REFOCUS: VISUAL EDITING AS A CHAIN OF THOUGHT FOR STRUCTURED IMAGE UNDERSTANDING

Xingyu Fu[1]*, **Minqian Liu**[2], **Zhengyuan Yang**[3], **John Corring**[3], **Yijuan Lu**[3],
Jianwei Yang[3], Dan Roth[1], Dinei Florencio[3], Cha Zhang[3]
[1]University of Pennsylvania, [2]Virginia Tech, [3]Microsoft

https://zeyofu.github.io/ReFocus/

ABSTRACT

Structured image understanding, such as interpreting tables and charts, requires strategically refocusing across various structures and texts within an image, forming a reasoning sequence to arrive at the final answer. However, current multimodal large language models (LLMs) lack this multihop selective attention capability. In this work, we introduce REFOCUS, a simple yet effective framework that equips multimodal LLMs with the ability to generate "visual thoughts" by performing visual editing on the input image through code, shifting and refining their visual focuses. Specifically, REFOCUS enables multimodal LLMs to generate Python codes to call tools and modify the input image, sequentially drawing boxes, highlighting sections, and masking out areas, thereby enhancing the visual reasoning process. We experiment upon a wide range of structured image understanding tasks involving tables and charts. REFOCUS largely improves performance on all tasks over GPT-4o without visual editing, yielding an average gain of 11.0% on table tasks and 6.8% on chart tasks. We present an in-depth analysis of the effects of different visual edits, and reasons why REFOCUS can improve the performance without introducing additional information. Further, we collect a 14k training set using REFOCUS, and prove that such visual chain-of-thought with intermediate information offers a better supervision than standard VQA data, reaching a 8.0% average gain over the same model trained with QA pairs and 2.6% over CoT.

1 INTRODUCTION

Structured images, such as tables and charts Kafle et al. (2018); Chen et al. (2020); Zhu et al. (2021); Masry et al. (2022); Kim et al. (2024); Wang et al. (2024), are essential for the efficient communication of information in our daily lives Chang et al. (2008); Groom et al. (2016). Understanding these structured images requires multiple reasoning steps – each paying **selective attention** Duncan (1984); Johnston & Dark (1986); Rizzolatti et al. (1994) to focus on particular pieces of related information while ignoring less pertinent details and distractions, to facilitate multi-hop visual reasoning and arrive at the final answer. For instance, `What's total wins by Belgian riders` in Figure 1? To answer this, we might 1) identify the teams with Country being Belgium, 2) find the wins of each team, 3) locate the wins of the teams from Belgium, and 4) sum the wins together.

One major limitation of current multimodal models is their lack of selective attention and multi-hop visual reasoning ability – they limit intermediate reasoning to textual formats only. Most methods often extract the information from the image to text first, and then rely on language model's chain-of-thought (CoT) Wei et al. (2022) reasoning to solve the real problems Masry et al. (2023); Han et al. (2023); Liu et al. (2023c); Zhang et al. (2023); Chen et al. (2024); Surís et al. (2023); Gupta & Kembhavi (2023), never looking back at the image again. Recent work Visual Sketchpad Hu et al. (2024) for the first time attempts to create visual artifacts as thoughts using vision tools. However, it only works on natural images and mainly benefits from additional information brought by tools instead of from visual reasoning.

*Work done during internship at Microsoft. Correspond to Xingyu Fu: xingyuf2@seas.upenn.edu.

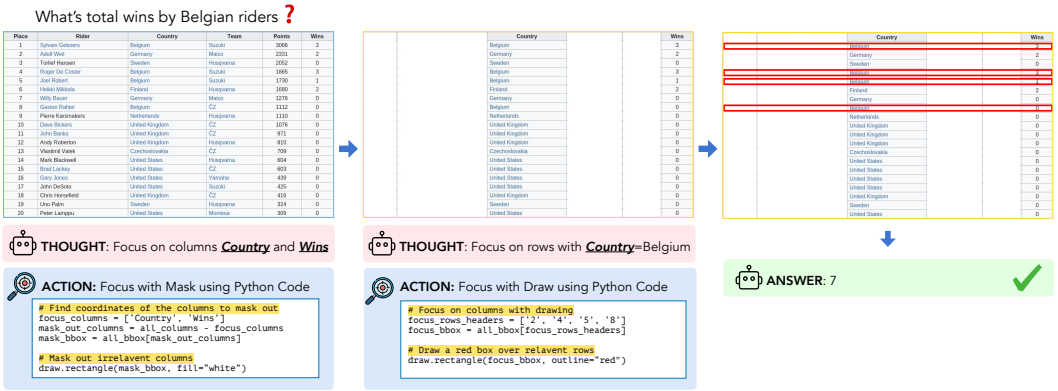


Figure 1: **Overview of REFOCUS**. REFOCUS performs visual chain of thought via input-image editing on an example data from TableVQA Kim et al. (2024). Given an image and question pair, REFOCUS equips GPT-4 with editing tools (details in §3), and GPT-4 generates pseudo code if an edit action is needed. REFOCUS then executes the editing actions, and feeds GPT-4 with the new image until an answer is reached. In the above example, `mask_column` and `draw_row` are performed.

In this work, we explore an improved representation of intermediate thoughts, to boost the visual reasoning ability of multimodal large language models (LLMs) on structured image understanding. We demonstrate that simple visual editing actions generated as Python code from multimodal LLMs– such as drawing boxes, highlighting areas, or masking regions–can significantly improve model performance by directing selective attention.

To this end, we introduce REFOCUS: a framework that enhances multimodal LLMs by integrating visual reasoning as an intermediate step. REFOCUS provides an interface that allows models to generate visual artifacts with a set of image editing tools implemented in Python code. Specifically, REFOCUS prompts the underlying multimodal LLM to program, executes the code, and produces visual artifacts that incorporate selective attention as the new input for the model. For instance, for long and complicated tabular image as in Figure 1, REFOCUS masks irrelevant columns and highlights important rows by drawing boxes around them. Similarly, to find "`the average of last four countries data`" in Figure 2, REFOCUS modifies the figure by removing the data of other countries. The new image therefore eliminate distractions, avoiding possible hallucinations, and become easier to solve for multimodal LLMs by refocusing.

We demonstrate the effectiveness of REFOCUS across a wide range of structured image tasks, focusing on table and chart visual question answering (VQA) sets. For tabular problems, REFOCUS enables models to selectively edit the columns and rows in the input image. REFOCUS consistently improves the baseline GPT-4o performance, yielding an average gain of **11.0%**. For chart problems, we tackle diverse types of charts including (1) horizontal bar charts, (2) vertical bar charts, and (3) complex scientific charts from arXiv papers. REFOCUS empowers multimodal LLMs to modify bars and subplots to pay selective attention, resulting in consistent improvements across different types of chart images, with the average gain over gpt-4o reaching **6.8%**. We present an in-depth analysis on REFOCUS. We first study why REFOCUS could achieve large gains, especially since it does not bring in external information as other methods Hu et al. (2024) do, and examine how different editing techniques could affect multimodal LLMs differently.

Further, we investigate whether we can distill such refocus abilities to smaller models through supervised-finetuning (SFT), and if such visual chain-of-thought (VCoT) data could benefit models more than the standard visual question answering (VQA) data. We collect a 14k training set data including the focus area bounding boxes and reasoning processes using REFOCUS + GPT4o. Surprisingly, comprehensive experiments on Phi-3.5-vision show that our collected REFOCUS data serves a better supervision signal than standard VQA data for SFT, achieving an average gain of **8.0%** over the same model trained with the same QA data, and **2.6%** over CoT data.

To summarize, we introduce (1) REFOCUS, a simple yet effective visual reasoning framework that enhances structured image understanding through input-image editing; (2) we demonstrate that REFOCUS consistently achieves performance improvements and analyze underlying reasons behind these gains; (3) we curate a 14k training set using REFOCUS and GPT-4o, and show that model

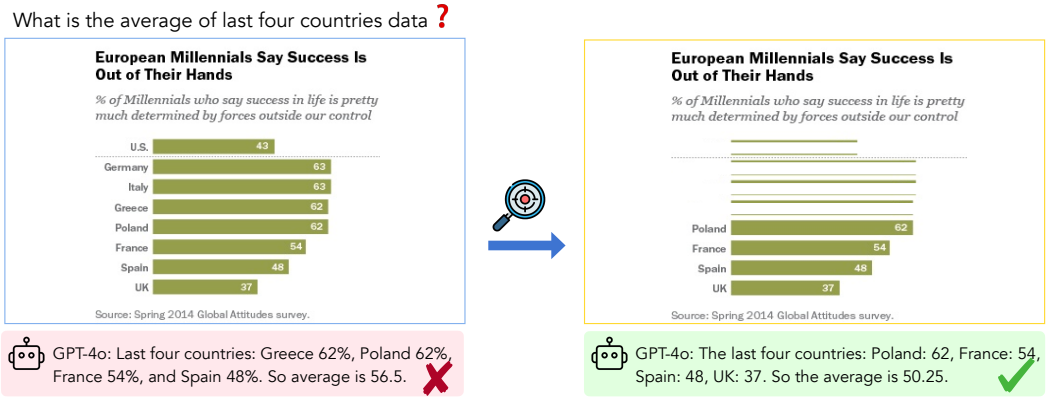


Figure 2: Example of how REFOCUS + GPT-4o solves previously unsolvable problem in ChartQA dataset Masry et al. (2022) through improved **visual grounding**. Given the original horizontal bar image (left), GPT-4o grounds to the wrong bars and thus gets the wrong answer. REFOCUS eliminates such possibility through editing, guiding the model to the correct answer (right).

finetuned with REFOCUS data consistently outperforms the same model trained with the same set of QA or CoT data, suggesting potential pathways for more intelligent reasoning in vision language models.

