Supplementary Material for ToolVQA: A Dataset for Multi-step Reasoning VQA with External Tools

Shaofeng Yin Ting Lei Yang Liu*
Wangxuan Institute of Computer Technology, Peking University
yin_shaofeng@stu.pku.edu.cn {ting_lei, yangliu}@pku.edu.cn

In this supplementary material, we provide more details of the paper. In Sec. 1, we introduce the specific functionalities and design principles of the ToolVQA toolset. In Sec. 2, we provide the pseudocode of the LCS-based example matching algorithm that is applied in ToolEngine. In Sec. 3, we provide more user study details, including the composition of our users, the sampling process of initial user logs, and methods for data quality evaluation metrics. In Sec. 4, we provide more experiment details, including the training pipeline, training hyperparameters, evaluation setting, and evaluation metrics. In Sec. 5, we fine-tune a new Qwen2-VL-7B model using the ToolVQA dataset and compare its performance with that of the recent MM-Traj [7] model. In Sec. 6, we provide the detailed prompts used in our pipeline. In Sec. 7, we provide some demonstrations of our real-world examples used to prompt ToolEngine, and some examples of ToolVQA's test set.

1. Toolset

Tab. 1 shows the details of our toolset. We adopt an open-sourced library AgentLego [10] to build our toolset and adopt Lagent [18] framework to let LFM-based tool agents interact with the tools.

Although our toolset is not very large, these tools are all selected to address LFM's shortcomings in certain aspects, such as external knowledge acquisition, text recognition, image generation, etc. Unlike previous works [13, 14] that set up a tool for each sub-task, our tools have diverse capabilities and strong generalization. For example, Google Search can search for external knowledge such as news, historical events, data, public information, academic papers, etc. As we have discussed in Sec.1 of the main paper, binding single-function tools to specific types of problems will turn the tool usage task into a query pattern recognition task, which does not allow LFMs to gain an intrinsic understanding of the tool affordance and functionality. In contrast, Our highly generalized tools contain countless possible argument combinations. Using them well requires a comprehensive understanding of the tools, scenarios, and queries, which is more similar to human tool use.

In order to ensure that the controller can see the image during the whole process of asking questions, we need to send the image to each conversation of the controller. However, this requires a high cost. To improve efficiency, we change it to fix the first tool to ImageCaption/OCR when asking questions, so that the controller can understand the overall information of the image in the first step. At the same time, this can also provide the necessary basic information for LFMs when answering queries. The ablation experiment in Sec.4.3 of the main paper shows that Caption is important for the fine-tuned model to answer queries.

2. Algorithm

Algorithm 1 Select Top-k Examples by LCS

function LCS(A, B)
  for i from 1 to $|A|$ **do**
    for j from 1 to $|B|$ **do**
      if $A[i-1] == B[j-1]$ **then**
        $\texttt{dp}[i][j] = \texttt{dp}[i-1][j-1] + 1$
      else
        $\texttt{dp}[i][j] = \max(\texttt{dp}[i-1][j], \texttt{dp}[i][j-1])$
      end if
    end for
  end for
  return $\texttt{dp}[|A|][|B|]$
end function

Initialize an empty list `LCS_values`
Initialize a 2D array `dp` of size $(|A|+1) \times (|B|+1)$ with all values 0
for each $\mathcal{P}_{ej}$ in $\mathcal{P}_e$ **do**
  Calculate `LCS_length` = LCS($\mathcal{P}_i$, $\mathcal{P}_{ej}$)
  Append $(\mathcal{P}_{ej}, \texttt{LCS\_length})$ to `LCS_values`
end for
return Ret = Top-k(`LCS_values`)

*Corresponding author

Tool Name	Description	Model	Input	Output
ImageCaption	Describe an image	ChatGPT-4o-latest	image	text
OCR	Recognize the text in image	PaddleOCR [6]	image	text
ObjectDetection	Detect an object in image	MM-Grounding-DINO [21]	image, object	bbox
Calculator	Calculate a math expression	Python.math	expression	number
TextToImage	Generate an image based on text	Stable-Diffusion-v1.5 [15]	text	image
RegionDescription	Describe attribute for a region	ChatGPT-4o-latest	image, bbox, attribute	text
Plot	Plot a diagram	Python.matplotlib [9]	python code	image
ItemCount	Count the number of an object	ChatGPT-4o-latest	image, object	number
GoogleSearch	Search external knowledge	Google Serper API	query	text
DrawBox	Draw a box on image	Python.pillow	image, bbox	image

Table 1. Details of our toolset.

