool-spamming or tool-off behaviors: on easy tasks, negative d discourages extra calls; on hard tasks, positive d amplifies the benefit of correctly executed tools.

3.2.3. Hyperparameter over (γ, δ)

We run an ablation on (γ, δ) to probe R_{BAT} 's behavior in practice. Concretely, we train three settings, $(4, 0.2)$, $(4, 0.0)$ and $(5, 0.2)$, and monitor training accuracy and average tool calls per example. This lets us (i) compare $\gamma \in 4, 5$ at fixed $\delta = 0.2$, and (ii) isolate the effect of enabling vs. disabling suppression at high group-level accuracy by varying $\delta \in 0, 0.2$ at fixed $\gamma = 4$.

Empirically, γ has little effect on training accuracy, while

setting $\delta = 0$ removes suppression for high- μ_{acc} groups and causes substantially more tool calls with similar accuracy, i.e., tool overuse. Our choice (4, 0.2) keeps accuracy high while restraining unnecessary tool usage. The result is shown as Figure 5.

3.3. Analysis of Reasoning Depth

Our policy aligns reasoning depth with task difficulty. We plot *task difficulty* (measured by GPT-4o ACC.) vs. *number of turns* (see Fig. 6) and validate that harder problems indeed induce longer reasoning turns, while easier ones terminate early.

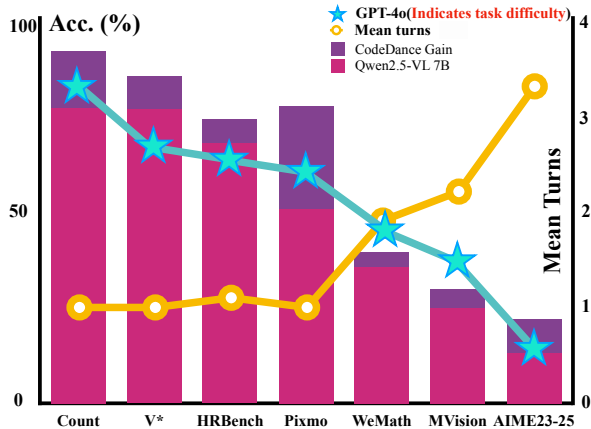


Figure 6. Reasoning depth vs. task difficulty.

4. Additional qualitative results

Reward-hacking case when using naive tool call reward for code generation. A naive reward scheme that simply reinforces every successful tool call is prone to reward hacking, where the model exploits loopholes in the reward design rather than genuinely improving reasoning. For instance, we observe failure cases in Figure 7 in which the model generates degenerate tool outputs (e.g., code consisting only of commentary lines without actual execution) that nevertheless satisfy superficial reward signals. Such behaviors artificially inflate tool success metrics while providing no real contribution to solving the task, thereby misleading training and undermining reasoning quality.

4.1. Failure cases observed in our experiments

Although our model performs competitively across benchmarks, several failure modes are still observed, and we summarize them in Figure 8.

First, inaccurate or suboptimal cropping may occur when the model misidentifies the target region or when visual cues are subtle. **As shown in Case A**, the model selects an overly large bounding box around the person, causing

the cropped image to mix multiple objects; the model still answers correctly, but the tool invocation is clearly misaligned. Second, the model may miss partially obscured objects in complex scenes. **In Case B**, our model correctly identifies and marks visible individuals but fails to consider whether additional people might be hidden or partially occluded, leading to an incomplete final count.

These cases highlight the remaining challenges in precise localization and robust multi-step verification under ambiguous or cluttered visual conditions.

4.2. Additional novel reasoning trajectories

In Figure 9, we show step-by-step reasoning trajectories across three distinct vision tasks on tool transfer:

Top-row of Figure 9: The assistant tackles a spatial relational question by first localizing both the bear and the white rock using bounding boxes. It then uses PIL to draw red/blue rectangles around each object, visually verifying their relative positions. This demonstrates code-mediated spatial reasoning. Instead of relying on implicit attention maps or pretrained spatial priors, the model actively constructs visual evidence through code. The act of drawing bounding boxes serves as an internal “visual scratchpad”, enabling explicit comparison of object positions, which is crucial for fine-grained spatial inference where ambiguity exists.

