Visual Reasoning through Tool-supervised Reinforcement Learning

Supplementary Material

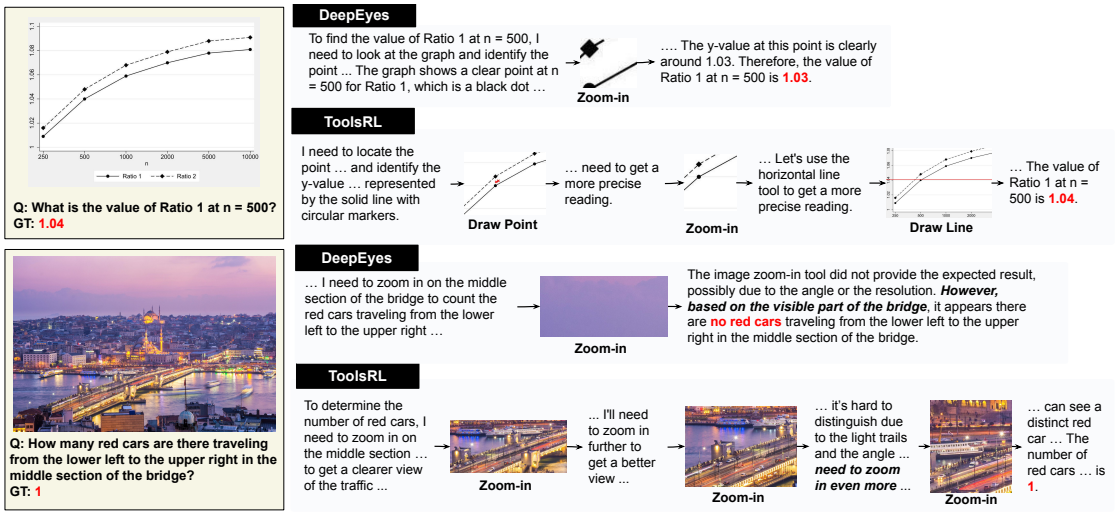


Figure 1. Qualitative comparison with DeepEyes.

1. Overview

Here we provide additional qualitative comparison (Sec. 2), details on our synthetic dataset generation (Sec. 3), tool-supervised design and ablations (Sec. 4), curriculum and ablations (Sec. 5), and the prompts and tool APIs used in both stages of training (Sec. 6).

2. Qualitative Comparison

More qualitative results (ToolsRL vs. DeepEyes) are displayed in Fig. 1, clearly showing ToolsRL is able to use tools more compositely as well as more effectively and precisely.

3. Synthetic Dataset Generation Details

3.1. Augmented Document Datasets

We follow the same augmentation pipeline for training and evaluation.

DocVQA-RF and InfoVQA-RF. As described in the main paper (Sec. 4), DocVQA and InfoVQA images are rotated by $90°/180°/270°$ or flipped horizontally/vertically, with a transformation applied with probability 0.7 and sampled uniformly from this set. The *same* rotation/flip distribution is used for both the 3k augmented DocVQA training subset and for constructing the DocVQA-RF and InfoVQA-RF evaluation benchmarks, ensuring that orientation statistics match between training and test.

InfoVQA-Res. For InfoVQA-Res, we again follow the construction in Sec. 4: we select InfoVQA images whose maximum edge length exceeds 1024 pixels and resize them so that the maximum dimension is at most 512 pixels while preserving aspect ratio.

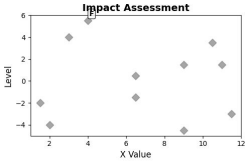
3.2. Synthetic Chart Datasets

We generate two synthetic chart datasets, **Read-Value** and **Compare-and-Count**, designed to provide unambiguous ground-truth supervision for drawing tools. These datasets are generated programmatically to ensure precise knowledge of data point locations and values.

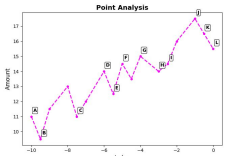
Read-Value. This dataset consists of synthetically generated scatter and line charts, each paired with a small number of question–answer pairs. Charts are rendered at up to 768 px on the longer edge, with axis ranges spanning moderate numeric intervals to keep coordinates readable.