Algorithm 1 shows the pseudocode for our LCS-based example matching algorithm introduced in Sec.3.2 of the main paper. The LCS algorithm is a classic method for measuring the similarity between two ordered lists. Unlike similarity calculation modules trained using neural networks, LCS offers a simple yet effective solution that ensures both accuracy and efficiency. Moreover, LCS is well-suited for handling longer trajectories in future application scenarios, maintaining its robustness and precision even as complexity increases.

3. User Study Details

We randomly select 10 real-world users from universities, including both students and professors from different disciplines. The participants spanned five fields: Mathematics, Computer Science, Economics, Chinese Language, and Art, with two users from each discipline.

3.1. Toolset Selection

We invite users to document 15 common tool-use scenarios they frequently encounter in their work, along with the corresponding trajectories. We then merged functionally similar tools and selected the 10 most frequently occurring tools as our final toolset. The detailed frequency distribution is shown in Fig. 1. Subsequently, human experts discussed and consolidated similar scenarios and trajectories. Through this process, we refined the initial 150 scenarios into a final set of 34 representative examples. These examples cover all 10 tools and most reasonable tool combinations, with only some differences in the number of tool usages (e.g., counting the number of different objects, iterative searching for external knowledge) that need to be compensated by our LCS-based example matching algorithm. These queries go through multiple rounds of iteration and expert discussion, aiming to meet the following requirements: (1) trajectories are necessary to answer queries; (2) understanding image is necessary to answer queries; (3) all queries cover the vast majority of reasonable trajectories; and (4) queries are helpful to real human life.

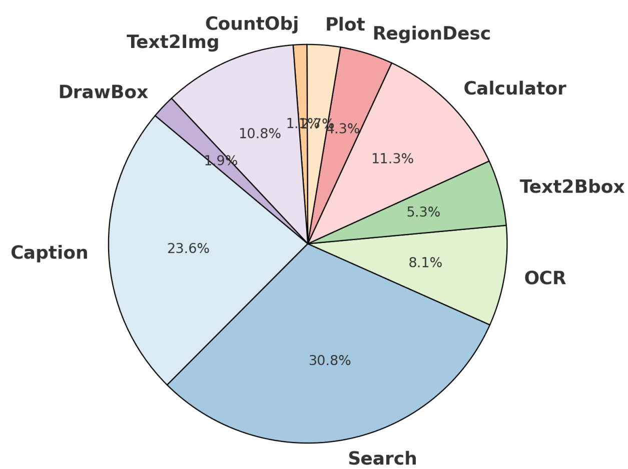

Figure 1. Real-world users tool frequency.

3.2. Evaluation Metrics

For each dataset, we randomly sampled 100 samples and invite users to annotate them. To calculate the metrics of Acc., Corr., and Nec.. on our dataset, we ask users to individually review each image-question-answer pair along with its corresponding trajectory. For deterministic-answer metrics (Acc., Nec..), we design a binary (Yes/No) selection. For degree-based metrics (Corr.), we use a 1-10 scoring scale to capture nuanced differences. To calculate the Reasoning

Complexity (R.C.), we ask users to write a tree-of-thought (ToT) [20] to solve the query for each sample. Then we calculate the mean depth of the samples' ToTs.

4. Experiment Details

4.1. Training

We use XTuner [5] as our training framework. We finetune LLaVA-7B using LoRA [8] algorithm, with $\text{batch\_size} = 2, \text{learning\_rate} = 2e-4$ on 4xGTX3090 GPUs for 4000 epochs.