2 RELATED WORKS

Structured Image Understanding Scientific chart and table figures have long been a challenging task for modern multimodal models. Previous methods mainly utilize Optical Character Recognition (OCR) Liu et al. (2024b), by turning the image into text format and then performing textual reasoning Liu et al. (2022; 2023a). Other methods Masry et al. (2023); Han et al. (2023) focus on enhancing end-to-end VQA capabilities by training on augmented data.

Visual Reasoning through Programming A significant recent trend in multimodal LLMs is using Python programs to facilitate chain-of-thought reasoning Gupta & Kembhavi (2023); Surís et al. (2023). However, these methods typically conduct reasoning at the text level. While some methods may utilize image crops, the images themselves remain unchanged, which limits their visual reasoning capabilities. More recent work, such as Visual Sketchpad Hu et al. (2024), advances visual reasoning by providing various computer vision tools like Segment Anything, DINO, and Depth Anything Yang et al. (2024); Li et al. (2023); Liu et al. (2023d) to create visual artifacts. However, Visual Sketchpad mainly achieves performance gains by incorporating external expert knowledge through these tools, whereas we explore the possibility of enhancing visual understanding without additional information. Also, Visual Sketchpad cannot solve text-rich images in structured image problems, as its tools are vision-based and only address object-centric issues. With REFOCUS, LLMs can write code to call different functions for editing the input image, sequentially simplifying the original problem to make it easier for multimodal LLMs to solve. This approach enables better visual understanding and reasoning without relying on external data or expert knowledge.

Visual Reasoning through Visualization It has become evident that certain types of visual prompts on the input images, such as adding a red circle around a target object, can significantly impact the performance of multimodal language models Shtedritski et al. (2023), on certain abilities such as visual grounding Shao et al. (2024a). The Set-of-Mark study Yang et al. (2023) extends this investigation by demonstrating that segmenting the original image can significantly enhance the visual grounding capabilities of GPT-4v models. Meanwhile, it has been highlighted in BLINK Fu et al. (2024) that most open-source multimodal language models may struggle to comprehend visual prompts. Recent works, such as VIP-LLaVA and List Items One by One Cai et al. (2024); Yan et al. (2024), have further refined these models to improve their understanding and processing of visual prompts. Most of these visual prompts are visual objects centered and do not apply to structured images. One recent work Visual Cot Shao et al. (2024a) extracts the subarea in images where the answer corresponds to, e.g. obtaining the coordinates of answer text using OCR on chart VQA problems, or find the answer object's coordinates in the images using object detection methods. However, Visual Cot do not involve multistep reasoning and refocusing processes and only focuses

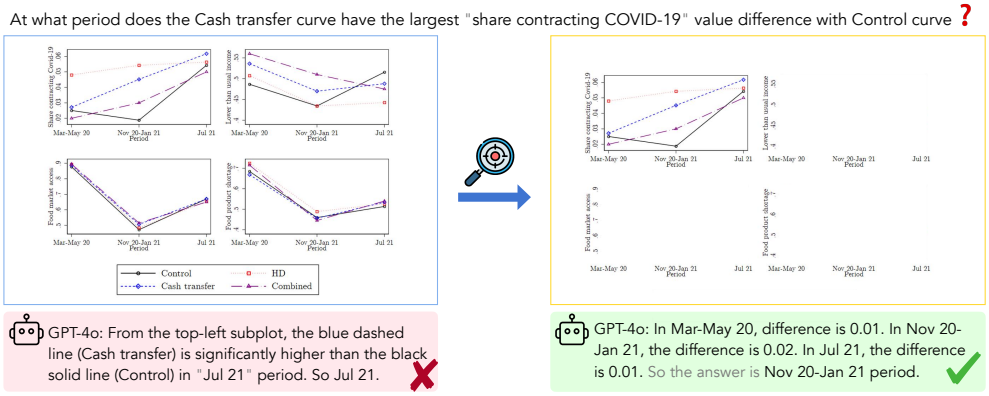


Figure 3: **REFOCUS equips GPT-4 with selective attention**. Above is an example of how REFOCUS + GPT-4o solves previously unsolvable problem in ChartXiv dataset Wang et al. (2024). Specifically, REFOCUS edits upon the original image by masking out all irrelevant information – the other three subplots that could be distracting. As a result, GPT-4o is able to conduct better reasoning with the edited image, and reach the correct answer.

on image region–answer text direct mapping. We compare our results with Visual Cot on Chart QA datasets in Section 5 and show absolute improvement.

3 REFOCUS

REFOCUS conducts visual editing on input images as a chain-of-thought, providing an intermediate interface for multimodal LLMs to facilitate visual reasoning and enforce the correct answer with a valid reasoning process. The entire pipeline is iterative, as illustrated in Figure 1: the multimodal LLM sees the image and question, proceeds with one step of thought, conducts visual editing using Python code, and continues with the next step of thought and editing until it arrives at the final answer. We specifically target two sets of structured image problems in this work: (1) tabular figures, and (2) chart figures. This section unfolds by introducing the details of our tasks (§3.1), the implementation of visual editing tools for tables (§3.2) and for charts, and how to equip multimodal LLMs with these editing tools (§3.3).

3.1 STRUCTURED IMAGE PROBLEMS

Tabular Problems. Tabular understanding has long been a challenging task for multimodal LLMs Liu et al. (2022; 2023a). In this paper, we use TableVQA Kim et al. (2024) as our test-bed. We experiment on three types of table data:

VWTQ. VWTQ is derived from WikiTableQuestion (WTQ) Pasupat & Liang (2015), which provides original HTML data on Wikipedia tables along with corresponding Question Answering (QA) pairs and maintain its accuracy-based evaluation metric. Kim et al. (2024) reproduces the original table images from WTQ by applying the stylesheet of Wikipedia to the HTML and capturing screenshots of the table images. There are a total of 750 Visual Question Answering (VQA) pairs.

VWTQ_syn. Considering that the table figures in VWTQ come from Wikipedia and can easily be web-crawled to gather pre-training data for multimodal LLMs, Kim et al. (2024) generates synthetic table images using a table rendering system based on the WTQ data. This system takes HTML as input and generates tables with various styles, featuring random attributes such as background colors, border margins, and font families. There are a total of 250 Visual Question Answering (VQA) pairs in this synthetic dataset.

VTabFact. TabFact Chen et al. (2019) represents a verification task that determines whether a statement is entailed or refuted given table data in text format. Kim et al. (2024) proposes a rendering

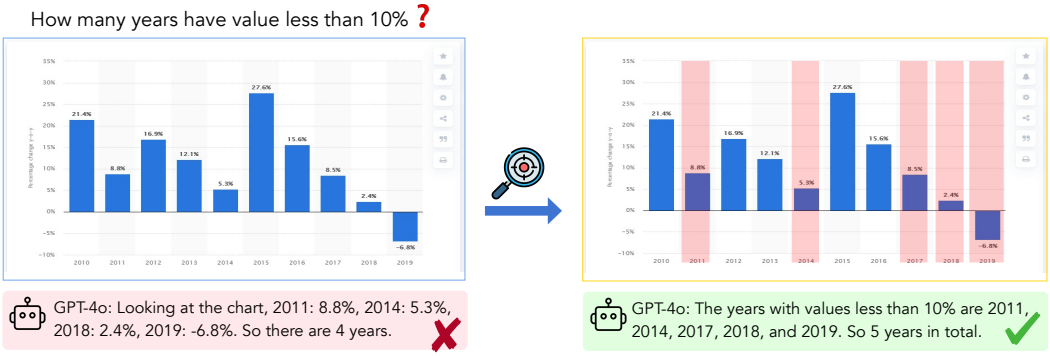


Figure 4: **REFOCUS unleashes better visual grounding and counting abilities for GPT-4** as in ChartQA Masry et al. (2022) Vertical Bar problems.

system that generates visual tables using pseudo-HTML converted from the textual table data. There are a total of 250 Visual Question Answering (VQA) pairs in this dataset.

Coordinate Acquisition for Tables.To support visual editing, we need to acquire coordinates for each column and row. We accomplish this using the opencv-python[1] package with functions such as *findContours()* and *getStructuringElement()*. The heuristic behind this process is that we detect lines and boxes around text in the figure. The longest vertical line should correspond to the row length, and the longest horizontal line should correspond to the column length. By combining the longest contours in the figure, the coordinates of text boxes, and the row and column lengths, we can determine the coordinates of each column and row.

