VTool-R1: VLMs Learn to Think with Images via Reinforcement Learning on Multimodal Tool Use

Anonymous ACL submission

Abstract

Reinforcement Learning Finetuning (RFT) has significantly advanced the reasoning capabilities of large language models (LLMs) by enabling long chains of thought, self-correction, and effective tool use. While recent works attempt to extend RFT to vision-language models (VLMs), these efforts largely produce text-only reasoning conditioned on static image inputs, falling short of true multimodal reasoning in the response. In contrast, test-time methods like Visual Sketchpad incorporate visual steps but lack training mechanisms.

We introduce VTool-R1, the first framework that trains VLMs to generate multimodal chains of thought by interleaving text and intermediate visual reasoning steps. VTool-R1 integrates Python-based visual editing tools into the RFT process, enabling VLMs to learn when and how to generate visual reasoning steps that benefit final reasoning. Trained with outcome-based rewards tied to task accuracy, our approach elicits strategic visual tool use for reasoning without relying on process-based supervision. Experiments on structured visual question answering over charts and tables show that VTool-R1 enhances reasoning performance by teaching VLMs to "think with images" and generate multimodal chain of thoughts with tools.

1 Introduction

Recent large language models (LLMs), notably DeepSeekR1 (DeepSeek-AI, 2025) and the GPT4o series (OpenAI, 2024), have demonstrated remarkable capabilities in text-based reasoning. Central to recent LLM reasoning advancements is Reinforcement Learning Finetuning (RFT), which enables these models to generate long chains of thought, engage in self-correction and verification for complex reasoning tasks (Kumar et al., 2025; Zeng et al., 2025). RFT has also shown promising results in effectively integrating external tool use, such as search engines (Jin et al., 2025; Chen et al., 2025b) and code interpreters (Feng et al., 2025), into the reasoning process of LLMs: Through RFT training, LLMs can effectively learn when to invoke the tools, how to use the tools, and more importantly, how to reason with the text output from the tools. Moreover, effective tool use essentially enriches LLMs with extra knowledge and specialized capabilities beyond what is embedded in their parameters, expanding model capabilities beyond narrow domains, such as math and programming.

Despite the rapid progress in LLM reasoning with text brought by RFT, there has not yet been a well-recognized breakthrough in improving multimodal reasoning capabilities of vision-language models (VLMs). Modern VLMs (Deitke et al., 2024; Bai et al., 2025; Grattafiori et al., 2024) typically consist of a strongly language-aligned image encoder like CLIP (Radford et al., 2021) that maps visual inputs into feature space and a connector module further projects these visual features into the token space of a decoder-only LLM. On top of these architectures, post-training strategies like visual instruction tuning (Xu et al., 2024) improve VLMs' ability to follow textual instructions. Building on these visual-language alignment efforts in VLM design, concurrent works (Zhou et al., 2025; Chen et al., 2025a; Zhang et al., 2025; Huang et al., 2025; Liu et al., 2025; Deng et al., 2025; Wang et al., 2025) attempt to replicate the success of LLMs with RFT in the VLM domain to enhance multimodal reasoning. These efforts demonstrate that RFT can be adapted to VLMs to improve their multimodal reasoning capabilities. **But which aspects of VLMs' reasoning does RFT actually improve?** A closer look reveals that these concurrent works do not enable VLMs to generate truly multimodal reasoning chains: This is because they prepend image features as fixed tokens to the input textual prompt, and train the model to generate purely textual reasoning responses conditioned on the input prompt. All generated reasoning remains

text-only, and no step-by-step "thinking with images" occurs during response generation. In contrast, emerging inference-time frameworks like Visual Sketchpad (Hu et al., 2024) and Refocus (Fu et al., 2025) demonstrate that incorporating intermediate visual reasoning steps during inference can improve performance beyond purely textual reasoning. However, these methods rely on highly capable models such as GPT-4o to produce meaningful visual steps, and they do not involve any training for reasoning with visual thoughts.

In this paper, we present the first work that directly enables VLMs to learn to think in images and texts and to be trained to generate multimodal chains of thoughts. We build our framework **VTool-R1**, where VLMs learn to generate intermediate visual reasoning steps through interaction with external image editing tools implemented in Python code. These visual reasoning steps are interleaved with textual chains of thought, resulting in multimodal reasoning in the model response. Following the design of DeepSeekR1 (DeepSeek-AI, 2025), VTool-R1 leverages RFT with outcome-based rewards tied to final task accuracy, while avoiding process-based rewards to mitigate reward hacking. VTool-R1 successfully trains VLMs to learn when and how to generate visual reasoning steps via external tool use and conduct chain of thought reasoning also based on the generated intermediate visual thoughts.

We demonstrate the effectiveness of VTool-R1 on challenging structured image understanding tasks, focusing on visual question answering (VQA) over tables and charts. Our experiments use a well-curated dataset and a visual editing tool set from Refocus (Fu et al., 2025). This toolset, which is directly callable by the VLM, enables selective attention on the image—simulating how humans process visual information through attention and reasoning before forming final conclusions. Through RFT, the model learns to use these tools strategically to guide its multimodal chain of thought and enhance reasoning performance.

Our key contributions can be summarized as follows:

- To the best of our knowledge, our work is the first work that successfully enables VLMs to learn to integrate intermediate vision reasoning steps into text-based chain of thoughts in the generated response (i.e. thinking with images and texts).
- We present VTool-R1, a novel RFT framework that supports VLM multimodal reasoning with visual editing tool use. We demonstrate that RFT with outcome based reward design can unexpectedly elicit visual reasoning steps for final reasoning accuracy.
- We conduct extensive experiments on challenging structured image understanding tasks. With VTool-R1,

2 Related Works

2.1 Visual Chain of Thought Reasoning

Early works have demonstrated that incorporating visual intermediate steps—often generated via external tools or Python scripts—can benefit a wide range of visual question answering (VQA) tasks, without any training. Pioneering efforts such as ViperGPT (Surís et al., 2023) and Visual Programming without Training (Gupta and Kembhavi, 2023) utilize Python-based visual tools to manipulate images during inference. More recently, Visual Sketchpad (Hu et al., 2024) introduced a framework that equips multimodal language models with a sketchpad and drawing tools. The model is prompted to generate visual artifacts during inference and uses them for iterative planning and reasoning. While this approach successfully introduces visual information into the reasoning process, it operates solely at inference time and cannot be trained or improved further. Refocus (Fu et al., 2025) takes a step forward by prompting the VLM to invoke visual editing tools for selective attention over images. These modified images are then used as inputs for further reasoning. However, Refocus does not train the model to reason with tools; instead, it relies on oracle-edited images generated by a commercially powerful model such as GPT-4o. Smaller models cannot gain the capability of such kind of tool use and reasoning.

2.2 LLM/VLM, Reinforcement Learning and Tool Use

Reinforcement learning (RL) was first introduced to LLM fine-tuning via RL from human feedback (RLHF) (Ouyang et al., 2022), which fits a reward model to human preferences, using Proximal Policy Optimization (PPO) (Schulman et al., 2017). While effective, PPO requires an actor model and involves multiple LLM optimizations in RL training. To make RL tuning easier, simpler alternatives such as Direct Preference Optimization (DPO)

(Rafailov et al., 2023), SimPO (Meng et al., 2024) have been proposed. These methods offer computational efficiency but often suffer from off-policy bias and may underperform full RL approaches (Pang et al., 2024). Group Relative Policy Optimization (GRPO) (Shao et al., 2024) mitigates these issues by foregoing the critic model and estimating baselines from grouped scores. Beyond preference tuning, RL has recently been used to enhance tool-augmented reasoning in LLMs, enabling models to learn when and how to invoke external tools such as calculators, code interpreters, and web search engines (Chen et al., 2025b; Feng et al., 2025; Jin et al., 2025). This line of work demonstrates that outcome-based RL rewards can effectively guide complex, multi-step reasoning involving external tools.