4.2. Evaluation

We use OpenCompass [4] as our evaluation framework. On our ToolVQA test set, we follow all the metrics used in GTA [19] that are mentioned in Sec.4.1 of the main paper. When comparing the output with the ground truth, we follow GTA to divide the ground truth into a whitelist and a blacklist for matching. Fig. 3 demonstrates some examples of our test set. The whitelist and blacklist are manually labeled and can generally accept most of the correct answers. We only evaluate 1622 samples with text answers under end-to-end mode due to the lack of generation metrics. We evaluate all 2550 samples under step-by-step mode.

On the various out-of-distribution (OOD) benchmarks (TextVQA [16], TallyQA [1], InfoSeek [3] and GTA [19]), we use their own evaluation metrics. Since the time required for tool-use VQA is significantly longer than that of traditional VQA (the former is about 4-6 times that of the latter, limited by the running speed of the tool itself), we follow previous works [2, 17] to randomly sample 1,000 examples from their test set for testing to ensure fairness and accuracy as much as possible. In addition, we notice that LLaVA [11] provide additional OCR tokens in each image to help VLM answer when testing TextVQA, and we believe that this cannot accurately evaluate VLM's text recognition ability, so we follow previous work [12] and remove these OCR tokens in the evaluation, which made our test results significantly lower than the results in [11].

5. Fine-tuning Other LFMs

To comprehensively evaluate the data quality of ToolVQA, we compare it with the recent work MM-Traj [7], which is also a fine-tuning dataset designed for LFM-based tool agents but does not incorporate ToolEngine's multi-step reasoning optimizations (DFS + Example matching). To make a fair comparison with MM-Traj, we fine-tune Qwen2-VL-7B on ToolVQA and conduct evaluations on GTA [19].

5.1. Training Setting

We use swift [22] as our training framework. We finetune Qwen2-VL-7B using LoRA [8] algorithm, with $\text{batch\_size} = 4, \text{learning\_rate} = 1e-4$ on 4xA100 GPUs for 1000 epochs.

5.2. Main Results

The main results are presented in Tab. 2. It shows that models fine-tuned on ToolVQA significantly outperform those fine-tuned on MM-Traj when evaluated on a public third-party test set. This highlights the higher quality of ToolVQA compared to MM-Traj and demonstrates the clear impact of ToolEngine in enhancing data quality. It is worth noticing that GTA includes unseen tools (`MathOCR`, `Solver`, `ImageStylization`, `AddText`) that are not present in our training toolset. Despite this, our fine-tuned model demonstrates strong generalization on unseen tools. Specifically, it can correctly execute these unseen tools (high Inst..), but select them less (lower Tool.), instead attempting to solve problems using seen tools as substitutes.

Model	Acc.	Inst.	Tool.	UnsI.	UnsT.
Qwen2-VL-7B	42.3	65.2	44.9	64.8	42.2
MM-Traj	53.9	84.3	64.6	80.6*	**61.4***
Tuned Qwen2	**66.5**	**90.7**	**72.1**	**83.2**	60.5

Table 2. Results on GTA. **Inst.**: tool success rate; **Tool.**: tool selection accuracy; **UnsI./UnsT.**: metrics on unseen tools. *MM-Traj has seen all the tools during training, while our model has not.

5.3. Ablation Study

To further analyze the factors affecting tool generalization, we conduct two ablations (Tab. 3) by varying reasoning steps and toolset size. Results show that reducing reasoning steps and shrinking the toolset both impair generalization.

Ablation	Setting	Acc.	UnsI.	UnsT.
Reasoning Steps	≤ 2	57.5	68.1	45.3
	≤ 4	64.2	78.4	55.0
	Unlimited	66.5	83.2	60.5
Toolset Size	Small (4)	58.7	76.4	52.5
	Medium (7)	62.0	79.5	55.8
	Large (10)	66.5	83.2	60.5

Table 3. Ablation results on GTA. We keep the same number of training samples across all settings, and train with equal FLOPs.

6. Prompts

We provide the exact prompts that we used in our pipeline.