Chart Problems. Chart understanding is another fundamental task Liu et al. (2022; 2023a), serving a pivotal role in real-world applications such as analyzing scientific papers or financial reports. We use the datasets CharXiv Wang et al. (2024) and ChartQA Masry et al. (2022) for our experiments:

CharXiv Multi-subplot. CharXiv is a high-quality, expert-annotated, challenging VQA benchmark that involves natural and diverse charts from arXiv papers. CharXiv includes two types of questions: descriptive questions that examine basic chart elements and reasoning questions that require synthesizing information across complex visual elements in the chart. In this paper, we specifically focus on the challenging reasoning questions that involve multiple subplots. Due to computation cost concerns, we randomly select 200 VQA pairs out of the 1000 test data and use GPT-4 to select the ones with multiple subplots for experimental purposes, yielding a total of 143 VQA pairs.

Horizontal Bar. ChartQA Masry et al. (2022) is one of the most popular chart-based VQA benchmarks, consisting of human-written questions focusing on logical and visual reasoning based on web-crawled diverse chart figures. For our experiments, we take the horizontal bar figures and their corresponding VQA data from the test split. This subset includes a total of 444 VQA pairs.

Vertical Bar. Similar to the horizontal bar figures, we also take the vertical bar figures along with their corresponding VQA data from the test split of the ChartQA Masry et al. (2022) dataset. This subset includes a total of 382 VQA pairs.

Coordinate Acquisition for Charts. For CharXiv multi-subplot figures, we acquire the coordinates of each subplot using the opencv-python[1] package with functions such as *findContours()* and *getStructuringElement()*. The heuristic behind this process is that we take the top k longest contours, provide them in a prompt to LLMs, and allow the LLMs to determine which coordinates correspond to which subplot (k = 10 in our experiments). Regarding ChartQA horizontal bar figures and vertical bar figures, we utilize the provided coordinates of x-axis values and y-axis values in the original dataset. Additionally, we combine this information with coordinate details about the chart area (excluding the caption) obtained using the opencv-python[1] package.

[1] https://pypi.org/project/opencv-python/

	Table			Chart		
Model	VWTQ	VWTQ_syn	VTabFact	CharXiv	Horizontal Bar	Vertical Bar
Prior Multimodal LMs						
LLaVA-NeXT-34B Liu et al. (2024a)	36.4	38.0	71.2	18.9	23.4	12.6
Phi 3 vision (Abdin et al., 2024)	44.7	53.2	74.4	16.2	60.8	66.5
Gemini-Pro 1.5 Team et al. (2023)	38.5	43.2	75.6	38.3	57.2	66.0
VisProg Gupta & Kembhavi (2023)	53.2	62.0	76.4	46.8	69.8	68.6
Latest multimodal LLMs + REFOCUS						
gpt-4o-2024-05-13 OpenAI (2023a)	66.5	73.2	89.6	49.0	78.2	76.2
+ REFOCUS	76.9	79.6	89.6	**57.3**	**85.4**	81.0
	+10.4	+3.4	+0.0	+8.3	+7.2	+4.8
gpt-4o-2024-08-06 OpenAI (2023a)	66.4	70.4	90.0	48.9	75.2	74.9
+ REFOCUS	**77.2**	**82.8**	**90.8**	46.2	82.0	**81.2**
	+9.8	+12.4	+0.8	-2.7	+5.0	+4.2

Table 1: REFOCUS yields consistent performance gains across all tasks and outperforms all baselines. Notice that the GPT baselines here are also in a conversational format but without editing abilities. For fair comparison, we modify the original Visprog Gupta & Kembhavi (2023) framework by replacing the LM and VQA components with the latest GPT-4o model.

3.2 VISUAL EDITING TOOLS

We adopt three types of visual editing method in REFOCUS: mask out, draw box, and highlight color using Python code. In our experiments for **tabular problems**, we utilize a variety of tools as follows:

Highlight Column, which overlays a light red color on the columns that need to be focused on.

Highlight Row, which overlays a light red color on the rows that need to be focused on.

Mask Column, which places a white mask over the columns that do not need attention.

Mask Row, which places a white mask over the rows that do not need attention.

Draw Column, which overlays a solid red bounding box on the columns that need to be focused on.

Draw Row, which overlays a solid red bounding box on the rows that need to be focused on.

A pseudo code for example tool: *Highlight Column* is illustrated in **??** 1. Visual output examples can be found in Figures 1 and 6.

```python
from PIL import Image, ImageDraw

def focus_on_columns_with_highlight(image, columns_to_focus_on, all_columns_bounding_boxes):
    """ This function is used to focus on some specific columns of the image. """

    # Draw a highlight color on the columns to focus on
    mask = image.convert('RGBA').copy()
    mask_draw = ImageDraw.Draw(mask)

    # Iterate over the columns to highlight
    for column_name in columns_to_focus_on:

        # Get the bounding box of the column
        column_bbox = all_columns_bounding_boxes[column_name]
        (x1, y1, x2, y2) = column_bbox
        mask_draw.rectangle(((x1, y1), (x2, y2)), fill =(255, 0, 0, 50))

    # Composite the overlay with the mask over the original image
    edited_image = Image.alpha_composite(image.convert('RGBA'), mask)
    return edited_image
```

Listing 1: Python code example for highlighting a column.

Similarly, for **chart problems**, we apply the same editing methods, but focus on different aspects: (1) subplots, particularly for CharXiv figures; (2) bars by x values, specifically for ChartQA vertical bar figures; (3) bars by y values, specifically for ChartQA horizontal bar figures. Each of these aspects is

	Table			Chart		
Model	VWTQ	VWTQ_syn	VTabFact	CharXiv	Horizontal Bar	Vertical Bar
GPT-4o Model						
Text input	68.1	69.6	80.0	\	66.2	74.9
Figure Input	61.0	68.4	88.4	47.9	75.2	74.9
Text + Figure Input	67.5	72.8	91.6	\	75.7	81.7
GPT-4o + REFOCUS						
Figure Input	77.2	82.8	90.8	57.3	85.4	81.2
	+16.2	+14.4	+2.4	+10.6	+9.7	+0.5

Table 2: **REFOCUS empowers GPT-4o to achieve the performance as if given gold text input.** The text input are mainly csv tables, as detailed in Section 3.1. Notice that Table 1 reports the conversational performance without visual editing, whereas the performance discussed here is based on direct question answering, leading to minor differences.

combined with the three editing methods: masking, drawing, and highlighting. Detailed visual output examples can be found in Figures 2 to 4, and a complete list of tools is available in the Appendix.

3.3 EQUIP LLMS WITH VISUAL EDITING TOOLS

We provide the function names of each visual editing tool in the multimodal LLM's prompts and ask it to generate pseudocode using these provided function names. The actual editing functions will then be executed when the multimodal LLM decides to perform a visual editing, and the modified images are returned to the model as the new input. More prompting details can be found in the Appendix.

4 EXPERIMENTS AND ANALYSES

We demonstrate that REFOCUS significantly improves multimodal LLMs' performance on structured images, while making the visual reasoning processes interpretable. In this section, we introduce the baseline models and experiment settings (§4.1), present comprehensive experiment results (§4.2), and show an in-depth analyses investing why REFOCUS is so effective despite its simplicity(§4.3).

4.1 BASELINES AND SETUPS

We apply REFOCUS on the state-of-the-art multimodal LLM: GPT-4o OpenAI (2023b), especially the gpt-4o-2024-05-13 and gpt-4o-2024-08-06 checkpoints to show its efficacy. We compare with the default GPT-4o for both checkpoints using chain-of-thought prompting for fair comparison. We also include several other powerful multimodal LLMs to compare with: LLaVA-NeXT Liu et al. (2024a), Phi 3 Vision[2]Abdin et al. (2024), and Gemini Pro 1.5 Team et al. (2023). In addition, we also compare our approach with the visual programming method VisProg Gupta & Kembhavi (2023), which uses Python code to generate intermediate reasoning with integrated vision modules. We modify the original VisProg framework by replacing the LLM and VQA components with the latest gpt-4o-2024-08-06 model for fair comparison. We do not include Visual Sketchpad Hu et al. (2024) since none of its tools can apply to structured image problems, leaving it the same as vanilla GPT-4o. More implementation details are provided in the Appendix.

4.2 RESULTS

As shown in Table 1, the mean accuracy of REFOCUS combined with GPT-4o consistently surpasses all baseline models, including the vanilla conversational GPT-4o model without editing abilities. REFOCUS is particularly effective on VWTQ, VWTQ_syn, Horizontal Bar, and Vertical Bar tasks, achieving a consistent 5-10% accuracy improvement over GPT-4o. To examine the quality of visual editings, as illustrated in Table 5, we further test LLaVA-NeXT (7B, 13B, 34) and Phi-3-vision

[2]We used the checkpoint at https://huggingface.co/microsoft/Phi-3-vision-128k-instruct.