In the vision-language domain, similar ideas are beginning to emerge. Concurrent works adapt RL for VLMs to incentivize multimodal reasoning behaviors (Zhou et al., 2025; Chen et al., 2025a; Zhang et al., 2025; Huang et al., 2025; Liu et al., 2025; Deng et al., 2025; Wang et al., 2025), but they primarily train VLMs to only generate textual chains of thought from visual inputs. The problem of training VLMs to dynamically generate and reason over visual intermediate steps via external tools remains largely unexplored—motivating our proposed framework.

3 VTool-R1

VLM Preliminaries. A VLM policy can be denoted as π_θ parametrized with model weights θ. Given a text prompt sequence x and an image I, the model can generate a text response sequence y, sampled from the $\pi_\theta(I, x)$. Some VLMs support multiple image inputs; however, their capabilities in parsing and understanding multiple images vary significantly and are highly dependent on the training procedure (Wang et al., 2024). In our setting, we require VLMs capable of processing multiple images because intermediate visual steps will serve as input.

In the following sections, we present the detailed design of VTool-R1, covering inference and training parts. In the Section 3.1, we show that how pre-trained VLM can be prompted to use visual editing tools and generate integrate intermediate visual steps. We then go beyond inference in the Section 3.2: by extending the RFT objective, we train VLMs to use these tools and generate multimodal chains of thought during rollout by themselves. We also introduce an outcome-based reward formulation that encourages effective visual reasoning while avoiding the pitfalls of process-based reward hacking.

3.1 VLM Inference and Rollout with Visual Chain of Thoughts

Refocus (Fu et al., 2025) demonstrates that a sufficiently capable VLM, such as GPT-4o, can be prompted to generate Python code for invoking external tools and editing images, followed by reasoning over the modified visual input. Our inference and rollout template closely follows their prompting instructions. The full prompt is in the Appendix.

VLM Inference/Rollout Prompts

```
<System Prompt><Python Codes Templates>
# GOAL #: Based on the above tools, I want you to
reason about how to solve the # USER REQUEST # and
generate the actions step by step (each action is a
python function call) to solve the request. You may
need to use the tools above to process the images and
make decisions based on the visual outputs of the
previous code blocks. You should only use the tools
above, you should not use other functions or code that
will not be executed.
# REQUIREMENTS #:
...3. If you think you got the answer, use ANSWER:
<your answer> Please extract the final answer in FINAL
ANSWER: <final answer> and ends with TERMINATE.
...8. If you do not think you have enough information
to answer the question on the images returned by the
tools, you should directly answer the question based
on the original image...
9. Only one turn of action, ACTION 0, is allowed. You
must provide the answer after a maximum one ACTION
call.
In-context Examples:
<Thought 0><Action 0><Observation><Edited
Image><Thought 1><Answer>
# USER Bounding Box Info: x_values_bbox, storing x
values and coordinates. y_values_bbox, storing x
values and coordinates. The x values in the image are:
<x_values>. The y values in the image are: <y_values>.
# USER IMAGE stored in image_1, as PIL image.
```

As illustrated in the prompt box above, we provide system instructions and task-specific goals to guide the VLM in using visual tools for reasoning. The model is given the names and definitions of visual editing functions, along with detailed descriptions of their usage. Through in-context examples, the VLM is prompted to begin its reasoning in *Thought 0*, which outlines where to focus in the image. It then produces *Action 0*, which is either a "no action needed" statement or a Python-like pseudocode snippet that invokes an appropriate visual editing tool.

The tool call is executed externally in a Python environment to generate a modified image, which is then fed back into the model as additional input.

The VLM continues its reasoning over this generated intermediate visual step, forming a richer visual chain of thought that supports the final answer. As specified in the Requirement section of the prompt, the model is allowed to either respond directly or invoke a tool once to create an intermediate visual step for further reasoning. In this work, we focus on single-turn tool use—i.e., the model may call a tool at most once and reason over the edited image. Extending this to multi-turn tool use, where the model iteratively edits and reasons for multiple rounds, is left for the future.

This inference and rollout process is inherently iterative and cannot be completed in a single VLM call if the model chooses to use tools. Execution must pause for the external Python environment to run the generated code. Once the modified image is available, it has to be reintroduced into the same VLM instance as the second image input, rather than being inserted at the original location in the generated response sequence, as is common in prior tool-use approaches such as Search-R1 (Jin et al., 2025).

Formally, if the VLM policy π_θ decides to invoke a tool from an external visual editing toolset T, the inference involves two rounds of model execution. The first round samples an initial response containing tool calls, $y' \sim \pi_\theta(\cdot \mid I, x)$, where the input text prompt x includes tool descriptions. The tool calls are then executed in the Python environment as $I' = \mathsf{T}(y', I)$ to generate a modified image. In the second round, the VLM performs reasoning over both the original and edited images:

$$\begin{aligned} y \sim \pi_\theta(\cdot \mid I, x;\, \mathsf{T}) &= \pi_\theta(\cdot \mid I \oplus I', x) \\ &= \pi_\theta(\cdot \mid I \oplus \mathsf{T}(y', I), x) \end{aligned} \tag{1}$$

where $\bigoplus$ denotes the concatenation of the original image I and the updated image I' as dual image inputs to the model. If the VLM chooses not to invoke any tools and instead answers the question directly, the final answer is obtained in the first round without needing a second inference pass: $y \sim \pi_\theta(\cdot \mid I, x)$.

3.2 RFT VLM to Generate Visual Chain of Thoughts

VTool-R1 adopts RFT to train VLMs to explore flexible reasoning trajectories and learn to invoke visual editing tools effectively. Given the two-stage iterative inference structure, we explore multiple rollout strategies—optimizing either just the response y with final answers, or both the intermediate tool-invoking output y' and y jointly. We maintain the notation introduced at the end of Section 3.1.

In VTool-R1, we assume a reward model r_ϕ under the Bradley-Terry formulation, and consider the following RFT training objectives:

Optimize the final reasoning response y during RL rollout:

$$\begin{aligned} &\max_{\pi_\theta} \mathbb{E}_{[I,x]\sim\mathcal{D},\, y\sim\pi_\theta(\cdot|I,x;\mathsf{T})} \big[r_\phi(I, x, y)\big] \\ &- \beta\, \mathbb{D}_{\mathrm{KL}}\big[\, \pi_\theta(\cdot \mid I, x; \mathsf{T}) \,\|\, \pi_{\mathrm{ref}}(\cdot \mid I, x; \mathsf{T})\big]. \end{aligned} \tag{2}$$

where π_θ is the policy VLM parametrized with model weights θ. π_{ref} is the reference VLM policy. r_ϕ is the reward function. $\mathbb{D}_{\mathrm{KL}}$ is the KL-divergence measure. $\beta > 0$ is the KL penalty coefficient. The input $[I, x]$ denotes multimodal samples drawn from the dataset $\mathcal{D}$. The generated response in the rollout $y \sim \pi_\theta(\cdot \mid I, x; \mathsf{T}) = \pi_\theta(\cdot \mid I \bigoplus I', x) = \pi_\theta(\cdot \mid I \bigoplus \mathsf{T}(y', I), x)$, if the model chooses to use a tool; otherwise, when no tool is invoked, the response simplifies to $y \sim \pi_\theta(\cdot \mid I, x; \mathsf{T}) = \pi_\theta(\cdot \mid I, x)$.