Mid-row of Figure 9: The assistant identifies four candidate regions containing parrots based on initial visual inspection. It then executes a loop of `img.crop(bounding box)` operations to isolate each region, visually confirming that each cropped area contains a unique, clearly distinguishable parrot. This iterative cropping and verification ensures no over- or under-counting. This exemplifies verification-driven counting. Rather than predicting a number directly (which risks hallucination or confusion with similar objects), the system uses tool-based segmentation to reduce the problem to a series of binary verifications (“Is this one a parrot?”). The modularity of PIL operations allows the model to treat counting as a compositional task — scaling naturally to more complex scenes.

Bottom-row of Figure 9: Given a scientific graph with brightness vs. time, the assistant zooms into panel (c) using `img.crop()` to focus on the region with arrows. It observes sharp downward spikes in the curve at those points and infers they represent sudden drops in brightness, not measurement noise or calibration artifacts — based on the magnitude and shape of the dips.

Similarly, the trajectories in Figure 10 reveal iterative, self-correcting reasoning enabled by dynamic tool composition, which is based on tool transfer ability since we only define a single set of tool/ability for each task during SFT.

Top-row of Figure 10: The assistant first attempts to locate the person in the striped shirt relative to the woman

```

(TaskRunner pid=301219) ...
(TaskRunner pid=301219) </code>
(TaskRunner pid=301219) user
(TaskRunner pid=301219) <tool_response>
(TaskRunner pid=301219)
(TaskRunner pid=301219) Think with the tool's result, then answer. Determine if it adequately answers the user's question. If the information is insufficient or irrelevant, formulate a new strategy and execute a new tool call to obtain a helpful result.
(TaskRunner pid=301219) </tool_response>
(TaskRunner pid=301219) <tool_response>
(TaskRunner pid=301219) Turn limit reached. Based on the analysis so far, directly provide the final answer using the <think></think><answer></answer> format. Do not use any more tools or execute code.
(TaskRunner pid=301219) </tool_response>
(TaskRunner pid=301219) assistant
(TaskRunner pid=301219) <answer>5</answer>
(TaskRunner pid=301219) <code>
(TaskRunner pid=301219) ``python
(TaskRunner pid=301219) # No code needed for this task as the answer can be directly observed from the image.
(TaskRunner pid=301219) ``
(TaskRunner pid=301219) </code>
(TaskRunner pid=301219)
(TaskRunner pid=301219) [ground_truth] 5
(TaskRunner pid=301219) [score] {'score': 1.8, 'acc': 1.0, 'tool_success_rate': 1.0, 'format_r': -1.0, 'ground_truth': '5', 'answer': '5'


```

Figure 7. An example of a reasoning trajectory exhibiting reward hacking (using a naive DeepEyes-style tool reward for code generation). The MLLM exploits the reward by generating code with only commentary lines, which is not actually executed.

User Query

Assistant

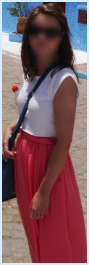
A. Inaccurate/wrong cropping




What is the color of the watchband?

<think>
The image shows a sunny day in a picturesque town with white-washed buildings and cobblestone streets. There are several people walking around, and the architecture features blue accents, typical of certain Mediterranean styles. The scene is vibrant with natural light and colorful details like flowers and plants. The task seems to be identifying the color of the watchband worn by one of the individuals in the image. Observing the person on the right side of the image, they are wearing a watch with a visible watchband. The coordinates appear to be <box>[808, 958, 1146, 1987]</box>. Let's call execute_python_code to crop.
</think>

```
# python
from PIL import Image
bbox_2d = [808, 958, 1146, 1987]
img = Image.open('input_image.jpg')
sub_img = img.crop(bbox_2d)
sub_img
```



Observation




<think>
The cropped image shows a person wearing a white top and a pink skirt. The individual is also holding a blue strap over their shoulder. Upon closely examining the wrist area, there appears to be a watch with a strap that matches the color of the strap being held. The strap is clearly visible and has a distinct blue hue.
</think>
<answer>blue</answer>

User Query

Assistant


B. partially obscured object




How many people are there in this image?