Figure 5: **Training set collection using REFOCUS on ChartQA dataset.**

models using the REFOCUS + GPT-4o edited images. Interestingly, while none of the tested models are fine-tuned on images with visual prompts, we still observe a consistent improvement on VWTQ, VWTQ_syn, CharXiv, and Vertical Bar tasks. In Table 2, we compare the performance of REFOCUS versus vanilla GPT-4o with different inputs (i.e., gold table / chart text, figure, or gold text + figure). Surprisingly, GPT-4o with REFOCUS often outperforms GPT-4o with gold text and figure inputs by a significant margin on most tasks. This demonstrates that REFOCUS can effectively enhance GPT-4o's structured image understanding capability, making it perform as if it had access to the gold text input.

4.3 ANALYSES

Why can REFOCUS improve the performance? This question is interesting because REFOCUS does not introduce any additional information as other methods Yang et al. (2023); Hu et al. (2024) do. Still, it achieves an impressive performance gain upon the already powerful GPT-4o models, using a simple strategy. Our observations on the intermediate outputs suggest that this might be because REFOCUS improves GPT-4o's visual grounding and OCR abilities through selective attention and eliminates hallucinations. As shown in Figure 2, for the original image on the left, GPT-4o mistakenly grounds to Greece and misses the UK, resulting in the wrong answer of 56.5. REFOCUS decomposes the steps of visual grounding and optical character recognition (OCR). In the edited image on the right, only the last four countries are shown, and GPT-4o successfully provides the correct answer with proper reasoning. Similarily, as shown in Figure 6, when GPT-4o tries to recognize characters in the original image, it makes a spelling mistake, identifying "Partenavia" incorrectly. However, with the highlighted columns, GPT-4o can correctly recognize the characters as "Partenvia."

Which editing method works the best? Since we provide three types of editing methods: draw box, highlight color, and mask out, it raises a natural question about which method works best for the models. We conducted experiments on VWTQ and VWTQ_syn sets and show the performance differences in Table 4. In short, all tools are similar.

How frequent is visual editing performed? Since the visual editing steps are conducted through LLM generated python code, it is possible that REFOCUS does not conduct any editing for some instances. To answer how often the underlying multimodal LLM, GPT-4o in our case, decides to proceed with a visual edit, we calculated the number of images that GPT-4o decides to edit and report the numbers in Figure 7. GPT-4o clearly shows a preference to edit ¿ 85% VWTQ and CharXiv images, while the portion of edited images on the other datasets remains around 40-55%.

5 FINETUNE WITH REFOCUS DATA

While REFOCUS shows its efficacy, it is still limited to a visual prompting method. Could the visual reasoning processes of REFOCUS be distilled into smaller Multimodal LLMs and provide additional or even better supervision for models to learn? What would be the difference between a model fine-tuned on the de facto image question answer pairs, versus one fine-tuned on CoT data, or on REFOCUS visual reasoning data?

Finetune Data Collection To answer this question, we further collect a 14k training set using REFOCUS and GPT-4o upon the ChartQA Masry et al. (2022) training data. As shown in Figure 5, we apply the pipeline on 15,059 ChartQA training data. If the prediction is correct, we keep the whole process and add the generated textual chain-of-thought (CoT) reasoning, python code for editing, and the area to focus on (in bounding box format) to our data. If the prediction is incorrect, we give it one

more try and hint it with the gold answer. If it still gets to a wrong answer, we discard this example since it's most likely a data noise from our observation. We end up with 14,344 training data, among which 12,819 has editing processes. More dataset details are in Appendix D.

Finetune Setups To test the effectiveness of visual chain-of-thought for multimodal LLMs, we adopt the standard supervised-Finetuning (SFT) strategies. We use Phi-3.5-vision[3] Abdin et al. (2024) as our base model, and compare its performance when finetuned with REFOCUS data, versus with same set of data but only with QA pairs. For both finetuning, we keep all settings the same and search through the same set of hyper-parameters for learning rate and epoch number, and report the best performance. During training time, the input is given in the format of '<image> <question> <thought1> <ReFocus bounding box> <thought2> <answer>', where <Thought1> is transformed through GPT-4o returned thought on what areas to refocus on, and <Thought2> is GPT-4o output when predicting based on the edited images. In comparison, the standard QA training input is '<image> <question> <answer>'. During inference time, we examine two types of prompt settings: '<image> <question> Answer:' as the QA prompting, and '<image> <question> Thought:' as the Visual CoT prompting. More details can be found in Appendix D.

	Horizontal Bar	Vertical Bar	Avg.
QA Prompting			
Phi-3.5-vision Hu et al. (2022)	60.1	63.1	61.5
SFT w/ QA Data	60.1	65.5	62.6
Visual CoT Prompting			
Phi-3.5-vision Hu et al. (2022)	69.4	66.8	68.2
SFT w/ QA Data	60.6	66.8	63.4
SFT w/ REFOCUS CoT	67.1	70.7	68.8
SFT w/ REFOCUS VCoT	**71.0**	**72.2**	**71.4**

Table 3: **SFT accuracy results.** The difference between REFOCUS VCoT data and CoT data is that VCoT contains refocus area bounding box coordinates whereas CoT does not. All trainings select the best performing hyper-parameters on the same set of training data.

Supervised-Finetuning Results We report our results as in Table 3 with multiple setups – the REFOCUS data is Visual CoT (VCoT) data, and we compare with the standard QA data. Another setup is finetuning with CoT data, which is VCoT data without refocus area bounding box information. The Phi-3.5-Vision model finetuned with REFOCUS VCoT data outperforms the base model by 3.2% in accuracy; over the same model finetuned with same set of QA data by 8.0%; and over the model finetuned with CoT data by 2.6%. Notably, our results largely exceed Visual Cot Shao et al. (2024b) which trains upon LLaVa-1.5 Liu et al. (2023b) with answer coordinate data by 50.6%. These results suggest that implicit visual reasoning data such as REFOCUS VCoT can consistently provide better supervision signals than standard QA pairs and CoT data for structured image understanding tasks. Our approach has the potential to provide rich and meaningful supervision data to enhance multimodal LLMs in general. More SFT details, result analyses, and image editing outputs can be found in Appendices D and E.

6 CONCLUSION

This work introduces REFOCUS, a simple yet effective framework that enhances the ability of multimodal large language models (LLMs) to interpret structured images, by incorporating visual editing on the input image through Python code. Our approach significantly boosts performance on table and chart tasks. Further, we present a 14k visual chain-of-thought training data built using REFOCUS, which demonstrates superior supervision over standard QA data.

[3] https://github.com/microsoft/Phi-3CookBook/blob/main/md/04.Fine-tuning/FineTuning_Vision.md

REFERENCES

Marah Abdin, Sam Ade Jacobs, Ammar Ahmad Awan, Jyoti Aneja, Ahmed Awadallah, Hany Awadalla, Nguyen Bach, Amit Bahree, Arash Bakhtiari, Harkirat Behl, et al. Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*, 2024. 6, 7, 9, 13

Mu Cai, Haotian Liu, Siva Karthik Mustikovela, Gregory P Meyer, Yuning Chai, Dennis Park, and Yong Jae Lee. Vip-llava: Making large multimodal models understand arbitrary visual prompts. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 12914–12923, 2024. 3

Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C Hsieh, Deborah A Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E Gruber. Bigtable: A distributed storage system for structured data. *ACM Transactions on Computer Systems (TOCS)*, 26(2):1–26, 2008. 1

Wenhu Chen, Hongmin Wang, Jianshu Chen, Yunkai Zhang, Hong Wang, Shiyang Li, Xiyou Zhou, and William Yang Wang. Tabfact: A large-scale dataset for table-based fact verification. *arXiv preprint arXiv:1909.02164*, 2019. 4

Wenhu Chen, Hanwen Zha, Zhiyu Chen, Wenhan Xiong, Hong Wang, and William Wang. Hybridqa: A dataset of multi-hop question answering over tabular and textual data. *arXiv preprint arXiv:2004.07347*, 2020. 1

Zhenfang Chen, Qinhong Zhou, Yikang Shen, Yining Hong, Zhiqing Sun, Dan Gutfreund, and Chuang Gan. Visual chain-of-thought prompting for knowledge-based visual reasoning. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pp. 1254–1262, 2024. 1

John Duncan. Selective attention and the organization of visual information. *Journal of experimental psychology: General*, 113(4):501, 1984. 1

Xingyu Fu, Yushi Hu, Bangzheng Li, Yu Feng, Haoyu Wang, Xudong Lin, Dan Roth, Noah A Smith, Wei-Chiu Ma, and Ranjay Krishna. Blink: Multimodal large language models can see but not perceive. *arXiv preprint arXiv:2404.12390*, 2024. 3

Colin R Groom, Ian J Bruno, Matthew P Lightfoot, and Suzanna C Ward. The cambridge structural database. *Structural Science*, 72(2):171–179, 2016. 1

Tanmay Gupta and Aniruddha Kembhavi. Visual programming: Compositional visual reasoning without training. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 14953–14962, 2023. 1, 3, 6, 7