Unlike prior RFT that simply relies on LLM/VLM policy to generate rollout during training (Ouyang et al., 2022), V-Tool-R1 explicitly incorporates the visual editing tool use from the toolset T in the rollout, and conditions the model on the edited image input. We refer readers to Section 3.1 for the formal definition and intuition of the iterative tool-use rollout policy, which mirrors the model inference pipeline. This iterative pipeline enables more effective step-by-step reasoning across both modalities: the model learns to modify the image using tools to support its reasoning before producing the final answer.

Note that we do not directly optimize the intermediate tool-invoking response y' in our RFT process, as our goal is to encourage the model to autonomously decide whether using a tool improves reasoning. This design supports a more end-to-end training objective.

Our training approach is built upon a well-established policy gradient method: Group Relative Policy Optimization (GRPO) (DeepSeek-AI, 2025; Shao et al., 2024), which offers improved stability and eliminates the need for a separate critic model. Unlike Proximal Policy Optimization (Schulman et al., 2017), which estimates advantages using a learned critic, GRPO estimates

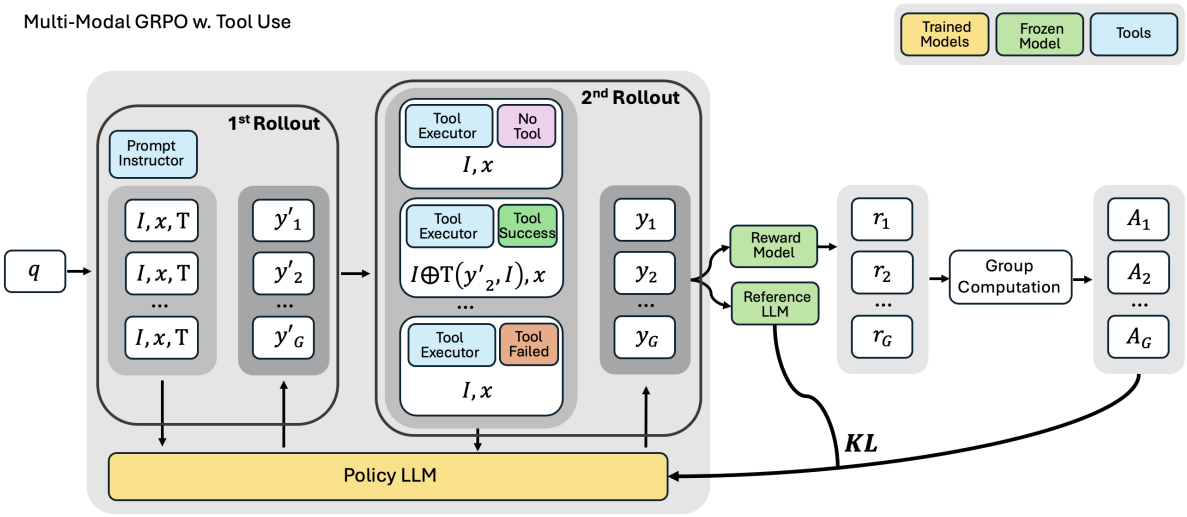

Figure 1: Multi-Modal GRPO w. Tool Use Training Pipeline, where the input q is a multimodal query

the baseline from a group of sampled responses and reduces training resources. Specifically, for each input $[I, x]$, GRPO samples a group of responses $\{y_i\}_{i=1}^{G} \sim \pi_{\text{old}}(\cdot|I, x; T)$ from the old policy model, and then optimize the current policy model by maximizing the following objective equation 3:

Here, ϵ and β are hyper-parameters, and $\hat{A}_{i,t} = \tilde{r}_i = \frac{r_i - \text{mean}(r)}{\text{std}(r)}$ denotes the normalized relative advantage computed within the group of sampled responses. This formulation avoids the need for critic model, while maintaining stable and reward-aligned policy updates by regularizing with the KL divergence between the updated policy π_θ and the reference policy π_{ref}. Notably, GRPO performs this regularization by directly adding the KL divergence term into the loss function.

We use inference template in the prompt box for training rollout as well. This template structures the model's output, think before actions and let the model decide whether we need a tool call, with the system instructions and requirements. We make the template highly formatted and listed clearly thoughts, actions, tool use function blocks and final answers. We also include few shot examples for better instruction and format following.

3.3 Reward Modeling

Following Deepseek-r1 (DeepSeek-AI, 2025), we adopt an outcome-based reward design that relies solely on the correctness of the model's final answer. For closed-ended tasks like factual QA, an exact string match works well. However, in our setting—structured visual understanding—the answers are more free-form and not easily judged by string match. To address this, we use a lightweight LLM-based judge to assess the match between the predicted answer and the ground truth. While not strictly rule-based, this serves as a pseudo rule-based reward appropriate for open-ended tasks such as ChartQA. We reward score of 1 when the judge thinks it is a match.

We also study process-based rewards that penalize incorrect tool use or reward successful invocations. However, this often results in reward hacking in RFT: models either avoid tools entirely when penalties are applied, or exploit success criteria by generating tool calls that superficially meet expectations without contributing to reasoning.

We do not use format-based rewards, as our models already learn to follow the structured format—Thoughts, Actions, Tools, and Final Answer—thanks to clear instruction templates. We leave further exploration of format rewards to future work, but find our current setup sufficient for reliable rollout behavior.

4 Experiment

4.1 Dataset

Following Refocus (Fu et al., 2025), we evaluate **VTool-R1** on structured image understanding tasks that are particularly suitable for assessing tool use. Our evaluation focuses on chart and table-based visual question answering (VQA), which poses significant challenges for early VLM works (Liu et al., 2023, 2022). To ensure fair comparison, we strictly

$$\mathcal{J}_{GRPO}(\theta) = \mathbb{E}_{[I,x]\sim\mathcal{D},\{y_i\}_{i=1}^{G}\sim\pi_{\text{old}}(\cdot|I,x;T)} \tag{3}$$
$$\left[\frac{1}{G}\sum_{i=1}^{G}\frac{1}{|y_i|}\sum_{t=1}^{|y_i|}\min\Big(r_{i,t}(\theta)\hat{A}_{i,t}, \text{clip}\big(r_{i,t}(\theta), 1-\epsilon, 1+\epsilon\big)\hat{A}_{i,t}\Big) - \beta\mathbb{D}_{KL}\left[\pi_\theta||\pi_{\text{ref}}\right]\right]$$

adhere to the dataset choices made in Refocus.

VWTQ. This dataset partition is derived from WikiTableQuestions (WTQ) (Pasupat and Liang, 2015), featuring 750 QA pairs fetched directly from Wikipedia HTML. The table images are styled from the stylesheet of Wikipedia as screenshots.

VWTQ_syn. To avoid potential data overlap with web-scraped training corpora in VLMs, Kim et al. (2024) generates 250 synthetic table images using a rendering engine that introduces random variations in layout, background color, font, and borders.

VTabFact. This data partition is fetched from TabFact dataset (Chen et al., 2019), that requires verifying whether a given statement is entailed by a table or refuted. (Kim et al., 2024) renders visual table images with pseudo-HTML from the table content and collects 250 pairs in total.

For all three table datasets, we randomly split 70% of the data for training, named and 30% for testing. No separate validation set is created due to the limited overall size of the data.

ChartQA – Horizontal Bar.. ChartQA (Masry et al., 2022) contains human-written questions grounded in real-world charts. The chart figures come from web-crawled results. We select 444 horizontal bar chart QA pairs from its test split, which involve logical and visual reasoning.

ChartQA – Vertical Bar.. An additional 382 vertical bar chart QA pairs are extracted from ChartQA, focusing on similar reasoning skills.