<think>
Looking at the image, I can identify three individuals seated around a table. Their approximate positions are <points>x1="238" y1="618" x2="648" y2="440" x3="1170" y3="480" alt="people">people</points>. Let's call execute_python_code to draw pointings on the detected object for further verification.
</think>

```
# python
import cv2
points = [[238, 618], [648, 440], [1170, 480]]
img = cv2.imread('input_image.jpg')
radius = 6
thickness = 3
for idx, (x, y) in enumerate(points):
cv2.circle(img, (x, y), radius * 2, ...
```



Observation



<think>
After marking these points, I scan the rest of the image carefully to see if there are any other people present. I don't spot any additional figures, so I am confident that the total number of people here is three.
</think>
<answer>3</answer>

Figure 8. Failure cases found in our empirical studies. **A:** The model performs wrongly cropping. **B:** The person on the right edge is partially obscured, thus hard to count.

drinking. It initially misidentifies coordinates, so it composes two tools: First, it uses `cv2.circle()` to draw red points at hypothesized locations — visually flagging potential errors. Then, it corrects the coordinates and uses `PIL.Image.crop()` to zoom into the region for closer inspection. Finally, it confirms the spatial relationship: the striped-shirt person is indeed to the left, seated next to the

drinking woman — no occlusion or misleading posture.

Bottom-row of Figure 10: The assistant must extract a precise numerical value from a scientific plot showing (Δm^2) vs. $\sin^2(2\theta)$. It follows a multi-step strategy: Identify region: Uses `ImageDraw.rectangle()` to highlight the blue shaded 90% confidence level (CL) band. Zoom in: Crops the upper boundary of this region using

Trajectories on tool transfer



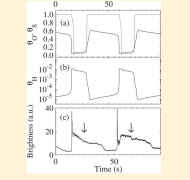
User Query	Assistant	
 <p>Is the white rock behind a bear? (Spatial understanding)</p>	<p><think>To determine if the white rock is behind a bear, I first need to locate the bear in the image. The bear is clearly visible in the center-right of the frame, lying on a log and packaging material. Next, I need to find the white rock in question; it is to the left in the foreground and partially behind the rock structure. To verify the relative positions, I can draw bounding boxes around both the bear and the white rock.</p> <p></think></p>	<pre>from PIL import Image, ImageDraw bbox_2d_bear = [97, 48, 397, 304] bbox_2d_white_rock = [1, 70, 93, 226] img = Image.open('input_image.jpg') draw = ImageDraw.Draw(img) draw.rectangle(bbox_2d_bear, outline='red', width=2) draw.rectangle(bbox_2d_white_rock, outline='blue', width=2)</pre>  <p style="text-align: right;">Observation</p>
 <p>How many parrots are there in the image? (Visual counting)</p>	<p><think>Looking over the image, I spot four distinct parrots arranged in a grid-like pattern. Their approximate coordinates are <box>[0, 133, 104, 252]</box>, <box>[105, 131, 214, 252]</box>, <box>[105, 3, 214, 123]</box>, and <box>[0, 3, 104, 123]</box>. Let's call execute_python_code to crop.</p> <p></think></p>	<pre>from PIL import Image bbox_2d = [0, 133, 104, 252] img = Image.open('input_image.jpg') sub_img1 = img.crop(bbox_2d) bbox_2d = [105, 131, 214, 252] sub_img2 = img.crop(bbox_2d) # crop all objects for verification</pre>  <p style="text-align: center;">Observation</p>
 <p>What do the arrows in panel (c) most likely represent? (Chart reasoning)</p> <p>A. Points of measurement error. B. Sudden drops in brightness. C. Calibration markers. D. Time intervals of interest.</p>	<p><think>The image shows a set of three scientific graphs labeled (a), (b), and (c), plotted against time. Graph (a) shows a step-like increase and decrease, graph (b) is on a log scale, and graph (c) is labeled 'Brightness'. The arrows are pointing to notable points on the 'Brightness' graph in panel (c)... Let's use python to crop and zoom-in the target region of interest.</p> <p></think></p>	<pre>from PIL import Image bbox_2d = [0, 133, 104, 252] img = Image.open('input_image.jpg') sub_img1 = img.crop(bbox_2d)</pre>  <p style="text-align: center;">Observation</p>