6.1. ToolEngine

In the ToolEngine, we use the LFM-based controller (in the main paper Fig.3) to construct the multi-step tool-use VQA samples. The controller has three main purposes: (1) select which tool to use in the next step; (2) explain why we select that tool (to obtain the chain-of-thought of the sample); (3) come up with the final question and answer of the sample. We list the prompt examples of each part below:

(1) Select the next tool:

```
You are a smart tool selector. I will
   provide you with some information
   extracted from an image, along with
   a list of available tool options for
    the next steps. Please choose the
   most suitable tool to obtain more
   information or generate a new image.

The tool options are as follows:
{options}

Your response should consist of two
   parts:

1. **Thought:** Explain your reasoning
   behind how you decide to choose the
   next tool.
2. **Choice:** Provide the specific
   tool you have selected.

Here are some examples to help you
   understand the task.

{examples}

Now that you understand the approach
   and format for selecting tools, I
   will provide you with the necessary
   information. Please choose the next
   tool using the same format.

Information:
{context}
```

(2) Explain the reason:

````
You are a smart information processor.
   I will provide you with a problem,
   an answer, and a process for solving
    the problem using different tools.
   Your task is to describe the
   thinking behind solving the problem,
    specifically explaining the purpose
    of using each tool.

Your response should include several
   lines, one for each tool, and each
   line should contain two parts:

1. **Tool Name:** The name of the tool
   used.
2. **Thought:** Explain the purpose of
   using the tool, including the
   information you expect to get from
   it to solve the problem.

Here are some examples to guide you:

Example 1:
```
{example1}
```

Example 2:
```
{example2}
```

Now that you understand the format, I
   will provide you with the
   information. Please generate your
   response accordingly.

Question: {question}

Solving Process:
{context}

Answer: {answer}
````

(3) Come up with the final question and answer:

```
You are a question generator that
   creates valuable queries based on
   extracted information from a tool
   process. The task is to formulate
   questions about the image data that
   meet the following conditions:

1. The answer must be the result
   returned by the LAST tool call. If
   the answer is a long sentence, you
   need to summarize it into a single
   word or phrase.

2. The question should address a
   scenario that can occur in real-life
    situations and meet practical needs
   .

3. It must be solvable through the tool
    call, avoiding trivial or overly
```

```
   complex inquiries unrelated to the
   data.

4. The answer requires analyzing the
   image or information extracted from
   the image. The direct content of the
    image must not appear in the
   question. Instead, refer to the
   image as "this," "image," or "
   picture."

Your response should consist of three
   parts:

1. **Thought:** Your reasoning for
   generating the question.
2. **Question:** The question you asked
   .
3. **Answer:** The answer to the
   question, either text or a picture
   generated by the last tool.

Here are some examples to help clarify:

{examples}

Now, I will provide you with the
   process information. Please create
   your answer accordingly.

Process:
{context}
```

6.2. Fine-tuning

An example of the fine-tuning prompts is shown below: (The question of this example is “In the image, Where can someone buy the soda shown in this image?”)

```
[system] You are an assistant who can
   utilize external tools.
[
   {
      "name": "ImageDescription",
      "description": "A useful tool
         that returns a brief
         description of the input image
         .",
      "inputs":
      [{
         "type": "image",
         "name": "image",
         "description": null,
         "optional": false,
         "default": null, "filetype":
            null
      }],
      "outputs":
      [{
         "type": "text",
         "name": null,
         "description": null,
         "optional": false,
         "default": null, "filetype":
            null
      }]
   },
   {
      ... (other tools)
   },
   ...
]
To use a tool, please use the following
    format:
```
Thought:Think what you need to solve,
   do you need to use tools?
Action:the tool name, should be one of
   [["ImageDescription", ... (other
   tool names)]]
Action Input:the input to the action
```
The response after utilizing tools
   should using the following format:
```
Tool Response:the results after call
   the tool.
```
If you already know the answer, or you
   do not need to use tools,please
   using the following format to reply:
```
Thought:the thought process to get the
   final answer
Final Answer:final answer
```
Begin!
<image>

[user] Where can someone buy the soda
   shown in this image?

[assistant] Thought:To answer the
   question, we first need to use the
   ImageDescription tool to identify
   the soda brand and flavor in the
   image, which is important for
```

```
    searching its availability.
Action:ImageDescription
Action Input:
```
```
{"image": <image>}
```