For Chart questions, we use all 444 + 382 questions as the test set. In addition, we use 14,344 training examples and 813 validation examples from the official training and validation splits. Validation accuracy is tracked throughout training.

4.2 Visual Editing Toolset T

While our long-term goal is to enable models to invoke arbitrary tools or APIs within a sandbox environment using outcome-based rewards, we begin by demonstrating VTool-R1's capabilities with a set of simple but effective visual editing tools. These tools are implemented in Python and help simulate visual attention by modifying table or chart images. We adopt the same tool set used in Refocus. In our experiments for tabular problems, we utilize a variety of tools as follows:

Highlight Column/Row: overlays a semi-transparent red on the selected columns/Rows.

Mask Column/Row: applies a white mask over irrelevant columns/rows.

Draw Column/Row: draws a solid red bounding box around selected columns/rows.

For charts, we apply analogous operations to highlight or mask individual bars, based on their positions along the x-axis or y-axis.

The model is instructed to call one or multiple tools at the same time, as many operations (e.g., drawing bounding boxes on multiple positions) can be performed in parallel and we involve at most one round of tool call in the experiments.

These tools leverage external libraries such as OpenCV to perform tasks like drawing bounding boxes and identifying maskable regions based on contours of bars or tables for selective attention in our tasks. Looking ahead, we envision integrating more advanced generative models as powerful tools that can execute more generalized visual modifications directly from language prompts.

4.3 Experiment Setup

We demonstrate the effectiveness of our reinforcement learning framework by training the state-of-the-art open-source VLMs—Qwen-VL 2.5 models (Bai et al., 2025) at 3B, 7B, and 32B scales. Training uses the open-source VeRL training infrastructure (Sheng et al., 2024). Training is conducted with the AdamW optimizer, using an initial learning rate of $1e{-}6$ and a weight decay of $1e{-}2$. Due to the large image sizes and long prompt sequences (up to 16,384 tokens), we set the micro-batch size to 2. The 3B/7B models are trained on 8/16 H100 GPUs; the 32B models are trained on 8 H200 GPUs. We standardize decoding across rollout and evaluation with temperature 1.0 and bf16 precision.

4.4 Baseline Models

We report several baselines to contextualize VTool-R1's performance in Table 1:

GPT-4o: Used as an upper bound across all benchmarks. This is a powerful commercial model that

Qwen2.5-VL (Bai et al., 2025)	3B			7B			32B			GPT-4o	
	Pure Run	Tool Use	VTool-R1	Pure Run	Tool Use	VTool-R1	Pure Run	Tool Use	VTool-R1	Pure Run	Tool Use
ChartQA (Masry et al., 2022)	51.8	24.6	64.0	76.2	53.4	73.1	88.0	85.0	86.7	82.9	80.5
TableVQA (Kim et al., 2024)	41.3	24.3	57.9	64.7	41.1	60.2	86.2	76.0	79.3	75.7	77.0

Table 1: Main Results of VTool-R1 and Baselines in Accuracy

has already shown remarkable tool use capability for reasoning in Refocus (Fu et al., 2025).

Qwen2.5-VL (3B / 7B / 32B): Included without any RFT in two configurations:

Prompted Tool-Use Inference: The model is prompted to use tools following our rollout template. However, prior to RFT, these models struggle to follow tool-use instructions. For fairness, when a tool call fails, we append a prompt indicating the failure and ask the model to regenerate its answer.

Direct Inference (No Tools): Only the image and question are provided, with no tool-related prompts. Surprisingly, this setting of open-source model yields strong results across datasets—often outperforming even GPT-4o—especially for larger models like 32B.

These findings about high pure run accuracy suggest that Qwen2.5-VL 3B and 7B may have been post-trained on VQA-style tasks or distilled from larger models, enabling strong direct-answering performance, but lacking the general-purpose tool-use capabilities seen in models like GPT-4o.

4.5 Main RFT Results and Findings

Qualitative Tool-use Example. Figure 2 presents a qualitative example where the VTool-R1 3B model successfully integrates intermediate visual steps through tool use as part of the reasoning process, ultimately arriving at the correct answer.

RFT makes Better Tool Use for Reasoning. When comparing VTool R1 Model reasoning accuracy with non-trained baselines in Table 1, we observe a significant improvement in tool use capability. After training, models learn to reason correctly with multimodal tool use, guided solely by outcome-based rewards. Remarkably, the 3B model, which initially failed to generate meaningful tool use (thoughts, actions, or tool calls), learns to use tools effectively to generate intermediate reasoning. This indicates that VTool-R1 not only improves final answer accuracy, but also enables models to internalize structured reasoning patterns. In several cases, VTool-R1 even outperforms the direct inference baseline.

Better Tool Use, or Not? It's Not Monotonic. Our goal goes beyond improving accuracy—we aim to teach the model when and how to use tools in a way that meaningfully supports reasoning in the RL training. As shown in Figure 3, VTool-R1 enables models to make nuanced, context-aware tool-use decisions. Interestingly, tool call frequency and success rate do not increase monotonically when the training proceeds and accuracy goes up. Instead, we observe fluctuations: models tend to overuse tools early in training due to prompt instruction exposure but later learn to invoke them more selectively. The 3B model becomes more cautious with tool use over time, leading to higher reasoning accuracy. Crucially, the model also learns when tools are unnecessary and confidently proceeds with direct reasoning. The 32B model (training curve shown in the Appendix) exhibits a higher overall tool use rate but similarly shows periods of decline, reflecting adaptive behavior. This adaptive tool-use behavior for reasoning is a key outcome of our RFT strategy. While the exact trends vary between table and chart tasks, the overall pattern remains consistent.

More Successful Tool Use or Not? Figure 3 also presents the tool call success rates of the most representative 3B model during RFT training on both chart and table tasks. It is important to note that we cannot evaluate tool call correctness with full precision and recall, as no oracle verifier is available. Instead, we rely on a proxy metric to approximate success: A tool call is considered successful if the python commands executed did not raise any exceptions inside a sandbox environment with the given functions, and a valid pillow image is returned from the execution through passing the processed image into the *display* function. According to this metric, the success rate of tool use steadily increases on table tasks, while for charts it fluctuates throughout training. We would like to highlight the need for future work to incorporate human-annotated oracle verifier to more accurately evaluate tool-use success.

Training Dynamics. Overall, the model's accuracy steadily improves throughout training, with minor fluctuations. Performance gradually converges and stabilizes around the final accuracy within approximately 50 training steps. The saturated step number

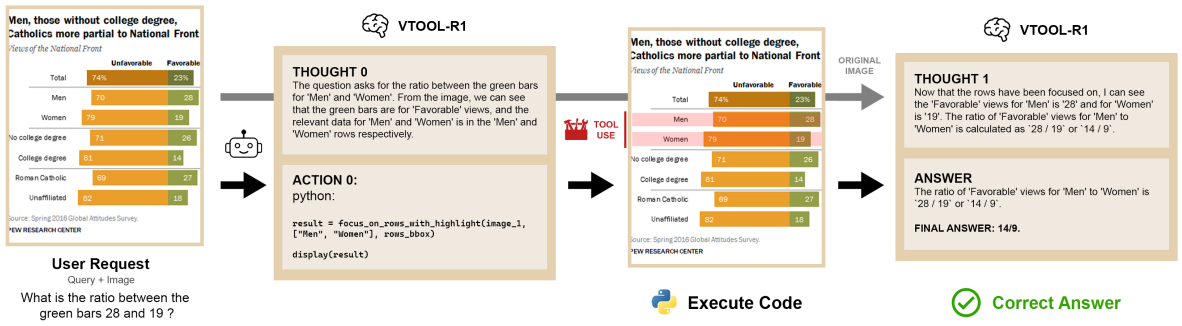

Figure 2: Qualitative Example from VTool-R1 (3B): The Model Successfully Integrates Intermediate Visual Steps.