- **Task:** Questions ask for the x-coordinate, y-coordinate, or full (x, y) coordinates of a labeled point (e.g., "What is the y-value of point B?"). Axis-aware variants reuse the chart titles and axis labels to make prompts natural.
- **Chart Design:** Labeled points are placed on scatter or polyline plots with diverse color/marker styles. Each label (letters, numbers, or alphanumeric IDs) is positioned with small offsets so text does not occlude the point, mirroring real chart layouts.
- **Ground Truth and Tool Supervision:** For every labeled point, we store both data coordinates and pixel coordinates on the rendered image. During training, the model is supervised to use `DrawLine` to draw a horizontal or vertical line from the point to the corresponding axis; this provides precise supervision for where the line should touch the axis and enables accurate re-

Question: This is the image. Try to use tools to get the correct orientation, and report the image index of the correct one (0 is the original image, index starts from 0). The answer should be an integer. For example, if you rotate the image and it becomes correct at image 3, answer: <answer>3</answer>



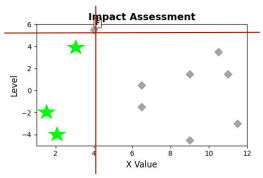
Question: Given the question "How many points have x > F and y < F?", draw reference lines from the reference point F, then report which image index shows the correct tool usage (0 is the original image, index starts from 0). Answer as an int. Example: if correct at image 2, write: <answer>2</answer>



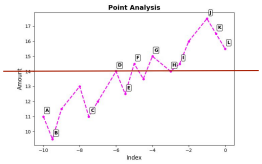
Question: Given the question "Give the x and y coordinates of point D.", draw lines on the point to assist with this task, then indicate which image index has the correctly drawn lines (0 is the original image, index starts from 0). Answer format: single int. For example, correct at image 1 means: <answer>1</answer>

Question: Given "Is the hot dog on the right side of the cup", the mentioned objects are: hot dog, cup. Zoom in to find these objects and report the zoom call index for each (index starts from 1). For instance, if "ruler" is the 4 zoom call and "book" is the 8, answer: <answer>{"ruler": 4, "book": 8}</answer>

GT Tool: Rotate, Flip



GT Tools: Lines, Points



GT Tools: Lines

GT Tools: Zoom-ins

Figure 2. Visual examples of Stage 1 tool-supervised training data. From left to right: (a) document Rotate/Flip orientation correction; (b) chart Read-Value with reference lines; (c) chart Compare-and-Count with marked points and a threshold line; and (d) zoom-in object localization. In each column, the top row shows the question and initial image presented to the model, while the bottom row overlays the ground-truth tool supervision (target orientation, reference lines and points, or zoom-in box) used to compute the rewards.

ward computation for both the drawn primitive and the numeric answer.

Compare-and-Count. This dataset contains synthetically generated scatter and line charts, each paired with one comparison question. Images are rendered up to 512 px on the long edge, and for each question we re-render the plot so that only the single *reference* point is visibly labeled.

- **Task:** Questions ask how many other points satisfy a relational condition with respect to the reference point, such as $x{>}x_{\text{ref}}$, $y{<}y_{\text{ref}}$, both-axes comparisons, or mixed conditions (e.g., "How many points have x greater than F and y less than F?").
- **Chart Design:** Under the hood, each chart contains 8–20 rounded points spanning moderate axis ranges, with labels shuffled across letters, numbers, or alphanumeric IDs to avoid positional shortcuts. Although only the reference label is shown in the rendered image, we retain full coordinate metadata for all points.
- **Ground Truth and Tool Supervision:** For every question, we precompute which points qualify under the comparison rule and store their coordinates and labels. During training, the model is encouraged to use `DrawPoint` to mark qualifying points and `DrawLine` to indicate threshold boundaries when appropriate (e.g., a vertical line at $x{=}x_{\text{ref}}$). This setup yields dense supervision for both counting behavior and precise spatial localization of the counted set.

These synthetic tasks serve as a curriculum capability designed to teach the model precise spatial manipulation and visual working memory usage before it attempts more complex, real-world chart reasoning tasks in Stage 2. Figure 2 shows representative Stage 1 supervision examples across document-rotation, chart reading/counting, and zoom-in tasks.

4. Tool-Supervised Design and Ablation Details

4.1. Zoom-in: ModF1 Reward and Ablation

Zoom-in in our setting is *not* an object detection task: we only need the zoom window to comfortably cover the region of interest, not to produce a tight bounding box. We therefore adopt a Modified F1 (ModF1) score that down-weights false positives with $w_{\text{fp}}{=}0.1$ while keeping $w_{\text{fn}}{=}1.0$, so missing the target area is penalized much more than including extra background. As illustrated in Fig. 3, this reward gives full credit to generous crops that contain the GT box, whereas a symmetric F1 would assign a low score and discourage such safe zoom behavior.