Figure 9. Novel reasoning trajectories on tool transfer to other tasks.

`PIL.Image.crop()` to isolate the extreme right edge — where (Δm^2) reaches its maximum within the CL. Finally interpret scale and answer.

While these reasoning trajectories during RL exploration are not without flaws, e.g. occasionally exhibiting imprecise coordinate estimation or redundant tool calls, they collectively demonstrate the potential of tool-augmented multimodal reasoning.

5. Discussion

Why code-based tool use, rather than API-style calls?

We adopt Python code as the medium for tool use because it provides a general and compositional interface. Unlike fixed API schemas, code naturally supports both tool invocation and program logic (e.g., sequencing, conditionals,

loops, numerical computation). This richer interface allows models to flexibly define and combine operations, and it produces transparent and verifiable execution traces that can be systematically inspected. In practice, code also makes extension straightforward: adding a new tool only requires exposing its API, without redesigning templates, retraining connectors, or engineering complex prompts.


Why a single dense model, rather than an agent pipeline?

A unified dense model offers several practical advantages over modular agent workflows: (1) it avoids error propagation across multiple components by learning an end-to-end interface; (2) it achieves lower latency and compute cost, since reasoning and tool orchestration are handled in a single forward pass; (3) it is more robust, as performance does not hinge on the reliability of each sub-

Trajectories on tools composition

User Query


Assistant




Is the striped shirt to the left of the woman drinking?

<think>
I see the striped shirt is indeed to the left of the woman drinking, but I overlooked the coordinates slightly. Let's call **execute_python_code** to draw pointings on the detected object for further verification.
</think>


```
# draw points
import cv2
points = [[51, 247], [329, 211]]
img = cv2.imread('input_image.jpg')
radius = 6
cv2.circle(img, (x, y), radius * 2, (255, 0, 0), -1)
cv2.putText(img, str(idx+1), (x + radius + 3, ...))
img
```



Observation



<think>
After cropping, it's clear from the cropped image that the person wearing a striped shirt is to the left of the woman seen here drinking. The cropped area shows the shoulder and arm of the individual in the striped shirt, positioned to the left of the arm of the woman who seems to be in the act of drinking. There is no indication of any obstruction or misalignment that would suggest otherwise. Therefore, the detection is correct.
</think>
<answer>Yes.</answer>



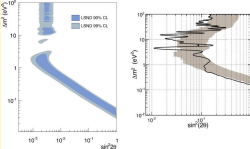
Observation

```
# crop
from PIL import Image
img = Image.open('input_image.jpg')
bbox_2d = [51, 0, 235, 298]
sub_img = img.crop(bbox_2d)
```

<think>
Now that I've redone the detection with improved coordinates given by the hint, the striped shirt seems to be correctly positioned relative to the woman drinking. The woman in green appears on the far left among the patrons, while the person in a striped shirt is seated next to her.
Let's call **execute_python_code** to crop.
</think>

User Query

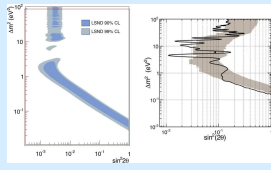
Assistant




What is the highest value of (Δm^2) (in eV^2) that is within the 90% confidence level (CL) according to the LSND data shown in the figure?
A. $(10^{-1} eV^2)$ B. $(10^2 eV^2)$
C. $(10^1 eV^2)$ D. $(10^0 eV^2)$