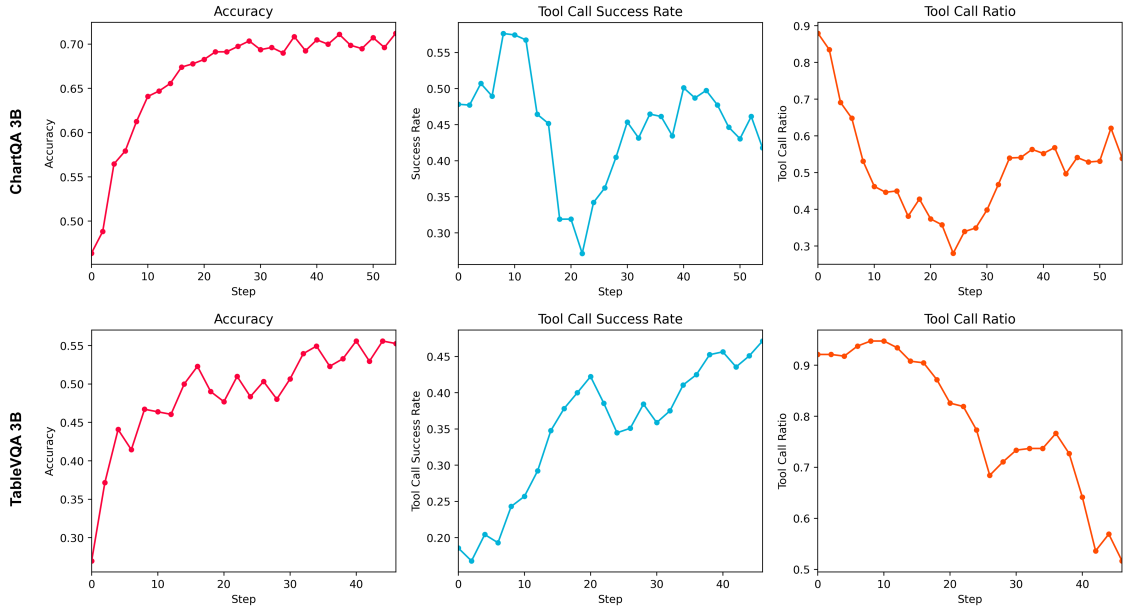

Figure 3: Multi-Modal GRPO w. Tool Use Training Dynamics, for 3B models

varies depending on the training configuration.

Reward Design. We also explore alternative reward settings beyond the standard 0/1 outcome-based reward. When applying process-based rewards—such as penalizing failed tool calls—we observe that the model quickly learns to avoid tool use entirely, driving the tool usage rate to zero. Conversely, when we add extra reward for successful tool use that leads to a correct answer, the model begins to exploit the verifier and hack to triggering a "success" signal. This holds even under stricter verifier criteria. These findings support our claims that outcome-based rewards tied solely to final task correctness serve as the most reliable and robust reward design for VTool-R1.

5 Conclusion

VTool-R1 demonstrates that RFT can effectively teach VLMs to reason multimodally by interleaving textual and visual steps. By integrating visual editing tools into the RL training loop and optimizing for outcome-based rewards, VTool-R1 enables models to learn when and how to use tools to support their reasoning—without requiring process-level supervision. Our experiments on structured visual question answering show that VTool-R1 not only improves final task accuracy but also equips models with the ability to generate coherent, multimodal chains of thought.

Broader Impacts VTool-R1 is the first framework to show that RFT can train VLMs to integrate visual reasoning steps by invoking visual editing tools and generating intermediate visual states to support their own reasoning goals. This opens up a novel and promising direction for multimodal AI, enabling models to reason more effectively across modalities and potentially unlocking fundamentally new capabilities that go beyond what is encoded in model parameters, especially for more tasks.

6 Limitations

VTool-R1 holds strong potential for scaling to a broader range of toolsets and generalizing to more diverse datasets. However, in this work, we focus on a straightforward task—selective attention in structured image understanding—as a starting point. We expect future extensions of the VTool-R1 framework to support the execution of more complex and diverse tools. While our current framework is limited to only a single round of tool invocation, we envision future extensions that enable multi-turn tool usage.

References

Shuai Bai, Keqin Chen, Xuejing Liu, Jialin Wang, Wenbin Ge, Sibo Song, and et al. 2025. Qwen2.5-vl technical report. *Preprint*, arXiv:2502.13923.

Liang Chen, Lei Li, Haozhe Zhao, Yifan Song, and Vinci. 2025a. R1-v: Reinforcing super generalization ability in vision-language models with less than $3. `https://github.com/Deep-Agent/R1-V`. Accessed: 2025-02-02.

Mingyang Chen, Tianpeng Li, Haoze Sun, Yijie Zhou, Chenzheng Zhu, Haofen Wang, Jeff Z. Pan, Wen Zhang, Huajun Chen, Fan Yang, Zenan Zhou, and Weipeng Chen. 2025b. Research: Learning to reason with search for llms via reinforcement learning. *Preprint*, arXiv:2503.19470.

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. 2019. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164*.

DeepSeek-AI. 2025. Deepseek-r1: Incentivizing reasoning capability in llms via reinforcement learning. *Preprint*, arXiv:2501.12948.

Matt Deitke, Christopher Clark, Sangho Lee, Rohun Tripathi, Yue Yang, Jae Sung Park, and et al. 2024. Molmo and pixmo: Open weights and open data for state-of-the-art vision-language models. *Preprint*, arXiv:2409.17146.

Yihe Deng, Hritik Bansal, Fan Yin, Nanyun Peng, Wei Wang, and Kai-Wei Chang. 2025. Openvlthinker: An early exploration to complex vision-language reasoning via iterative self-improvement. *Preprint*, arXiv:2503.17352.

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

Xingyu Fu, Minqian Liu, Zhengyuan Yang, John Corring, Yijuan Lu, Jianwei Yang, Dan Roth, Dinei Florencio, and Cha Zhang. 2025. Refocus: Visual editing as a chain of thought for structured image understanding. *arXiv preprint arXiv:2501.05452*.

Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, and et al. 2024. The llama 3 herd of models. *Preprint*, arXiv:2407.21783.

Tanmay Gupta and Aniruddha Kembhavi. 2023. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14953–14962.

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. 2024. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *arXiv preprint arXiv:2406.09403*.

Wenxuan Huang, Bohan Jia, Zijie Zhai, Shaosheng Cao, Zheyu Ye, Fei Zhao, Zhe Xu, Yao Hu, and Shaohui Lin. 2025. Vision-r1: Incentivizing reasoning capability in multimodal large language models. *Preprint*, arXiv:2503.06749.

Bowen Jin, Hansi Zeng, Zhenrui Yue, Jinsung Yoon, Sercan Arik, Dong Wang, Hamed Zamani, and Jiawei Han. 2025. Search-r1: Training llms to reason and leverage search engines with reinforcement learning. *Preprint*, arXiv:2503.09516.

Yoonsik Kim, Moonbin Yim, and Ka Yeon Song. 2024. Tablevqa-bench: A visual question answering benchmark on multiple table domains. *arXiv preprint arXiv:2404.19205*.