[user] Tool Response:The image shows a
    can of Postobon apple-flavored soda
    ("Manzana/Postobon") placed on a
    light blue surface, possibly a metal
     table. In the background, there is
    a colorful tiled mural featuring
    vibrant, geometric floral designs
    and fruits. The can is pink and red
    with the brand name prominently
    displayed in the center. The mural
    adds a lively and artistic vibe to
    the setting.

[assistant] Thought:...

[user] Tool Response:...

... (action-observation loops)

7. Demostrations

Fig. 2 shows some of the real-world examples we used in the ToolEngine pipeline. Fig. 3 demonstrates the data quality of the ToolVQA's test set. These data satisfy our definition of real scenarios and queries, and each sample requires more than one step of reasoning to solve.

References

[1] Manoj Acharya, Kushal Kafle, and Christopher Kanan. Tallyqa: Answering complex counting questions, 2018. 3

[2] Lluis Castrejon, Thomas Mensink, Howard Zhou, Vittorio Ferrari, Andre Araujo, and Jasper Uijlings. Hammr: Hierarchical multimodal react agents for generic vqa, 2024. 3

[3] Yang Chen, Hexiang Hu, Yi Luan, Haitian Sun, Soravit Changpinyo, Alan Ritter, and Ming-Wei Chang. Can pre-trained vision and language models answer visual information-seeking questions?, 2023. 3

[4] OpenCompass Contributors. Opencompass: A universal evaluation platform for foundation models. https://github.com/open-compass/opencompass, 2023. 3

[5] XTuner Contributors. Xtuner: A toolkit for efficiently fine-tuning llm. https://github.com/InternLM/xtuner, 2023. 3

[6] Yuning Du, Chenxia Li, Ruoyu Guo, Xiaoting Yin, Weiwei Liu, Jun Zhou, Yifan Bai, Zilin Yu, Yehua Yang, Qingqing Dang, and Haoshuang Wang. Pp-ocr: A practical ultra lightweight ocr system, 2020. 2

[7] Zhi Gao, Bofei Zhang, Pengxiang Li, Xiaojian Ma, Tao Yuan, Yue Fan, Yuwei Wu, Yunde Jia, Song-Chun Zhu, and Qing Li. Multi-modal agent tuning: Building a vlm-driven agent for efficient tool usage, 2025. 1, 3

[8] 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, 2021. 3

[9] J. D. Hunter. Matplotlib: A 2d graphics environment. *Computing in Science & Engineering*, 9(3):90–95, 2007. 2

[10] InternLM. Agentlego: Enhance llm agents with rich tool apis. https://github.com/InternLM/agentlego, 2024. 1

[11] Haotian Liu, Chunyuan Li, Yuheng Li, and Yong Jae Lee. Improved baselines with visual instruction tuning, 2024. 3

[12] 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, 2024. 3

[13] Zixian Ma, Weikai Huang, Jieyu Zhang, Tanmay Gupta, and Ranjay Krishna. m&m's: A benchmark to evaluate tool-use for multi-step multi-modal tasks. *EECV 2024*, 2024. 1

[14] Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, Dahai Li, Zhiyuan Liu, and Maosong Sun. Toolllm: Facilitating large language models to master 16000+ real-world apis, 2023. 1

[15] Robin Rombach, Andreas Blattmann, Dominik Lorenz, Patrick Esser, and Björn Ommer. High-resolution image synthesis with latent diffusion models, 2021. 2

[16] Amanpreet Singh, Vivek Natarajan, Meet Shah, Yu Jiang, Xinlei Chen, Dhruv Batra, Devi Parikh, and Marcus Rohrbach. Towards vqa models that can read, 2019. 3

[17] Dídac Surís, Sachit Menon, and Carl Vondrick. Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 11888–11898, 2023. 3