4.2. Draw: Discrete vs. Continuous Rewards

For `DrawLine` and `DrawPoint` tasks, we compare a simple discrete reward against the continuous margin-based reward used in our main draw metric, both defined in terms of the same distance $d(u, g)$ and tolerance T_g between a predicted primitive u and a ground-truth primitive g in Sec. 3.2.

The discrete reward only gives credit when the prediction lands inside the tolerance window:

$$R_{\text{discrete}}(u, g) = \mathbb{1}\big(d(u, g) < 10\big), \tag{1}$$

while the continuous version scales linearly with distance:

$$s(u, g) = \max\Big(0,\ 1 - \tfrac{d(u,g)}{T_g}\Big). \tag{2}$$

The discrete signal is too sparse: at the start of training the model almost never discovers useful draw behavior, and the average usage of the point-marking tool stays extremely low (0.23 calls per sample). After switching to the continuous reward (which gives partial credit to near misses), the model begins to actively explore

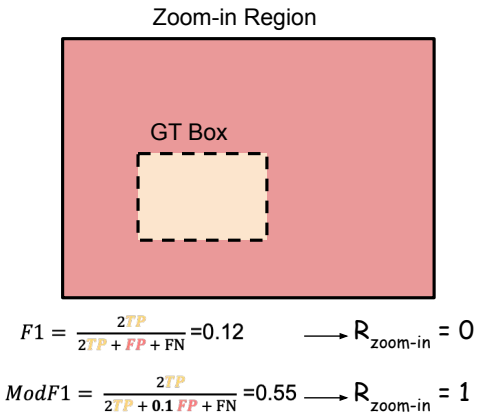


Figure 3. Illustration of standard F1 vs. ModF1 for zoom-in. In this example, the predicted zoom region fully contains the ground-truth (GT) box but is much larger. Standard F1 yields a low score (and thus $R_{\text{zoom-in}}$=0) because the FP area dominates. ModF1, with a smaller FP weight (w_{fp}=0.1), assigns a much higher score (and $R_{\text{zoom-in}}$=1), reflecting that generous crops that include target area should still receive full credit.

drawing, and the average mark-point usage rises to 0.643. This empirical gap shows that more informative, continuous rewards are essential for learning reliable draw-tool usage.

4.3. Rotate and Flip: Training with Mixed Orientations

As observed in our ablation study in Tab. 3, we find it crucial to use *only* the augmented (rotated/flipped) samples in Stage 1 training. If the training mix includes many canonical (un-augmented) documents, the model learns a shortcut: it predicts the answer directly assuming the image is upright, as this strategy works for the canonical subset (which is often easier to answer). By restricting Stage 1 to only rotated/flipped examples, we force the model to use the `Rotate` or `Flip` tools to recover the readable orientation before answering. This enforced active perception prevents reward hacking and ensures the tool-use policy is robustly learned. Figure 4 illustrates this phenomenon: when canonical documents are included in the training mix, the model shortcuts by predicting answer index 0 directly; training exclusively on rotated/flipped samples forces active tool use and leads to robust behavior.

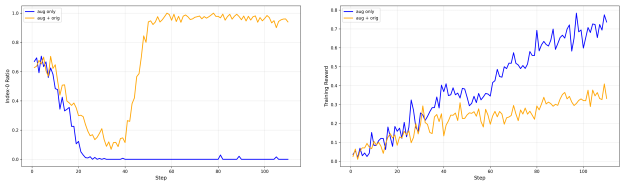


Figure 4. Analysis of the Stage-1 rotation/flip training design. Mixing original and augmented documents ("aug + orig") encourages a shortcut where the model always predicts index 0 in answer, while training on augmented-only samples ("aug only") removes this shortcut and yields higher reward.

5. Curriculum and Ablation Details

5.1. Overview of Curricula and Baselines

Table 2 in the main paper compares a family of training strategies: (i) *Accuracy Reward Only*, which optimizes answer correctness without any explicit tool signal; (ii) *Tool-Conditioned Reward*, which adds a scalar bonus when tools are used on correctly answered trajectories; and (iii) our *Tool-supervision Curricula*, where Stage 1 is trained with global and/or answer-conditioned tool rewards and Stage 2 uses only the answer-accuracy reward.