Aviral Kumar, Vincent Zhuang, Rishabh Agarwal, Yi Su, John D Co-Reyes, Avi Singh, Kate Baumli, Shariq Iqbal, Colton Bishop, Rebecca Roelofs, Lei M Zhang, Kay McKinney, Disha Shrivastava, Cosmin Paduraru, George Tucker, Doina Precup, Feryal Behbahani, and Aleksandra Faust. 2025. Training language models to self-correct via reinforcement learning. In *Proceedings of the 12th International Conference on Learning Representations (ICLR)*.

Fangyu Liu, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. 2023. Deplot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the 61st Annual Meeting of the Association for Computational Linguistics*.

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Martin Eisenschlos. 2022. Matcha: Enhancing visual language pretraining with math reasoning and chart derendering. *arXiv preprint arXiv:2212.09662*.

Yuqi Liu, Bohao Peng, Zhisheng Zhong, Zihao Yue, Fanbin Lu, Bei Yu, and Jiaya Jia. 2025. Seg-zero: Reasoning-chain guided segmentation via cognitive reinforcement. *Preprint*, arXiv:2503.06520.

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. 2022. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244*.

Yu Meng, Mengzhou Xia, and Danqi Chen. 2024. Simpo: Simple preference optimization with a reference-free reward. *Advances in Neural Information Processing Systems*, 37:124198–124235.

OpenAI. 2024. Openai o1 system card. *Preprint*, arXiv:2412.16720.

Long Ouyang, Jeff Wu, Xu Jiang, Diogo Almeida, Carroll L. Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, Maddie Simens, Amanda Askell, Peter Welinder, Paul Christiano, Jan Leike, and Ryan Lowe. 2022. Training language models to follow instructions with human feedback. *Preprint*, arXiv:2203.02155.

Richard Yuanzhe Pang, Weizhe Yuan, He He, Kyunghyun Cho, Sainbayar Sukhbaatar, and Jason Weston. 2024. Iterative reasoning preference optimization. *Advances in Neural Information Processing Systems*, 37:116617–116637.

Panupong Pasupat and Percy Liang. 2015. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*.

Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, Gretchen Krueger, and Ilya Sutskever. 2021. Learning transferable visual models from natural language supervision. *Preprint*, arXiv:2103.00020.

Rafael Rafailov, Archit Sharma, Eric Mitchell, Christopher D Manning, Stefano Ermon, and Chelsea Finn. 2023. Direct preference optimization: Your language model is secretly a reward model. *Advances in Neural Information Processing Systems*, 36:53728–53741.

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

Zhihong Shao, Peiyi Wang, Qihao Zhu, Runxin Xu, Junxiao Song, Xiao Bi, Haowei Zhang, Mingchuan Zhang, Y. K. Li, Y. Wu, and Daya Guo. 2024. Deepseekmath: Pushing the limits of mathematical reasoning in open language models. *Preprint*, arXiv:2402.03300.

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

Dídac Surís, Sachit Menon, and Carl Vondrick. 2023. Vipergpt: Visual inference via python execution for reasoning. *Proceedings of IEEE International Conference on Computer Vision (ICCV)*.

Fei Wang, Xingyu Fu, James Y Huang, Zekun Li, Qin Liu, Xiaogeng Liu, Mingyu Derek Ma, Nan Xu, Wenxuan Zhou, Kai Zhang, et al. 2024. Muirbench: A comprehensive benchmark for robust multi-image understanding. *arXiv preprint arXiv:2406.09411*.

Haozhe Wang, Chao Qu, Zuming Huang, Wei Chu, Fangzhen Lin, and Wenhu Chen. 2025. Vl-rethinker: Incentivizing self-reflection of vision-language models with reinforcement learning. *Preprint*, arXiv:2504.08837.

Guowei Xu, Peng Jin, Hao Li, Yibing Song, Lichao Sun, and Li Yuan. 2024. Llava-cot: Let vision language models reason step-by-step. *Preprint*, arXiv:2411.10440.

Weihao Zeng, Yuzhen Huang, Qian Liu, Wei Liu, Keqing He, Zejun Ma, and Junxian He. 2025. Simplerl-zoo: Investigating and taming zero reinforcement learning for open base models in the wild. *Preprint*, arXiv:2503.18892.

Jingyi Zhang, Jiaxing Huang, Huanjin Yao, Shunyu Liu, Xikun Zhang, Shijian Lu, and Dacheng Tao. 2025. R1-vl: Learning to reason with multimodal large language models via step-wise group relative policy optimization. *Preprint*, arXiv:2503.12937.

Hengguang Zhou, Xirui Li, Ruochen Wang, Minhao Cheng, Tianyi Zhou, and Cho-Jui Hsieh. 2025. R1-zero's "aha moment" in visual reasoning on a 2b non-sft model. *Preprint*, arXiv:2503.05132.

A Appendix

LLM Use: We use LLM for grammar checks.