Yucheng Han, Chi Zhang, Xin Chen, Xu Yang, Zhibin Wang, Gang Yu, Bin Fu, and Hanwang Zhang. Chartllama: A multimodal llm for chart understanding and generation. *arXiv preprint arXiv:2311.16483*, 2023. 1, 3

Edward J Hu, yelong shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shean Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*, 2022. URL https://openreview.net/forum?id=nZeVKeeFYf9. 9

Yushi Hu, Weijia Shi, Xingyu Fu, Dan Roth, Mari Ostendorf, Luke Zettlemoyer, Noah A Smith, and Ranjay Krishna. Visual sketchpad: Sketching as a visual chain of thought for multimodal language models. *arXiv preprint arXiv:2406.09403*, 2024. 1, 2, 3, 7, 8

William A Johnston and Veronica J Dark. Selective attention. *Annual review of psychology*, 1986. 1

Kushal Kafle, Brian Price, Scott Cohen, and Christopher Kanan. Dvqa: Understanding data visualizations via question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 5648–5656, 2018. 1

Yoonsik Kim, Moonbin Yim, and Ka Yeon Song. Tablevqa-bench: A visual question answering benchmark on multiple table domains. *arXiv preprint arXiv:2404.19205*, 2024. 1, 2, 4, 13

Feng Li, Hao Zhang, Peize Sun, Xueyan Zou, Shilong Liu, Jianwei Yang, Chunyuan Li, Lei Zhang, and Jianfeng Gao. Semantic-sam: Segment and recognize anything at any granularity. *arXiv preprint arXiv:2307.04767*, 2023. 3

Fangyu Liu, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Yasemin Altun, Nigel Collier, and Julian Martin Eisenschlos. Matcha: Enhancing visual language pretraining with math reasoning and chart derendering. *arXiv preprint arXiv:2212.09662*, 2022. 3, 4, 5

Fangyu Liu, Julian Martin Eisenschlos, Francesco Piccinno, Syrine Krichene, Chenxi Pang, Kenton Lee, Mandar Joshi, Wenhu Chen, Nigel Collier, and Yasemin Altun. Deplot: One-shot visual language reasoning by plot-to-table translation. In *Findings of the 61st Annual Meeting of the Association for Computational Linguistics*, 2023a. URL https://arxiv.org/abs/2212.10505. 3, 4, 5

Haotian Liu, Chunyuan Li, Qingyang Wu, and Yong Jae Lee. Visual instruction tuning, 2023b. 9

Haotian Liu, Chunyuan Li, Yuheng Li, Bo Li, Yuanhan Zhang, Sheng Shen, and Yong Jae Lee. Llava-next: Improved reasoning, ocr, and world knowledge, January 2024a. URL https://llava-vl.github.io/blog/2024-01-30-llava-next/. 6, 7, 13

Shilong Liu, Hao Cheng, Haotian Liu, Hao Zhang, Feng Li, Tianhe Ren, Xueyan Zou, Jianwei Yang, Hang Su, Jun Zhu, et al. Llava-plus: Learning to use tools for creating multimodal agents. *arXiv preprint arXiv:2311.05437*, 2023c. 1

Shilong Liu, Zhaoyang Zeng, Tianhe Ren, Feng Li, Hao Zhang, Jie Yang, Chunyuan Li, Jianwei Yang, Hang Su, Jun Zhu, et al. Grounding dino: Marrying dino with grounded pre-training for open-set object detection. *arXiv preprint arXiv:2303.05499*, 2023d. 3

Yuliang Liu, Zhang Li, Mingxin Huang, Biao Yang, Wenwen Yu, Chunyuan Li, Xucheng Yin, Cheng lin Liu, Lianwen Jin, and Xiang Bai. Ocrbench: On the hidden mystery of ocr in large multimodal models, 2024b. URL https://arxiv.org/abs/2305.07895. 3

Ahmed Masry, Do Xuan Long, Jia Qing Tan, Shafiq Joty, and Enamul Hoque. Chartqa: A benchmark for question answering about charts with visual and logical reasoning. *arXiv preprint arXiv:2203.10244*, 2022. 1, 3, 5, 8, 22

Ahmed Masry, Parsa Kavehzadeh, Xuan Long Do, Enamul Hoque, and Shafiq Joty. Unichart: A universal vision-language pretrained model for chart comprehension and reasoning. *arXiv preprint arXiv:2305.14761*, 2023. 1, 3

OpenAI. Gpt-4 technical report, 2023a. 6

OpenAI. Gpt-4 technical report, 2023b. 7

Panupong Pasupat and Percy Liang. Compositional semantic parsing on semi-structured tables. *arXiv preprint arXiv:1508.00305*, 2015. 4

Giacomo Rizzolatti, Lucia Riggio, Boris M Sheliga, et al. Space and selective attention. *Attention and performance XV*, 15:231–265, 1994. 1

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual cot: Advancing multi-modal language models with a comprehensive dataset and benchmark for chain-of-thought reasoning. In *The Thirty-eight Conference on Neural Information Processing Systems Datasets and Benchmarks Track*, 2024a. 3

Hao Shao, Shengju Qian, Han Xiao, Guanglu Song, Zhuofan Zong, Letian Wang, Yu Liu, and Hongsheng Li. Visual cot: Unleashing chain-of-thought reasoning in multi-modal language models, 2024b. 9

Aleksandar Shtedritski, Christian Rupprecht, and Andrea Vedaldi. What does clip know about a red circle? visual prompt engineering for vlms. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pp. 11987–11997, 2023. 3

Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. *Proceedings of IEEE International Conference on Computer Vision (ICCV)*, 2023. 1, 3

Gemini Team, Rohan Anil, Sebastian Borgeaud, Yonghui Wu, Jean-Baptiste Alayrac, Jiahui Yu, Radu Soricut, Johan Schalkwyk, Andrew M Dai, Anja Hauth, et al. Gemini: a family of highly capable multimodal models. *arXiv preprint arXiv:2312.11805*, 2023. 6, 7

Zirui Wang, Mengzhou Xia, Luxi He, Howard Chen, Yitao Liu, Richard Zhu, Kaiqu Liang, Xindi Wu, Haotian Liu, Sadhika Malladi, et al. Charxiv: Charting gaps in realistic chart understanding in multimodal llms. *arXiv preprint arXiv:2406.18521*, 2024. 1, 4, 5

Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al. Chain-of-thought prompting elicits reasoning in large language models. *Advances in neural information processing systems*, 35:24824–24837, 2022. 1

An Yan, Zhengyuan Yang, Junda Wu, Wanrong Zhu, Jianwei Yang, Linjie Li, Kevin Lin, Jianfeng Wang, Julian McAuley, Jianfeng Gao, et al. List items one by one: A new data source and learning paradigm for multimodal llms. *arXiv preprint arXiv:2404.16375*, 2024. 3

Jianwei Yang, Hao Zhang, Feng Li, Xueyan Zou, Chunyuan Li, and Jianfeng Gao. Set-of-mark prompting unleashes extraordinary visual grounding in gpt-4v. *arXiv preprint arXiv:2310.11441*, 2023. 3, 8

Lihe Yang, Bingyi Kang, Zilong Huang, Xiaogang Xu, Jiashi Feng, and Hengshuang Zhao. Depth anything: Unleashing the power of large-scale unlabeled data. In *CVPR*, 2024. 3

Zhuosheng Zhang, Aston Zhang, Mu Li, Hai Zhao, George Karypis, and Alex Smola. Multimodal chain-of-thought reasoning in language models. *arXiv preprint arXiv:2302.00923*, 2023. 1

Fengbin Zhu, Wenqiang Lei, Youcheng Huang, Chao Wang, Shuo Zhang, Jiancheng Lv, Fuli Feng, and Tat-Seng Chua. Tat-qa: A question answering benchmark on a hybrid of tabular and textual content in finance. *arXiv preprint arXiv:2105.07624*, 2021. 1

A OVERVIEW

In the supplemental materials, Appendix B contains additional details for REFOCUS, including editing tools(§B.3), and experiment configurations such as baseline model setups (§B.4). Appendix D discusses supervised fine-tuning details including experiment setups(§D.2), dataset statistics(§D.1), and result analyses(§D.3). Appendix C demonstrates prompts in REFOCUS. Appendix E shows qualitative SFT output examples.

B REFOCUS PROMPTING DETAILS

B.1 ADDITIONAL EXAMPLE

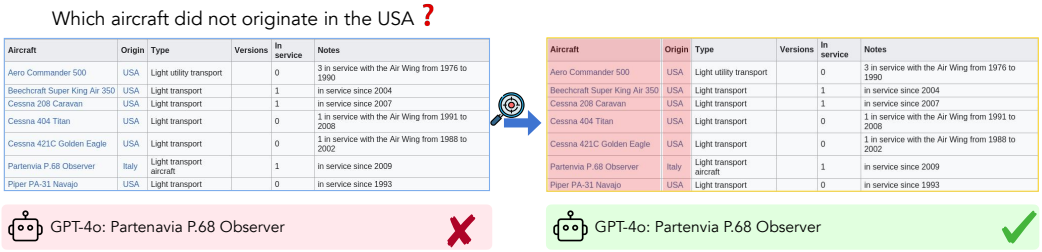


Figure 6: **REFOCUS unleashes better OCR for GPT-4.** In this example from TableVQA Kim et al. (2024), REFOCUS + GPT-4 conducts the edit action `highlight_column`. With this simple action, GPT-4 can focus more on the important subarea, and recognize the characters better.