Tool-Conditioned Reward (DeepEyes baseline). Following the DeepEyes setup, we define a binary tool-conditioned bonus on each trajectory τ using the answer reward $R_{\text{answer}}(\tau)$ and the total number of tool calls tool_count(τ):

$$R_{\text{tool\_cond}}(\tau) = \mathbb{1}\big[R_{\text{answer}}(\tau) > 0.5\big] \cdot \mathbb{1}\big[\text{tool\_count}(\tau) > 0\big] \quad (3)$$

where tool_count(τ) counts all native visual-tool API calls. The DeepEyes baseline then optimizes

$$R_{\text{DeepEyes}}(\tau) = R_{\text{answer}}(\tau) + R_{\text{format}}(\tau) + R_{\text{tool\_cond}}(\tau) \quad (4)$$

so tools are rewarded only when the final answer is already correct, without any guidance on which tools to invoke or how to structure tool trajectories.

5.2. Training Dynamics vs Tool-Conditioned Reward

Figure 5 (left) compares the average number of tool calls per step during Stage 2 training for Accuracy Reward Only, Tool-Conditioned Reward, and our ToolsRL curriculum. All three settings are trained on the same data with identical prompts and the same answer-accuracy objective; the only difference is whether having Stage 1 for tool supervision. Accuracy Reward Only almost never invokes tools (staying near one call per sample), and Tool-Conditioned Reward increases tool usage only modestly. In contrast, the model trained with our Tool-supervision Curriculum consistently uses tools much more frequently (around 3–5 calls per sample) even though Stage 2 has *no explicit tool bonus*, indicating that Stage 1 tool rewards induce a persistent, tool-centric reasoning policy rather than transient reward hacking.

5.3. Tool Call Rates of Ablated Curricula

Figure 5 (right) analyzes Stage 1 ablations of our curriculum. Using only answer-conditioned tool supervision $R_{\text{tool}}^{\text{answer}}$ yields relatively low tool counts: the agent learns to be conservative and invokes tools only when they directly affect the final answer, which limits exploration. In contrast, using only global tool supervision $R_{\text{tool}}^{\text{global}}$ leads to very high call rates and many redundant actions, since any tool call on the trajectory is rewarded regardless of its usefulness, encouraging inefficient behavior. Our full ToolsRL curriculum, which combines global and answer-conditioned rewards, lies between these extremes: it promotes rich exploration early in training while gradually shaping the policy toward efficient, task-relevant tool usage.

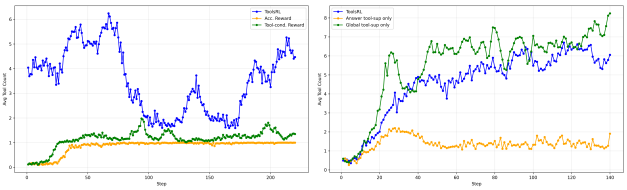


Figure 5. Average tool usage during training across ablation experiments. **Left:** comparison between Accuracy Reward Only, Tool-Conditioned Reward, and our ToolsRL during Stage 2, all trained on the same data, prompts, and answer-accuracy objective. **Right:** Stage 1 average tool calls for our full curriculum versus answer-only and global-only tool supervision.

6. Prompt and Tool Argument Details

6.1. Stage 1 Tool-supervised Prompts

Stage 1 uses task-specific system prompts that expose only the tools needed for each supervision regime, paired with a shared lightweight user instruction. The literal system prompts are shown in Tables 1–3; for tool-argument details, refer to the Tool API in Sec. 6.3.

- **Read-Value and Compare-and-Count.** The system prompt advertises the point and line tools—`image_mark_points_tool`, `image_draw_horizontal_line_tool`, and `image_draw_vertical_line_tool`—and emphasizes that all coordinates must be given in pixel space on the rendered chart (image columns/rows), not axis values.
- **Rotate/Flip tasks.** The system prompt exposes `image_rotate_tool` and `image_flip_tool` and explains that angles are in degrees (positive = clockwise) and flips are either horizontal or vertical.
- **Zoom-in tasks.** The system prompt exposes only `image_zoom_in_tool` with a bounding-box interface for cropping.