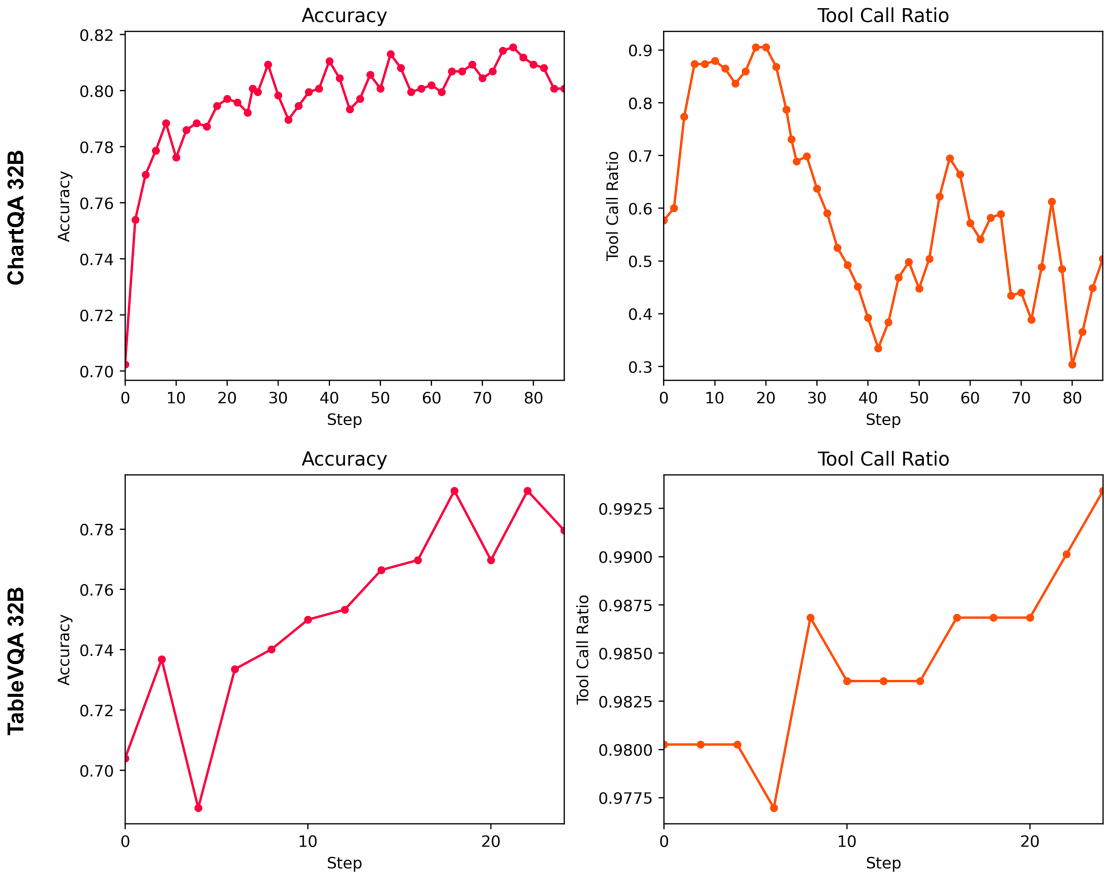

Figure 4: Multi-Modal GRPO w. Tool Use Training Dynamics, for 32B models

Prompts

Here are some tools that can help you. All are python codes. They are in tools.py and will be imported for you. You will be given a table figure: image_1 and a question. Notice that you, as an AI assistant, are not good at answering questions when there are too many unnecessary and irrelevant information. You should determine which are the relevant columns to the question, and specify them in a python list. You should use the given column headers. You should also determine which are the relevant rows to the question, and specify them in a python list. You should use the given row headers. You could select the tools to focus on some columns / rows, or mask out some columns / rows. Use whichever tool you think is more appropriate. Below are the tools in tools.py:

```python
def focus_on_columns_with_highlight(image, columns_to_focus_on,
    all_columns_bounding_boxes):
    \"\"\"
    This function is useful when you want to focus on some
    specific columns of the image.
```

```
    It does this by adding light transparent red highlight to
   the columns that need to be focused on.
    For example, you can focus on the columns in a table that
   are relevant to your analysis.
    Return the drawed image.

    Args:
        image (PIL.Image.Image): the input image
        columns_to_mask (List[str]): a list of column names to
   focus on.
        all_columns_bounding_boxes (Dict[Dict]]): a dictionary
   of bounding boxes for all columns in the image. key is column
    name and value is the bounding box of that column. Each
   bounding box is in the format {'x1': x1, 'y1': y1, 'x2': x2,
   'y2': y2}.

    Returns:
        image_with_focused_columns (PIL.Image.Image): the image
   with specified columns focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_columns =
   focus_on_columns_with_highlight(image, ["Year", "Name"], {"
   Year": {'x1': 0.1, 'y1': 0.1, 'x2': 0.3, 'y2': 0.9}, "Team":
   {'x1': 0.4, 'y1': 0.1, 'x2': 0.6, 'y2': 0.9}, "Name": {'x1':
   0.7, 'y1': 0.1, 'x2': 0.9, 'y2': 0.9}})
        display(image_with_focused_columns)
    \"\"\"

def focus_on_rows_with_highlight(image, rows_to_focus_on,
   all_rows_bounding_boxes):
    \"\"\"
    This function is useful when you want to focus on some
   specific rows of the image.
    It does this by adding light transparent red highlight to
   the rows that need to be focused on.
    For example, you can focus on the rows in a table that are
   relevant to your analysis.
    Return the drawed image.

    Args:
        image (PIL.Image.Image): the input image
        rows_to_focus_on (List[str]): a list of row headers to
   focus on.
        all_rows_bounding_boxes (Dict[Dict]): a dictionary of
   bounding boxes for all rows in the image. key is row header
   and value is the bounding box of that row. Each bounding box
   is in the format {'x1': x1, 'y1': y1, 'x2': x2, 'y2': y2}.
```

```
    Returns:
        image_with_focused_rows (PIL.Image.Image): the image
   with specified rows focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_rows = focus_on_rows_with_highlight(
   image, ["1972"], ["Year": {'x1': 0.1, 'y1': 0.1, 'x2': 0.9, '
   y2': 0.15}, "1969": {'x1': 0.1, 'y1': 0.2, 'x2': 0.9, 'y2':
   0.5}, "1972": {'x1': 0.1, 'y1': 0.6, 'x2': 0.9, 'y2': 0.9}])
        display(image_with_focused_rows)
    \"\"\"

def focus_on_columns_with_mask(image, columns_to_focus_on,
   all_columns_bounding_boxes):
    \"\"\"
    This function is useful when you want to focus on some
   specific columns of the image.
    It does this by masking out the columns that are not needed.
    For example, you can focus on the columns in a table that
   are relevant to your analysis and ignore the rest.
    Return the masked image.

    Args:
        image (PIL.Image.Image): the input image
        columns_to_mask (List[str]): a list of column names to
   focus on.
        all_columns_bounding_boxes (Dict[Dict]]): a dictionary
   of bounding boxes for all columns in the image. key is column
    name and value is the bounding box of that column. Each
   bounding box is in the format {'x1': x1, 'y1': y1, 'x2': x2,
   'y2': y2}.

    Returns:
        image_with_focused_columns (PIL.Image.Image): the image
   with specified columns focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_columns = focus_on_columns(image, ["
   Year", "Name"], {"Year": {'x1': 0.1, 'y1': 0.1, 'x2': 0.3, '
   y2': 0.9}, "Team": {'x1': 0.4, 'y1': 0.1, 'x2': 0.6, 'y2':
   0.9}, "Name": {'x1': 0.7, 'y1': 0.1, 'x2': 0.9, 'y2': 0.9}})
        display(image_with_focused_columns)
    \"\"\"

def focus_on_rows_with_mask(image, rows_to_focus_on,
   all_rows_bounding_boxes):
    \"\"\"
```

```
    This function is useful when you want to focus on some
   specific rows of the image.
    It does this by masking out the rows that are not needed.
    For example, you can focus on the rows in a table that are
   relevant to your analysis and ignore the rest.
    Return the masked image.

    Args:
        image (PIL.Image.Image): the input image
        rows_to_focus_on (List[str]): a list of row headers to
   focus on.
        all_rows_bounding_boxes (Dict[Dict]): a dictionary of
   bounding boxes for all rows in the image. key is row header
   and value is the bounding box of that row. Each bounding box
   is in the format {'x1': x1, 'y1': y1, 'x2': x2, 'y2': y2}.

    Returns:
        image_with_focused_rows (PIL.Image.Image): the image
   with specified rows focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_rows = focus_on_rows(image, ["1972"],
    ["Year": {'x1': 0.1, 'y1': 0.1, 'x2': 0.9, 'y2': 0.15},
   "1969": {'x1': 0.1, 'y1': 0.2, 'x2': 0.9, 'y2': 0.5}, "1972":
    {'x1': 0.1, 'y1': 0.6, 'x2': 0.9, 'y2': 0.9}])
        display(image_with_focused_rows)
    \"\"\"

def focus_on_columns_with_draw(image, columns_to_focus_on,
   all_columns_bounding_boxes):
    \"\"\"
    This function is useful when you want to focus on some
   specific columns of the image.
    It does this by drawing a red box around the columns that
   need to be focused on.
    For example, you can focus on the columns in a table that
   are relevant to your analysis.
    Return the drawed image.

    Args:
        image (PIL.Image.Image): the input image
        columns_to_mask (List[str]): a list of column names to
   focus on.
        all_columns_bounding_boxes (Dict[Dict]]): a dictionary
   of bounding boxes for all columns in the image. key is column
    name and value is the bounding box of that column. Each
   bounding box is in the format {'x1': x1, 'y1': y1, 'x2': x2,
   'y2': y2}.
```

```
    Returns:
        image_with_focused_columns (PIL.Image.Image): the image
    with specified columns focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_columns = focus_on_columns(image, ["
    Year", "Name"], {"Year": {'x1': 0.1, 'y1': 0.1, 'x2': 0.3, '
    y2': 0.9}, "Team": {'x1': 0.4, 'y1': 0.1, 'x2': 0.6, 'y2':
    0.9}, "Name": {'x1': 0.7, 'y1': 0.1, 'x2': 0.9, 'y2': 0.9}})
        display(image_with_focused_columns)
    \"\"\"

def focus_on_rows_with_draw(image, rows_to_focus_on,
    all_rows_bounding_boxes):
    \"\"\"
    This function is useful when you want to focus on some
    specific rows of the image.
    It does this by drawing a red box around the rows that need
    to be focused on.
    For example, you can focus on the rows in a table that are
    relevant to your analysis.
    Return the drawed image.

    Args:
        image (PIL.Image.Image): the input image
        rows_to_focus_on (List[str]): a list of row headers to
    focus on.
        all_rows_bounding_boxes (Dict[Dict]): a dictionary of
    bounding boxes for all rows in the image. key is row header
    and value is the bounding box of that row. Each bounding box
    is in the format {'x1': x1, 'y1': y1, 'x2': x2, 'y2': y2}.

    Returns:
        image_with_focused_rows (PIL.Image.Image): the image
    with specified rows focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_rows =
    focus_on_columns_with_highlight(image, ["1972"], ["Year": {'
    x1': 0.1, 'y1': 0.1, 'x2': 0.9, 'y2': 0.15}, "1969": {'x1':
    0.1, 'y1': 0.2, 'x2': 0.9, 'y2': 0.5}, "1972": {'x1': 0.1, '
    y1': 0.6, 'x2': 0.9, 'y2': 0.9}])
        display(image_with_focused_rows)
    \"\"\"
```