[18] Lagent Developer Team. Lagent: Internlm a lightweight open-source framework that allows users to efficiently build large language model(llm)-based agents. https://github.com/InternLM/lagent, 2023. 1

[19] Jize Wang, Zerun Ma, Yining Li, Songyang Zhang, Cailian Chen, Kai Chen, and Xinyi Le. Gta: A benchmark for general tool agents, 2024. 3

[20] Shunyu Yao, Dian Yu, Jeffrey Zhao, Izhak Shafran, Thomas L. Griffiths, Yuan Cao, and Karthik Narasimhan. Tree of thoughts: Deliberate problem solving with large language models, 2023. 3

[21] Xiangyu Zhao, Yicheng Chen, Shilin Xu, Xiangtai Li, Xinjiang Wang, Yining Li, and Haian Huang. An open and comprehensive pipeline for unified object grounding and detection, 2024. 2

[22] Yuze Zhao, Jintao Huang, Jinghan Hu, Xingjun Wang, Yunlin Mao, Daoze Zhang, Zeyinzi Jiang, Zhikai Wu, Baole Ai, Ang Wang, Wenmeng Zhou, and Yingda Chen. Swift:a scalable lightweight infrastructure for fine-tuning, 2024. 3

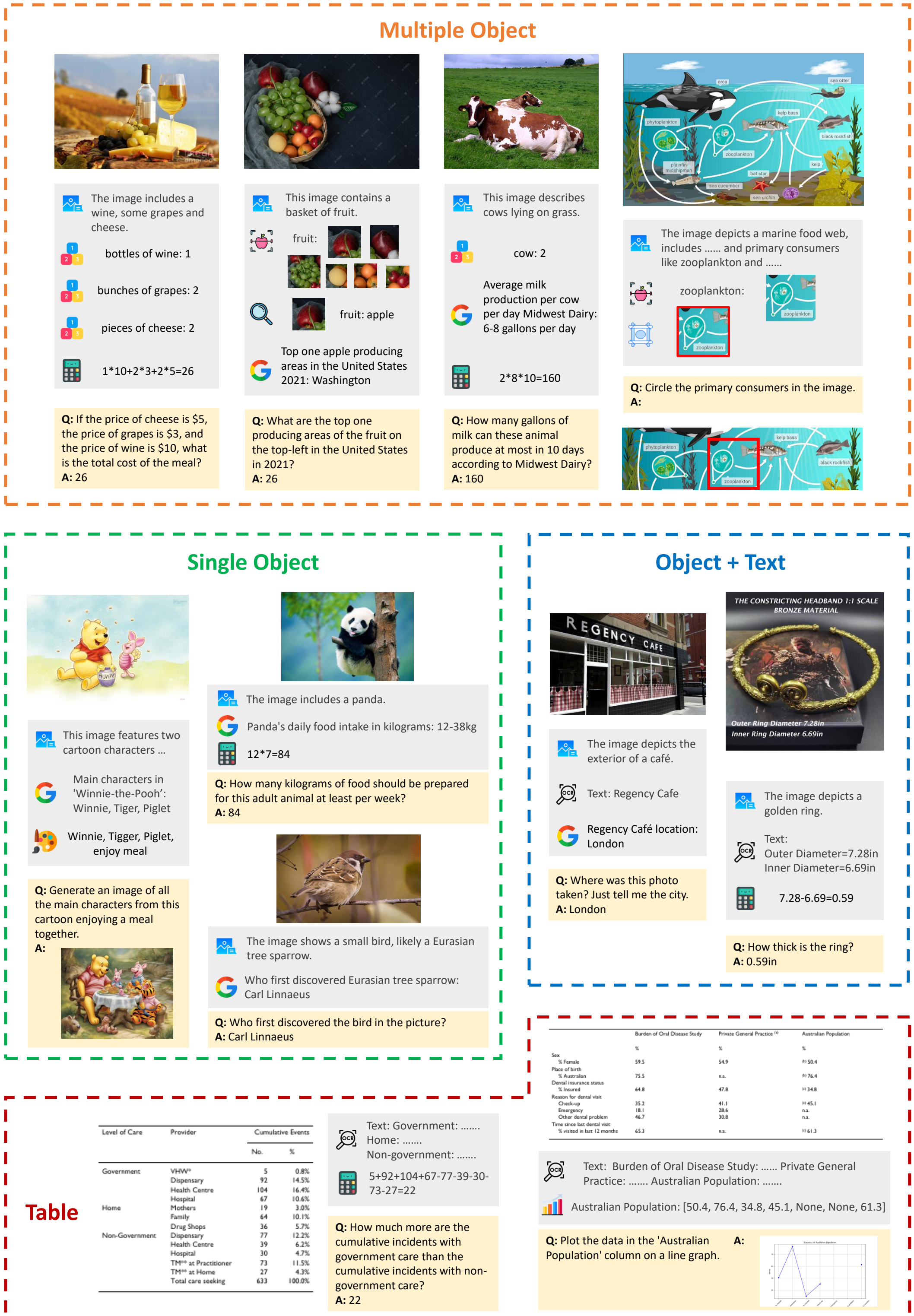

Figure 2. Demonstrations of real-world examples.

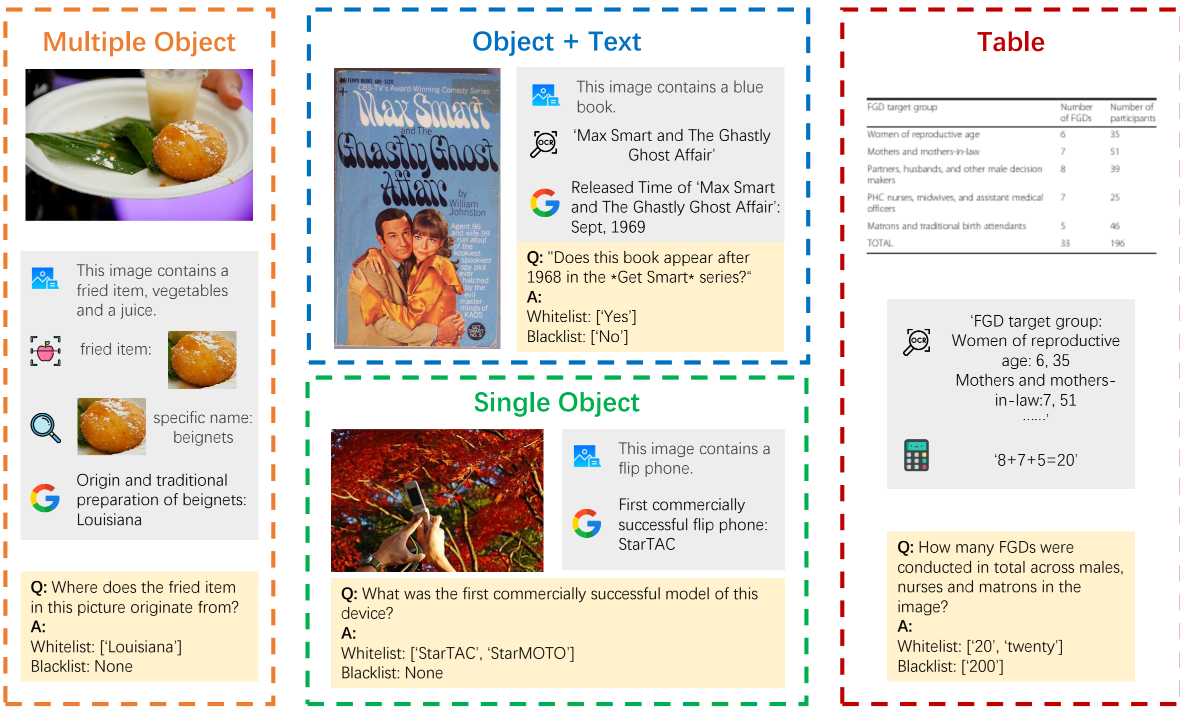

Figure 3. Demostrations of ToolVQA test set.