All Stage 1 datasets share the same concise user prompt:

Shared User Prompt (Stage 1 and Stage 2)

```
{Question}
Think first, call tools if needed, then answer.
Format as: \texttt{<think>...</think> <tool_call
>...</tool_call>} (if tools needed) or \texttt{<
think>...</think> <answer>...</answer>} (if you
know the answer).
```

This enforces a consistent trace structure across zoom, rotate/flip, and drawing tasks without leaking reward details.

6.2. Stage 2 QA Prompts

Stage 2 switches to a unified QA setting where a single system prompt exposes the full toolbox. The full Stage 2 system prompt is listed in Table 4; for tool-argument details, refer to the Tool API in Sec. 6.3. The system prompt advertises `image_zoom_in_tool`, `image_rotate_tool`, `image_flip_tool`, `image_draw_horizontal_line_tool`, `image_draw_vertical_line_tool`, and `image_mark_points_tool` together, using the same JSON-style `<tool_call>` convention as in Stage 1. The user prompt is kept identical to Stage 1, so the main change between stages is the reward objective (answer accuracy) and toolbox breadth, not the prompt format.

6.3. Tool API

For completeness, we briefly summarize the tool interfaces (including arguments, constraints, and defaults) used during both stages.

- **`image_zoom_in_tool`**: Takes a 4-number array `bbox_2d = [x1, y1, x2, y2]` in pixel coordinates (image width, height).[1] Optionally accepts a textual `label`. The optional `target_image` index selects which buffered image to crop: negative values (`-1`, `-2`, ...) denote the last N processed images, positive values index the image lineage (`0` = original, `1` = after first tool call, etc.), and the default is `-1` (last processed image).
- **`image_rotate_tool`**: Requires an integer `angle` in degrees; positive values rotate clockwise and negative values counterclockwise (e.g., `90`, `180`, `270`). Optionally takes a descriptive `label` and a `target_image` index with the same convention as above (default `-1`).
- **`image_flip_tool`**: Requires a `direction` string in {`horizontal`, `vertical`}, indicating left–right vs. top–bottom flips. Optionally accepts a `label` and `target_image` (again defaulting to `-1` for the last processed image).
- **`image_draw_horizontal_line_tool`**: Requires an integer `height_location` giving the image row (pixel) where the horizontal line should be drawn. Optional arguments include `color` (e.g., `"red"`, `"blue"`, `"green"`), `thickness` (pixels), `style` in {`solid`, `dashed`}, a textual `label`, and `target_image` (default `-1`).
- **`image_draw_vertical_line_tool`**: Requires an integer `width_location` giving the image column (pixel) where the vertical line should be drawn, with the same optional `color`, `thickness`, `style`, `label`, and `target_image` (default `-1`) as the horizontal line tool.
- **`image_mark_points_tool`**: Takes `point_2d` as either a single `[width_location, height_location]` pair or an array of such pairs (all in pixel coordinates). Optional arguments include `color`, `size` (marker radius in pixels), `shape` (e.g., `circle`, `X`, `star`), `label` (single string or list of strings for multiple points), and `target_image` (default `-1`).

Across all tools, the prompts consistently remind the model that every coordinate—`bbox_2d`, `height_location`, `width_location`, and `point_2d`—must be expressed in the pixel space of the current rendered image, rather than in chart-axis units.

[1] Throughout, "pixel coordinates" mean column/row indices on the rendered image, not axis values.

Table 1. System Prompt for Zoom-in Task (Stage 1; abbreviated tool-argument text, see Tool API in Sec. 6 for full definitions).

```
You are a helpful assistant.

# Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type":"function","function":{"name":"image_zoom_in_tool","description":"Zoom in on a specific region...","
parameters":{"type":"object","properties":{"bbox_2d":{"type":"array","items":{"type":"number"},"minItems":4,"
maxItems":4,"description":"The bounding box of the region to zoom in..."},"label":{"type":"string","description
":"..."},"target_image":{"type":"integer","description":"...","default":-1}},"required":["bbox_2d"]}}
</tools>

# How to call a tool
Return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

**Example**:
<tool_call>
{"name": "image_zoom_in_tool", "arguments": {"bbox_2d": [50, 100, 200, 300], "label": "the object in the center", "
target_image": 1}}
</tool_call>
```