GOAL #: Based on the above tools, I want you to reason about how to solve the # USER REQUEST # and generate the actions step by step (each action is a python function call) to solve the request. You may need to use the tools

above to process the images and make decisions based on the visual outputs of the previous code blocks. You should only use the tools above, you should not use other functions or code which will not be executed.
REQUIREMENTS #:
1. The generated actions can resolve the given user request # USER REQUEST # perfectly. The user request is reasonable and can be solved. Try your best to solve the request.
2. The arguments of a tool must be the same format specified in # TOOL LIST #;
3. If you think you got the answer, use ANSWER: <your answer> Please extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE.
4. All images in the initial user request are stored in PIL Image objects named image_1, image_2, ..., image_n. You can use these images in your code blocks. Use display() function to show the image in the notebook for you too see.
5. Use as few tools as possible. Only use the tools for the use cases written in the tool description. You can use multiple tools in a single action.
6. If you have multiple answers, please separate them with || marks. For example, if the answer is 'Alice' and 'Bob', you should write 'Alice||Bob'.
7. When you focus on columns in the image, most like you need to look at multiple columns instead of a single one.
8. If you do not think you have enough information to answer the question on the images returned by the tools, you should directly answer the question based on the original image.
Below are some examples of how to use the tools to solve the user requests. You can refer to them for help. You can also refer to the tool descriptions for more information.
9. Only one turn of action, ACTION 0, is allowed. You must provide the answer after maximum one ACTION call.

EXAMPLE: Simple question that does not require any tool
USER REQUEST #: <A image here> What is the title of this table?
USER Bounding Box Info: columns_bbox, where keys are column headers and values are column bounding boxes. rows_bbox, where keys are row headers and values are row bounding boxes. The columns in the image are: ["Grade", "Mentor", "Salary"]. The rows in the image start with: ["Grade", "A", "B", "C"].
USER IMAGE stored in image_1, as PIL image.
RESULT #:
THOUGHT 0: The question does not require any tool. I can see the title of the table is "Customer Information".
ACTION 0: No action needed.
ANSWER: The title of the table is "Customer Information". FINAL ANSWER: Customer Information. TERMINATE

EXAMPLE: Focus on specific columns in the image
USER REQUEST #: <A image here> Who had the same game version as John Roth?
USER Bounding Box Info: columns_bbox, where keys are column headers and values are column bounding boxes. rows_bbox, where keys are row headers and values are row bounding boxes. The columns in the image are: ['Manager Name', 'Game Version', 'Game Score']. The rows in the image start with: ['Manager Name', 'John Roth', 'Alice Smith', 'Bob Johnson'].
USER IMAGE stored in image_1, as PIL image.

RESULT #:
THOUGHT 0: To identify who had the same game version as John Roth, I need to focus on the 'Game Version' column and the 'Manager Name' column. Also, I need to focus on all the rows so I do not need to focus on some specific rows.
ACTION 0:

```python
image_with_focused_columns = focus_on_columns_with_draw(image\_1
    , ["Game Version", "Manager Name"], columns\_bbox)
display(image_with_focused_columns)
```

OBSERVATION: Execution success. The output is as follows:
<the image outputs of the code is added as the second image>
THOUGHT 1: Now I can see the 'Game Version' column and the 'Manager Name' column more clearly. The game version of John Roth is 'v1.2'. Other people with the same game version are 'Alice Smith' and 'Bob Johnson'.
ANSWER: 'Alice Smith' and 'Bob Johnson' had the same game version as John Roth are. FINAL ANSWER: Alice Smith||Bob Johnson. TERMINATE

EXAMPLE: Focus on specific rows in the image
USER REQUEST #: <A image here> How many games did Josh win after 1996?
USER Bounding Box Info: columns_bbox, where keys are column names and values are column bounding boxes. rows_bbox, where keys are row headers and values are row bounding boxes. The columns in the image are: ["Rank", "Year", "Score", "Month"]. The rows in the image start with: ["Rank", "0", "1", "2", "3", "4", "5"].
USER IMAGE stored in image_1, as PIL image.
RESULT #:
THOUGHT 0: This table is about the games played by Josh that he won. I do not need to focus on any specific columns. I need to focus on the rows with the year after 1996. Three rows have year after 1996: one has year 1997 and this row starts with "3", one has year 1998 and this row starts with "4", and one has year 1999 and this row starts with "5". So I will focus on the rows with row starters "3", "4", and "5".
ACTION 0:

```python
image_with_focused_rows = focus_on_rows_with_highlight(image\_1,
    ["3", "4", "5"], rows\_bbox)
display(image_with_focused_rows)
```

OBSERVATION: Execution success. The output is as follows:
<the image outputs of the code is added as the second image>
THOUGHT 1: I can see that Josh won 3 games after 1996.
ANSWER: Josh won 3 games after 1996. FINAL ANSWER: 3. TERMINATE

EXAMPLE: Focus on specific columns and specific rows in the image
USER REQUEST #: <A image here> what is the sum of annual earnings after 2006?
USER Bounding Box Info: columns_bbox where keys are column names and values are column bounding boxes. rows_bbox, where keys are row headers and values are row bounding boxes. The columns in the image are: ["Index", "Year",

"Cost", "Earning"]. The rows in the image start with: ["Index", "0", "1", "2", "3", "4", "5"].
USER IMAGE stored in image_1, as PIL image.
RESULT #:
THOUGHT 0: I need to focus on the 'Year' column and the 'Earning' column. I also need to focus on the rows with years after 2006. The row with year being 2006 starts with "3". So I will focus on the rows starting with "4", and "5".
ACTION 0:

```python
image_with_focused_columns = focus_on_columns_with_mask(image\_1, ["Year", "Earning"], columns\_bbox)
image_with_focused_rows = focus_on_rows_with_draw(image_with_focused_columns, ["4", "5"], rows\_bbox)
display(image_with_focused_rows)
```

OBSERVATION: Execution success. The output is as follows:
<the image outputs of the code is added as the second image>
THOUGHT 1: I can see that the annual earnings after 2006 are $165,498 and $198,765. The sum of the annual earnings after 2006 is $364,263.
ANSWER: The sum of the annual earnings after 2006 is $364,263. FINAL ANSWER: 364263. TERMINATE.
USER Bounding Box Info: x_values_bbox, storing x values and coordinates. y_values_bbox, storing x values and coordinates. The x values in the image are: <x_values>. The y values in the image are: <y_values>.
USER IMAGE stored in image_1, as PIL image.