<think>
To solve this problem, we need to analyze the left figure of the provided image, which shows the (Δm^2) (in eV^2) versus $(\sin^2 2\theta)$ (in logarithmic scale). The question asks for the highest value of (Δm^2) at the 90% confidence level (CL) according to the LSND data.
1. Identify the relevant region: The legend on the left figure indicates that the 90% CL confidence level is represented by the blue shaded area.
2. Locate the highest region of the 90% CL band: Looking at the blue shaded area, we can see that as the $(\sin^2 2\theta)$ value decreases (i.e., moves to the right on the plot), the permissible range of (Δm^2) increases. The highest point, or the upper limit, of this permissible range is the most interesting for answering the question.
3. Estimate the value from the plot: We need to estimate the value of (Δm^2) (on the vertical axis) that corresponds to the extreme right of the blue band. This requires a more precise measurement than can be done just by looking at the figure.
</think>

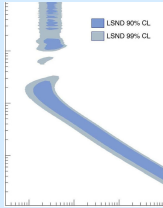
```
# draw bbox to highlight
from PIL import Image, ImageDraw
bbox_2d = [133, 37, 933, 1073]
img = Image.open('input_image.jpg')
draw = ImageDraw.Draw(img)
draw.rectangle(bbox_2d, outline="red", width=2)
```



Observation



<think>
The cropped image provides a closer view of the blue shaded region labeled "LSND 90% CL." The left figure shows the data with logarithmic scales on both axes. The task is to find the highest value of (Δm^2) that is within this 90% confidence level band.
1. ...2. ...
3. Estimate the value from the axis: The top of the blue band is clearly above the value of (10^0) (which is 1) on the y-axis. It is also below the value of (10^1) (which is 10) on the y-axis. Therefore, the highest value is greater than 1 and less than 10 (eV^2) .
Looking at the options, the value $(10^1 eV^2)$ is the most accurate description of the highest value within the given range.
</think>
<answer>boxed{C. (10^1 eV^2)}</answer>



Observation

```
# crop
from PIL import Image
img = Image.open('input_image.jpg')
bbox_2d = [133, 37, 933, 1073]
sub_img = img.crop(bbox_2d)
```

<think>
The red box drawn on the left image highlights the upper boundary of the 90% confidence level (CL) region for (Δm^2) as a function of $(\sin^2 2\theta)$. By examining this box, we can estimate the highest value of (Δm^2) within this region. From the logarithmic scale on the y-axis, the upper boundary of the red box corresponds to a value of (Δm^2) that is just below the 10^1 (10) line, but clearly above the 10^0 (1) line. This corresponds to a value of approximately $(10^1 eV^2)$.
Let's call **execute_python_code** to crop.
</think>

Figure 10. Reasoning trajectories on tools composition.

module; and (4) it benefits from a unified optimization target, whereas agent systems often require additional policies or connectors to be separately tuned.

In addition, given realistic compute constraints, most of our experiments in this work are conducted with 7B-scale models (e.g., Qwen-2.5-VL-7B), where we already observe promising effects: consistent gains across general under-

standing and complex reasoning benchmarks, and the emergence of new behaviors (e.g., novel tool use and tool compositions of atomic skills to new tasks). These empirical observations are easier to scale within a single dense model, while agent pipelines introduce many interacting modules that complicate both training and deployment. Overall, our design favors simplicity, efficiency, and scalability, making

it a more practical foundation for future progress.