Table 2. System Prompt for Rotate/Flip Tasks (Stage 1; abbreviated tool-argument text, see Tool API in Sec. 6 for full definitions).

```
You are a helpful assistant.

# Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type":"function","function":{"name":"image_rotate_tool","description":"Rotate an image by a specified angle...","
parameters":{"type":"object","properties":{"angle":{"type":"integer","description":"Rotation angle in degrees
...","label":{"type":"string","description":"..."},"target_image":{"type":"integer","description":"...","default
":-1}},"required":["angle"]}}
{"type":"function","function":{"name":"image_flip_tool","description":"Flip an image horizontally or vertically
...","parameters":{"type":"object","properties":{"direction":{"type":"string","enum":["horizontal","vertical"],"
description":"Direction to flip..."},"label":{"type":"string","description":"..."},"target_image":{"type":"integer
","description":"...","default":-1}},"required":["direction"]}}
</tools>

# How to call a tool
Return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

**Examples**:
<tool_call>
{"name": "image_rotate_tool", "arguments": {"angle": 270, "label": "rotate image for better view", "target_image":
2}}
</tool_call>
<tool_call>
{"name": "image_flip_tool", "arguments": {"direction": "horizontal", "label": "flip image to correct orientation",
"target_image": 1}}
</tool_call>
```

Table 3. System Prompt for Read-Value and Compare-and-Count Tasks (Stage 1; abbreviated tool-argument text, see Tool API in Sec. 6 for full definitions).

```
You are a helpful assistant.

# Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type":"function","function":{"name":"image_mark_points_tool","description":"Mark specific points on an image
...","parameters":{"type":"object","properties":{"point_2d":{"type":"array","description":"Use pixel coordinates
..."},"color":{"type":"string","description":"..."},"size":{"type":"integer","description":"..."},"shape":{"type":"
string","enum":["X","star"],"description":"..."},"label":{"type":"string","description":"..."},"target_image":{"
type":"integer","description":"...","default":-1}},"required":["point_2d"]}}
{"type":"function","function":{"name":"image_draw_horizontal_line_tool","description":"Draw a horizontal line...","
parameters":{"type":"object","properties":{"height_location":{"type":"integer","description":"Image row index
..."},"color":{"type":"string","description":"..."},"thickness":{"type":"integer","description":"..."},"style":{"
type":"string","enum":["solid","dashed"],"description":"..."},"label":{"type":"string","description":"..."},"
target_image":{"type":"integer","description":"...","default":-1}},"required":["height_location"]}}
{"type":"function","function":{"name":"image_draw_vertical_line_tool","description":"Draw a vertical line...","
parameters":{"type":"object","properties":{"width_location":{"type":"integer","description":"Image column index
..."},"color":{"type":"string","description":"..."},"thickness":{"type":"integer","description":"..."},"style":{"
type":"string","enum":["solid","dashed"],"description":"..."},"label":{"type":"string","description":"..."},"
target_image":{"type":"integer","description":"...","default":-1}},"required":["width_location"]}}
</tools>

# How to call a tool
Return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

**Examples**:
<tool_call>
{"name": "image_mark_points_tool", "arguments": {"point_2d": [[253, 375], [190, 220]], "color": "yellow", "size":
8, "shape": "star", "label": ["Point A", "Point B"], "target_image": -1}}
</tool_call>
<tool_call>
{"name": "image_draw_horizontal_line_tool", "arguments": {"height_location": 157, "color": "purple", "thickness":
3, "style": "dashed", "label": "horizontal guide at this height location", "target_image": 0}}
</tool_call>
<tool_call>
{"name": "image_draw_vertical_line_tool", "arguments": {"width_location": 209, "color": "blue", "thickness": 4, "
style": "solid", "label": "vertical guide at this width location", "target_image": 0}}
</tool_call>

Always remember: location values such as the `[width_location, height_location]` pairs in `point_2d` and the
separate `width_location` / `height_location` arguments for line tools are pixel measurements from the rendered
image (width = columns, height = rows), not the chart-axis numbers. Match the screenshot pixels: if a chart point
is labelled (15,10) but appears at pixel column 157 and row 200, call the tools with `point_2d`: [157, 200], `
width_location`: 157, and `height_location`: 200.
```