B.2 ADDITIONAL ANALYSIS

Dataset	Original	Mask out	Draw Box	Highlight
VWTQ	66.4	77.2	**77.6**	74.8
VWTQ_syn	70.4	**82.4**	78.8	80.8

Table 4: Analysis on how different editing tool can affect model performance. We control the tool type provided by REFOCUS and experiment on the VWTQ and VWTQ_syn datasets.

	Table			**Chart**		
Model	VWTQ	VWTQ_syn	VTabFact	CharXiv	Horizontal Bar	Vertical Bar
Multimodal LLMs with original visual inputs						
LLaVA-NeXT-7B Liu et al. (2024a)	21.7	24.0	56.8	16.1	8.1	7.3
LLaVA-NeXT-13B Liu et al. (2024a)	25.6	30.4	62.4	18.9	13.5	13.1
LLaVA-NeXT-34B Liu et al. (2024a)	36.4	38.0	71.2	18.9	23.4	12.6
Phi 3 vision (Abdin et al., 2024)	44.7	53.2	**74.4**	16.2	**60.8**	**66.5**
Multimodal LLMs with REFOCUS *edited visual inputs*						
LLaVA-NeXT-7B + Oracle REFOCUS	25.7	26.8	55.2	15.4	6.8	8.4
LLaVA-NeXT-13B + Oracle REFOCUS	30.3	31.6	61.2	17.5	15.5	13.1
LLaVA-NeXT-34B + Oracle REFOCUS	39.1	41.2	68.0	**21.0**	26.1	15.2
Phi 3 vision + Oracle REFOCUS	**48.3**	**56.4**	72.8	17.6	60.4	**66.5**

Table 5: Open-source models' performance upon the original visual input versus GPT-4o + REFOCUS edited images as visual input (referred by + oracle REFOCUS). Notice that + oracle REFOCUS uses the visual artifact generated in the last action of GPT-4o + REFOCUS as inputs.

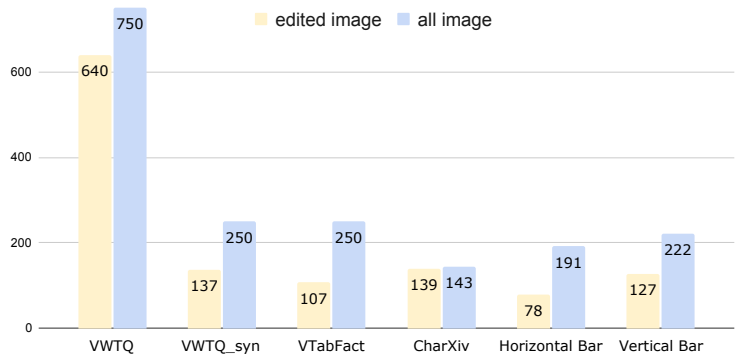


Figure 7: Statistics of how often visual editing are performed.

B.3 Visual Editing Tools for Charts

As introducted in Section 3.2, we adopt three types of visual editing method in REFOCUS: mask out, draw box, and highlight color using Python code. In our experiments for **chart problems**, we deploy a list of tools as follows:

Highlight Bar at X, which overlays a light red color over a bar in a vertical bar figure that needs to be focused. It operates based on the x-value of the chosen bar, as in Figure 4.

Highlight Bar at Y, which overlays a light red color over a bar in a horizontal bar figure that needs to be focused. It operates based on the y-value of the chosen bar, as in Figure 2.

Mask Bar at X, which places a white mask over bars in a vertical bar figure that do not need attention.

Mask Bar at Y, which places a white mask over bars in a horizontal bar figure that do not need attention.

Draw Bar at X, which overlays a solid red bounding box over bars in a vertical bar figure that need to be focused on.

Draw Bar at Y, which overlays a solid red bounding box over bars in a horizontal bar figure that need to be focused on.

B.4 Experiment Details

For all the experiments, the temperature is set to 0. All evaluations are done using GPT-4, where it's given a prediction and the gold answer, and decides whether the prediction is correct or not. For the Section 4 experiments, we use 4 NVIDIA Quadro RTX 8000 GPUs for the inference of open-source multimodal LLMs, including LLaVA-NeXT-7B, LLaVA-NeXT-13B, LLaVA-NeXT-34B, and Phi 3 vision (4B). These experiments cost around 40 GPU hours.

C PROMPTS

We show REFOCUS prompting details for table problems and chart problems with running examples randomly selected from VWTQ and Horizontal Bar datasets as follows.

Prompts for Table Problems:

SYSTEM PROMPT

You are a helpful multimodal AI assistant. [MORE INSTRUCTIONS ...]
For each turn, you should first do a "THOUGHT", based on the images and text you see. If you think you get the answer to the intial user request, you can reply with "ANSWER: <your answer>" and ends with "TERMINATE".

Initial Prompt + Request

```
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 jupyter notebook code block) 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.
Your visual ability is not perfect, so you should use these tools to assist you in reasoning about the images.
The jupyter notebook has already executed the following code to import the necessary packages:
```python
from PIL import Image
from IPython.display import display
from tools import *
```

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.

[four in-context examples here]

USER REQUEST #:
```
```

Golfer	Country	Wins	Match Play	Championship	Invitational	Champions
Woods, TigerTiger Woods	United States	18	**3:** 2003, 2004, 2008	**7:** 1999, 2002, 2003, 2005, 2006, 2007, 2013	**8:** 1999, 2000, 2001, 2005, 2006, 2007, 2009, 2013	—
Ogilvy, GeoffGeoff Ogilvy	Australia	3	**2:** 2006, 2009	**1:** 2008	—	—
Clarke, DarrenDarren Clarke	Northern Ireland	2	**1:** 2000	—	**1:** 2003	—
Els, ErnieErnie Els	South Africa	2	—	**2:** 2004, 2010	—	—
Mahan, HunterHunter Mahan	United States	2	**1:** 2012	—	**1:** 2010	—
Mickelson, PhilPhil Mickelson	United States	2	—	**1:** 2009	—	**1:** 2009
Poulter, IanIan Poulter	England	2	**1:** 2010	—	—	**1:** 2012

```
Tell me the number of wins els had.

# USER Bounding Box Info: columns_bbox, where keys are column headers and values are column bounding boxes. rows_bbox, where keys row headers and values are row bounding boxes. The columns in the image are: ['Golfer', 'Country', 'Wins', 'Match Play', 'Championship', 'Invitational', 'Champions']. The rows in the image start with: ['Golfer', 'Woods, TigerTiger Woods', 'Ogilvy, GeoffGeoff Ogilvy', 'Clarke, DarrenDarren Clarke', 'Els, ErnieErnie Els', 'Mahan, HunterHunter Mahan', 'Mickelson, PhilPhil Mickelson', 'Poulter, IanIan Poulter'].

# USER IMAGE stored in image_1, as PIL image.
Now please generate only THOUGHT 0 and ACTION 0 in RESULT. If no action needed, also reply with ANSWER: <your answer> Please extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE:

# RESULT #:
```

Thought + Action 0

```
THOUGHT 0: To find the number of wins Els had, I need to focus on the 'Wins' column and the row for 'Els, ErnieErnie Els'.

ACTION 0:
```python
image_with_focused_columns = focus_on_columns_with_draw(image_1, ["Wins"], columns_bbox)
image_with_focused_rows = focus_on_rows_with_draw(image_with_focused_columns, ["Els, ErnieErnie Els"], rows_bbox)
display(image_with_focused_rows)
```
```

Observation 0

Execution success. The output is as follows.

Golfer	Country	Wins	Match Play	Championship	Invitational	Champions
Woods, TigerTiger Woods	United States	18	**3:** 2003, 2004, 2008	**7:** 1999, 2002, 2003, 2005, 2006, 2007, 2013	**8:** 1999, 2000, 2001, 2005, 2006, 2007, 2009, 2013	—
Ogilvy, GeoffGeoff Ogilvy	Australia	3	**2:** 2006, 2009	**1:** 2008	—	—
Clarke, DarrenDarren Clarke	Northern Ireland	2	**1:** 2000	—	**1:** 2003	—
Els, ErnieErnie Els	South Africa	2	—	**2:** 2004, 2010	—	—
Mahan, HunterHunter Mahan	United States	2	**1:** 2012	—	**1:** 2010	—
Mickelson, PhilPhil Mickelson	United States	2	—	**1:** 2009	—	**1:** 2009
Poulter, IanIan Poulter	England	2	**1:** 2010	—	—	**1:** 2012

Answer the question tell me the number of wins els had. You can turn the table image into text and answer with step of thinking.
Reply with ANSWER: <your answer>
Please extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE.

Thought + Action 1

ANSWER: The number of wins Els had is 2. FINAL ANSWER: 2. TERMINATE

Prompts for Chart Problems:

SYSTEM PROMPT – CHARTS

You are a helpful multimodal AI assistant. [MORE INSTRUCTIONS ...]
For each turn, you should first do a "THOUGHT", based on the images and text you see. If you think you get the answer to the intial user request, you can reply with "ANSWER: <your answer>" and ends with "TERMINATE".

Initial Prompt + Request