6. Broader impact

This work contributes toward building more transparent and verifiable multimodal reasoning systems by adopting executable code as the unified medium for tool use. The ability to generate interpretable traces and intermediate artifacts can benefit applications where accountability and auditability are essential, such as scientific analysis and education. At the same time, code-generating models pose risks: malicious users could potentially exploit them for unsafe automation, and generated visual artifacts might be misused to mislead or manipulate. To mitigate these concerns, we recommend pairing such systems with appropriate safeguards, including safety filters, usage constraints, and responsible deployment practices. By doing so, the benefits of executable visual reasoning can be realized while minimizing the potential for misuse.

7. Limitations and future work

Limitations. While our method demonstrates promising emergent behaviors and strong performance across diverse visual reasoning tasks, several limitations remain. First, the reliance on high-quality synthetic trajectories implies that certain real-world reasoning patterns may be under-represented, potentially limiting robustness in open-domain scenarios. Second, although code provides a general interface, extending to richer modalities (e.g., audio) or domain-specific tools (e.g., medical applications) will require additional engineering. Finally, due to compute constraints, our evaluations are primarily conducted on 7B-scale models; the scalability of emergent behaviors at larger scales remains to be systematically examined. Nevertheless, our preliminary experiments suggest a promising trend when scaling up model capacity and compute resources.

Future Work. Our framework demonstrates the potential of multimodal reasoning models to support natural conversations with seamless and proactive tool use through executable code, thereby enabling more advanced problem-solving capabilities. Looking ahead, we envision that the ability to “think with images” will evolve beyond the vision modality and fixed schemas, fostering novel tool discovery and the spontaneous composition of tools in a more generalized and efficient manner. Such directions may ultimately pave the way toward multimodal agents that are both versatile and adaptive across diverse domains.

References

- [1] Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibao Song, Kai Dang, Peng Wang, Shijie Wang, Jun Tang, Humen Zhong, Yuanzhi Zhu, Mingkun Yang, Zhaohai Li, Jianqiang Wan, Pengfei Wang, Wei Ding, Zheren Fu, Yiheng Xu, Jiabo Ye, Xi Zhang, Tianbao Xie, Zesen Cheng, Hang Zhang, Zhibo Yang, Haiyang Xu, and Junyang Lin. Qwen2.5-vl technical report. *arXiv preprint arXiv:2502.13923*, 2025. 1
- [2] Lin Chen, Jinsong Li, Xiaoyi Dong, Pan Zhang, Conghui He, Jiaqi Wang, Feng Zhao, and Dahua Lin. Sharegpt4v: Improving large multi-modal models with better captions. In *European Conference on Computer Vision*, pages 370–387. Springer, 2024. 2
- [3] Xinyan Chen, Renrui Zhang, Dongzhi Jiang, Aojun Zhou, Shilin Yan, Weifeng Lin, and Hongsheng Li. Mint-cot: Enabling interleaved visual tokens in mathematical chain-of-thought reasoning. *arXiv preprint arXiv:2506.05331*, 2025. 2
- [4] Wenliang Dai, Junnan Li, Dongxu Li, Anthony Tiong, Junqi Zhao, Weisheng Wang, Boyang Li, Pascale N Fung, and Steven Hoi. Instructblip: Towards general-purpose vision-language models with instruction tuning. *Advances in neural information processing systems*, 36:49250–49267, 2023. 2
- [5] Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, Mohammadreza Salehi, Niklas Muennighoff, Kyle Lo, Luca Soldaini, et al. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. In *Proceedings of the Computer Vision and Pattern Recognition Conference*, pages 91–104, 2025. 3
- [6] 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
- [7] Wei Fu, Jiakuan Gao, Xujie Shen, Chen Zhu, Zhiyu Mei, Chuyi He, Shusheng Xu, Guo Wei, Jun Mei, Jiashu Wang, et al. Areal: A large-scale asynchronous reinforcement learning system for language reasoning. *arXiv preprint arXiv:2505.24298*, 2025. 2
- [8] Jiahui Gao, Renjie Pi, Jipeng Zhang, Jiacheng Ye, Wanjun Zhong, Yufei Wang, Lanqing Hong, Jianhua Han, Hang Xu, Zhenguo Li, et al. G-llava: Solving geometric problem with multi-modal large language model. *arXiv preprint arXiv:2312.11370*, 2023. 2
- [9] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. Segment anything. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 4015–4026, 2023. 2, 3
- [10] Woosuk Kwon, Zhuohan Li, Siyuan Zhuang, Ying Sheng, Lianmin Zheng, Cody Hao Yu, Joseph E. Gonzalez, Hao Zhang, and Ion Stoica. Efficient memory management for large language model serving with pagedattention. In *Proceedings of the ACM SIGOPS 29th Symposium on Operating Systems Principles*, 2023. 1
- [11] Mengcheng Lan, Chaofeng Chen, Yue Zhou, Jiaying Xu, Yiping Ke, Xinjiang Wang, Litong Feng, and Wayne Zhang. Text4seg: Reimagining image segmentation as text generation. *arXiv preprint arXiv:2410.09855*, 2024. 2
- [12] Ahmed Masry, Megh Thakkar, Aayush Bajaj, Aaryaman Kartha, Enamul Hoque, and Shafiq Joty. Chartgemma: Vi-