Table 4. System Prompt for Stage 2 QA (abbreviated tool-argument text, see Tool API in Sec. 6 for full definitions).

```
You are a helpful assistant.

# Tools
You may call one or more functions to assist with the user query.
You are provided with function signatures within <tools></tools> XML tags:
<tools>
{"type":"function","function":{"name":"image_mark_points_tool","description":"Mark specific points on an image
...","parameters":{"type":"object","properties":{"point_2d":{"type":"array","description":"Use pixel coordinates
..."},"color":{"type":"string","description":"..."},"size":{"type":"integer","description":"..."},"shape":{"type":"
string","enum":["circle","X","star"],"description":"..."},"label":{"type":"string","description":"..."},"
target_image":{"type":"integer","description":"...","default":-1}},"required":["point_2d"]}}
{"type":"function","function":{"name":"image_zoom_in_tool","description":"Zoom in on a specific region...","
parameters":{"type":"object","properties":{"bbox_2d":{"type":"array","items":{"type":"number"},"minItems":4,"
maxItems":4,"description":"Bounding box as [x1, y1, x2, y2]..."},"label":{"type":"string","description":"..."},"
target_image":{"type":"integer","description":"...","default":-1}},"required":["bbox_2d"]}}
{"type":"function","function":{"name":"image_draw_horizontal_line_tool","description":"Draw a horizontal line...","
parameters":{"type":"object","properties":{"height_location":{"type":"integer","description":"Image row index
..."},"color":{"type":"string","description":"..."},"thickness":{"type":"integer","description":"..."},"style":{"
type":"string","enum":["solid","dashed"],"description":"..."},"label":{"type":"string","description":"..."},"
target_image":{"type":"integer","description":"...","default":-1}},"required":["height_location"]}}
{"type":"function","function":{"name":"image_draw_vertical_line_tool","description":"Draw a vertical line...","
parameters":{"type":"object","properties":{"width_location":{"type":"integer","description":"Image column index
..."},"color":{"type":"string","description":"..."},"thickness":{"type":"integer","description":"..."},"style":{"
type":"string","enum":["solid","dashed"],"description":"..."},"label":{"type":"string","description":"..."},"
target_image":{"type":"integer","description":"...","default":-1}},"required":["width_location"]}}
{"type":"function","function":{"name":"image_rotate_tool","description":"Rotate an image by a specified angle...","
parameters":{"type":"object","properties":{"angle":{"type":"integer","description":"Rotation angle in degrees
..."},"label":{"type":"string","description":"..."},"target_image":{"type":"integer","description":"...","default
":-1}},"required":["angle"]}}
{"type":"function","function":{"name":"image_flip_tool","description":"Flip an image horizontally or vertically
...","parameters":{"type":"object","properties":{"direction":{"type":"string","enum":["horizontal","vertical"],"
description":"Flip direction."},"label":{"type":"string","description":"..."},"target_image":{"type":"integer","
description":"...","default":-1}},"required":["direction"]}}
</tools>

# How to call a tool
Return a json object with function name and arguments within <tool_call></tool_call> XML tags:
<tool_call>
{"name": <function-name>, "arguments": <args-json-object>}
</tool_call>

**Examples**:
<tool_call>
{"name": "image_mark_points_tool", "arguments": {"point_2d": [[188, 114], [324, 200]], "color": "green", "size": 6,
 "shape": "X", "label": ["Point A", "Point B"], "target_image": 2}}
</tool_call>
<tool_call>
{"name": "image_zoom_in_tool", "arguments": {"bbox_2d": [120, 90, 360, 320], "label": "zoom to the bar labels", "
target_image": 0}}
</tool_call>
<tool_call>
{"name": "image_draw_horizontal_line_tool", "arguments": {"height_location": 205, "color": "yellow", "thickness":
3, "style": "dashed", "label": "horizontal guide for reading value", "target_image": 0}}
</tool_call>
<tool_call>
{"name": "image_draw_vertical_line_tool", "arguments": {"width_location": 178, "color": "purple", "thickness": 3, "
style": "solid", "label": "vertical guide for comparing x-values", "target_image": -1}}
</tool_call>
<tool_call>
{"name": "image_rotate_tool", "arguments": {"angle": -45, "label": "rotate image to upright orientation", "
target_image": 1}}
</tool_call>
<tool_call>
{"name": "image_flip_tool", "arguments": {"direction": "horizontal", "label": "flip image to correct orientation",
"target_image": 0}}
</tool_call>

Always remember: all coordinates (`bbox_2d`, `height_location`, `width_location`, `point_2d`) must use pixel
locations from the rendered image (width = columns, height = rows), not chart-axis values.
```