```
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 chart 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.
If you are dealing with a vertical bar chart figure, you should determine which are the relevant x values to the question, and specify them in a python list. You should use the given x value names.
If you are dealing with a horizontal bar chart figure, you should also determine which are the relevant y values to the question, and specify them in a python list. You should use the given y value names.
Below are the tools in tools.py:
```python
def focus_on_x_values_with_mask(image, x_values_to_focus_on, all_x_values_bounding_boxes):
    """
    This function is useful when you want to focus on some specific x values in the image.
    It does this by masking out the x values that are not needed.
    This function is especially useful for vertical bar charts.
    For example, you can focus on the x values in a chart that are relevant to your analysis and ignore the rest.
    Return the masked image.

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

    Returns:
        image_with_focused_x_values (PIL.Image.Image): the image with specified x values focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_x_values = focus_on_x_values(image, ["2005", "2006"], {"2005": {'x1': 0.1, 'y1': 0.1, 'x2': 0.3, 'y2': 0.9}, "2006": {'x1': 0.4, 'y1': 0.1, 'x2': 0.6, 'y2': 0.9}, "2007": {'x1': 0.7, 'y1': 0.1, 'x2': 0.9, 'y2': 0.9}})
        display(image_with_focused_x_values)
    """

def focus_on_y_values_with_mask(image, y_values_to_focus_on, all_y_values_bounding_boxes):
    """
    This function is useful when you want to focus on some specific y values in the image.
    It does this by masking out the y values that are not needed.
    This function is especially useful for horizontal bar charts.
    For example, you can focus on the y values in a chart that are relevant to your analysis and ignore the rest.
    Return the masked image.

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

    Returns:
        image_with_focused_y_values (PIL.Image.Image): the image with specified y values focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_y_values = focus_on_y_values(image, ["0", "10"], {"0": {'x1': 0.1, 'y1': 0.1, 'x2': 0.9, 'y2': 0.15}, "10": {'x1': 0.1, 'y1': 0.2, 'x2': 0.9, 'y2': 0.5}, "20": {'x1': 0.1, 'y1': 0.6, 'x2': 0.9, 'y2': 0.9}})
    """

def focus_on_x_values_with_draw(image, x_values_to_focus_on, all_x_values_bounding_boxes):
    """
    This function is useful when you want to focus on some specific x values in the image.
    It does this by drawing a red box around the x values that need to be focused on.
    This function is especially useful for vertical bar charts.
    For example, you can focus on the x values in a chart that are relevant to your analysis.
    Return the masked image.

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

    Returns:
        image_with_focused_x_values (PIL.Image.Image): the image with specified x values focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_x_values = focus_on_x_values(image, ["2005", "2006"], {"2005": {'x1': 0.1, 'y1': 0.1, 'x2': 0.3, 'y2': 0.9}, "2006": {'x1': 0.4, 'y1': 0.1, 'x2': 0.6, 'y2': 0.9}, "2007": {'x1': 0.7, 'y1': 0.1, 'x2': 0.9, 'y2': 0.9}})
        display(image_with_focused_x_values)
    """
```
```

```
def focus_on_y_values_with_draw(image, y_values_to_focus_on, all_y_values_bounding_boxes):
    """
    This function is useful when you want to focus on some specific y values in the image.
    It does this by drawing a red box around the y values that need to be focused on.
    This function is especially useful for horizontal bar charts.
    For example, you can focus on the y values in a chart that are relevant to your analysis.
    Return the masked image.

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

    Returns:
        image_with_focused_y_values (PIL.Image.Image): the image with specified y values focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_y_values = focus_on_y_values(image, ["0", "10"], {"0": {'x1': 0.1, 'y1': 0.1, 'x2': 0.9, 'y2': 0.15}, "10": {'x1': 0.1, 'y1': 0.2, 'x2': 0.9, 'y2': 0.5}, "20": {'x1': 0.1, 'y1': 0.6, 'x2': 0.9, 'y2': 0.9}})
    """

def focus_on_x_values_with_highlight(image, x_values_to_focus_on, all_x_values_bounding_boxes):
    """
    This function is useful when you want to focus on some specific x values in the image.
    It does this by adding light transparent red highlight to the x values that need to be focused on.
    This function is especially useful for vertical bar charts.
    For example, you can focus on the x values in a chart that are relevant to your analysis.
    Return the masked image.

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

    Returns:
        image_with_focused_x_values (PIL.Image.Image): the image with specified x values focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_x_values = focus_on_x_values(image, ["2005", "2006"], {"2005": {'x1': 0.1, 'y1': 0.1, 'x2': 0.3, 'y2': 0.9}, "2006": {'x1': 0.4, 'y1': 0.1, 'x2': 0.6, 'y2': 0.9}, "2007": {'x1': 0.7, 'y1': 0.1, 'x2': 0.9, 'y2': 0.9}})
        display(image_with_focused_x_values)
    """

def focus_on_y_values_with_highlight(image, y_values_to_focus_on, all_y_values_bounding_boxes):
    """
    This function is useful when you want to focus on some specific y values in the image.
    It does this by adding light transparent red highlight to the y values that need to be focused on.
    This function is especially useful for horizontal bar charts.
    For example, you can focus on the y values in a chart that are relevant to your analysis.
    Return the masked image.

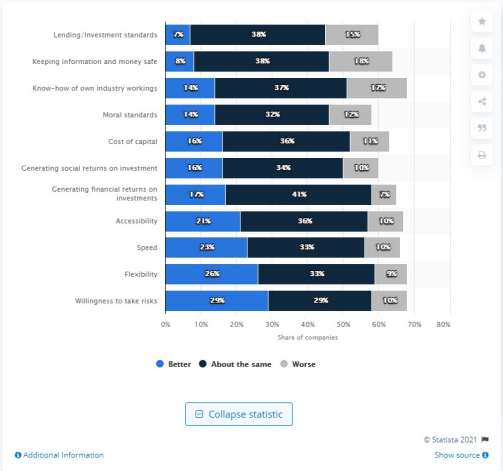
    Args:
        image (PIL.Image.Image): the input image
        y_values_to_focus_on (List[str]): a list of y values to focus on.
        all_y_values_bounding_boxes (Dict[Dict]): a dictionary of bounding boxes for all y values in the image. key is y value and value is the bounding box of that y value.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

    Returns:
        image_with_focused_y_values (PIL.Image.Image): the image with specified y values focused on

    Example:
        image = Image.open("sample_img.jpg")
        image_with_focused_y_values = focus_on_y_values(image, ["0", "10"], {"0": {'x1': 0.1, 'y1': 0.1, 'x2': 0.9, 'y2': 0.15}, "10": {'x1': 0.1, 'y1': 0.2, 'x2': 0.9, 'y2': 0.5}, "20": {'x1': 0.1, 'y1': 0.6, 'x2': 0.9, 'y2': 0.9}})
    """
```
```
# 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 jupyter notebook code block) 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.
Your visual ability is not perfect, so you should use these tools to assist you in reasoning about the images.
The jupyter notebook has already executed the following code to import the necessary packages:
```python
from PIL import Image
from IPython.display import display
from tools import *
```

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 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.
7. If all the x values are relevant to the question for a vertical bar chart, you do not need to focus on any specific x values. You should directly answer the question based on the original image.
8. If all the y values are relevant to the question for a horizontal bar chart, you do not need to focus on any specific y values. 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.

[four in-context examples here]

USER REQUEST #:
```




```
What is the average of blue bar above 20 value?

# USER Bounding Box Info: x_values_bbox, storing x values and coordinates. y_values_bbox, storing x values and coordinates. x_intervals_bbox, storing x intervals and coordinates. y_intervals_bbox, storing y intervals and coordinates. The x values in the image are: []. The y values in the image are: ['Lending/Investment standards', 'Keeping information and money safe', 'Know-how of own industry workings', 'Moral standards', 'Cost of capital', 'Generating social returns on investment', 'Generating financial returns on', 'Accessibility', 'Speed', 'Flexibility', 'Willingness to take risks']. The x intervals in the image are: []. The y intervals in the image are: [].

# USER IMAGE stored in image_1, as PIL image.

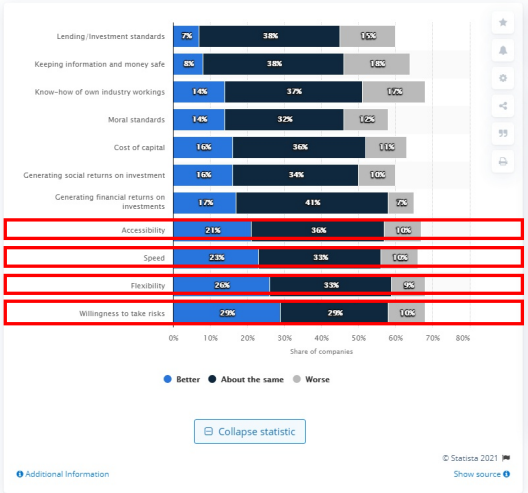
Now please generate only THOUGHT 0 and ACTION 0 in RESULT. If no action needed, also reply with ANSWER: <your answer> Please extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE:

# RESULT #:
```

Thought + Action 0

```
THOUGHT 0: This is a horizontal bar chart image. I need to focus on the y values where the blue bar is above 20 and calculate the average of those values.