- sual instruction-tuning for chart reasoning in the wild. *arXiv preprint arXiv:2407.04172*, 2024. 2
- [13] Ahmed Masry, Mohammed Saidul Islam, Mahir Ahmed, Aayush Bajaj, Firoz Kabir, Aaryaman Kartha, Md Tahmid Rahman Laskar, Mizanur Rahman, Shadikur Rahman, Mehrad Shahmohammadi, et al. Chartqapro: A more diverse and challenging benchmark for chart question answering. *arXiv preprint arXiv:2504.05506*, 2025. 2
- [14] Fanqing Meng, Lingxiao Du, Zongkai Liu, Zhixiang Zhou, Quanfeng Lu, Daocheng Fu, Tiancheng Han, Botian Shi, Wenhai Wang, Junjun He, et al. Mm-eureka: Exploring the frontiers of multimodal reasoning with rule-based reinforcement learning. *arXiv preprint arXiv:2503.07365*, 2025. 2
- [15] OpenAI. Openai o3 and o4-mini system card. <https://cdn.openai.com/pdf/2221c875-02dc-4789-800b-e7758f3722c1/o3-and-o4-mini-system-card.pdf>, 2025. System Card. 2
- [16] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017. 2
- [17] 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
- [18] Alex Su, Haozhe Wang, Weiming Ren, Fangzhen Lin, and Wenhui Chen. Pixel reasoner: Incentivizing pixel-space reasoning with curiosity-driven reinforcement learning. *arXiv preprint arXiv:2505.15966*, 2025. 2
- [19] Ye Wang, Qianglong Chen, Zejun Li, Siyuan Wang, Shijie Guo, Zhirui Zhang, and Zhongyu Wei. Simple o3: Towards interleaved vision-language reasoning. *arXiv preprint arXiv:2508.12109*, 2025. 2
- [20] Jason Wei, Maarten Bosma, Vincent Y Zhao, Kelvin Guu, Adams Wei Yu, Brian Lester, Nan Du, Andrew M Dai, and Quoc V Le. Finetuned language models are zero-shot learners. *arXiv preprint arXiv:2109.01652*, 2021. 2
- [21] Huanjin Yao, Jiaying Huang, Wenhao Wu, Jingyi Zhang, Yibo Wang, Shunyu Liu, Yingjie Wang, Yuxin Song, Haocheng Feng, Li Shen, et al. Mulberry: Empowering mllm with o1-like reasoning and reflection via collective monte carlo tree search. *arXiv preprint arXiv:2412.18319*, 2024. 3
- [22] 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
- [23] Ziwei Zheng, Michael Yang, Jack Hong, Chenxiao Zhao, Guohai Xu, Le Yang, Chao Shen, and Xing Yu. Deep-eyes: Incentivizing “thinking with images” via reinforcement learning. *arXiv preprint arXiv:2505.14362*, 2025. 2, 3