ACTION 0:
```python
image_with_focused_y_values = focus_on_y_values_with_draw(image_1, ["Accessibility", "Speed", "Flexibility", "Willingness to take risks"], y_values_bbox)
display(image_with_focused_y_values)
```
```

Observation 0

Execution success. The output is as follows.



Answer the question What is the average of blue bar above 20 value? You can turn the table image into text and answer with step of thinking.
Reply with ANSWER: <your answer>
Please extract the final answer in FINAL ANSWER: <final answer> and ends with TERMINATE.

Thought + Action 1

ANSWER: The blue bars above 20 are for "Accessibility" (21%), "Speed" (23%), "Flexibility" (26%), and "Willingness to take risks" (29%). To find the average:

$$\text{Average} = \frac{21 + 23 + 26 + 29}{4} = \frac{99}{4} = 24.75$$

FINAL ANSWER: 24.75. TERMINATE

D FINETUNE DETAILS

D.1 REFOCUS DATASET STATISTICS

The detailed dataset statistics are demonstrated in Table 6. One example input data can be viewed in Listing 1.

	Horizontal Bar	Vertical Bar	Total
QA Data Masry et al. (2022)	4,990	10,069	15,059
REFOCUS Data	4,722	9,622	14,344
w/ Editing	4,220	8,599	12,819

Table 6: **Detailed statistics about REFOCUS Data.** We count the total number of our training cases, and the ones with visual editing.

D.2 FINETUNE EXPERIMENT DETAILS

For the fine-tuning experiments in Section 5, we use 8 NVIDIA RTX A6000 GPUs (48GB RAM per GPU), following the Phi-3.5-vision github instructions[4]. We adopt full training, and iterate through the hyperparameter searching space to find the best performing set to report as our result score. The hyperparameter search focuses specifically on (1)learning rate, (2) epoch number, (3) whether to include edited image in training input, with learning rate ranging in (5e-5 5e-6 5e-4 1e-4 1e-5 1e-6 5e-7), epoch ranging in (1 2), and all other parameters set to default values. In Table 7, we present detailed hyper-parameter configurations for results in Table 3.

	w/ default QA Data	w/ REFOCUS VCoT	w/ REFOCUS CoT
data type	bf16	bf16	bf16
batch size	64	64	64
learning rate	5×10^{-7}	1×10^{-6}	5×10^{-6}
epoch number	2	2	2
include edited image in input	No	No	Yes

Table 7: Hyper-parameter settings for our best fine-tuned models.

D.3 SFT RESULT ANALYSES

Looking at the SFT results in Table 3, we see a consistent improvement on the chart problems using REFOCUS data. If we use the de facto QA data to finetune, the chain of thought ability will be impaired, comparing to the original model. Also, the improvement on Vertical Bar problems reaches 5.4%, which is quite large for 14k training data.

E SFT QUALITATIVE EXAMPLES

In this section, we show some example output from the Phi-3.5-vision model finetuned on REFOCUS visual CoT data in Figure 8. Notice that the model output are in text formats with bounding box coordinates for the areas to focus on. We draw these areas in red boxes for illustration purposes. The examples are selected from the Horizontal Bar set and Vertical Bar set.

[4] https://github.com/microsoft/Phi-3CookBook/blob/main/md/04.Fine-tuning/FineTuning_Vision.md

```
{
    "id": "train-two_col_103562",
    "query": "As of 2021, how many championship titles had
    Ferrari won?",
    "answer": "16",
    "source": "h_bar",
    "images": ["two_col_103562.png"],
    "response0": "This is a horizontal bar chart image.
    I need to focus on the part where the y-axis value is
    /"Ferrari" to find out how many championship titles they
    have won.",
    "edited_images": ["two_col_103562/f19f2fd043444680a02764
    d66fd6b22d.png"],
    "response1": "As of 2021, Ferrari had won 16 championship
    titles.",
    "focus_areas": [
        {
            "x1": 5,
            "y1": 38,
            "x2": 795,
            "y2": 72
        }],
    "vcot_input": "This is a horizontal bar chart image. I
    need to focus on the part where the y-axis value is
    "Ferrari" to find out how many championship titles they
    have won. The areas to focus on in the image have
    bounding box coordinates: [{"x1": 5, "y1": 38, "x2": 795,
    "y2": 72}]. Looking at these areas, As of 2021, Ferrari
    had won 16 championship titles."
}
```


Listing 1: Example REFOCUS data input for SFT experiment. The “response0” and “response1” are GPT-4o outputs that generate codes and answer based on the edited image respectively. We use “vcot_input” as the target answer.

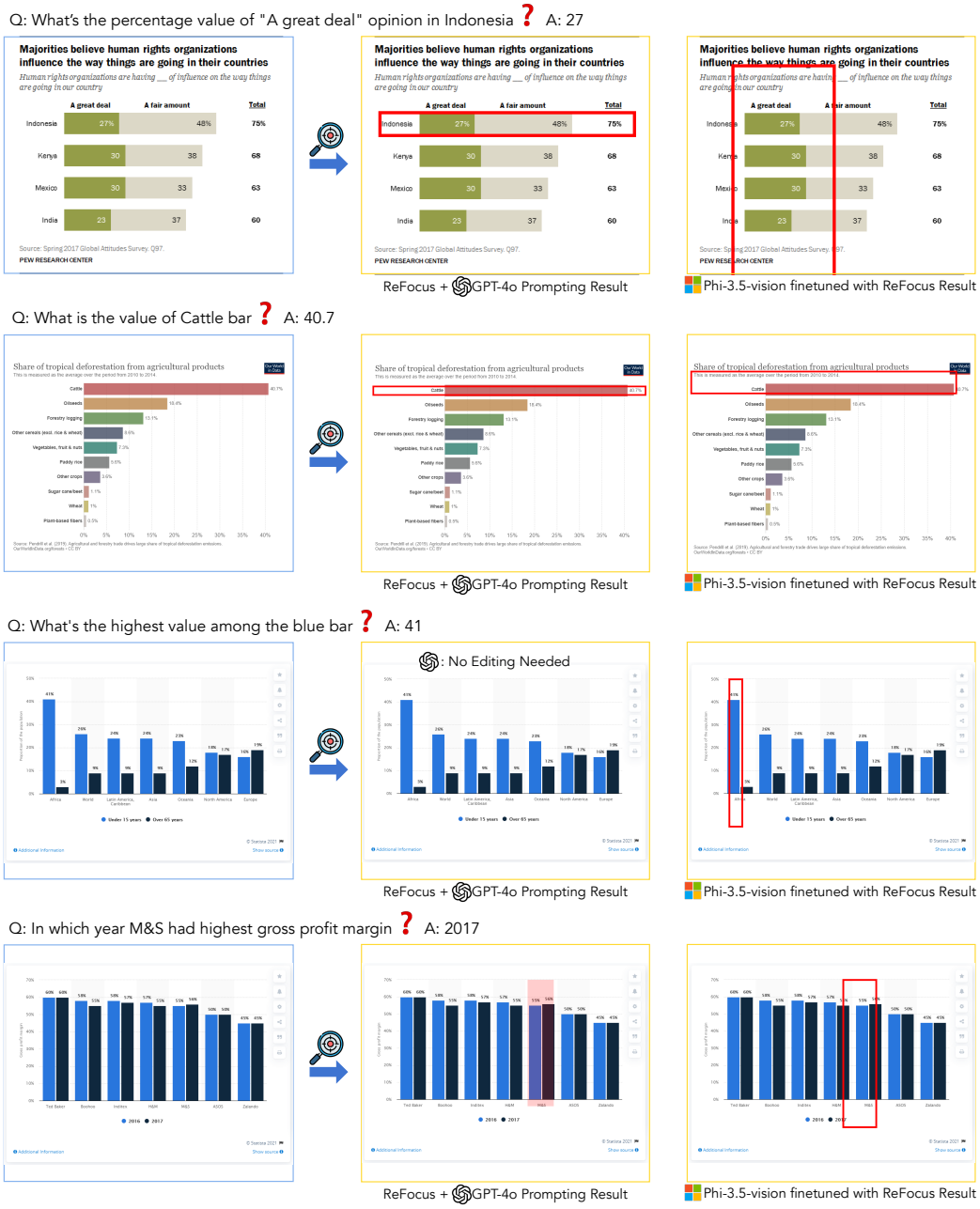


Figure 8: **Phi-3.5-vision finetuned with REFOCUS visual chain of thought data** outputs the areas to focus on. For illustration purposes, we draw these areas in red boxes, and compare with the REFOCUS + GPT-4o prompting output.