# SpatiaLQA: A Benchmark for Evaluating Spatial Logical Reasoning in Vision-Language Models

Supplementary Material

## A. SpatiaLQA Benchmark Details

This section provides additional details on the construction of the SpatiaLQA benchmark. Specifically, Sec. A.1 describes the data collection rules, and Sec. A.2 outlines the manual annotation and review process.

### A.1. Data Collection Rules

During data collection, we followed the rules below to ensure both diversity and category balance in the dataset:

(1) The same object is allowed to appear no more than ten times within a single scene, and its appearance frequency should be kept as low as possible across the entire dataset.

(2) The same scene (*e.g.*, the same desk, the same sofa) may appear approximately 20 times. Changes in camera angle, lighting, *etc*., do not count as a new scene.

(3) The scene layout may be modified without changing the scene itself. For example, using an office desk as the base scene, one may rearrange or replace the items on the desk (each item still appearing no more than ten times).

### A.2. Manual Annotation and Review

The SpatiaLQA dataset consists of an image directory and a JSON file. The image directory contains multiple subfolders, each corresponding to a specific scene. Subfolders follow the naming format 'sceneIndex-sceneName', such as '0001-office-1' or '0022-bedroom-2'. The prefix number indicates the global index of the scene across the entire dataset, while the suffix number denotes the index of that scene type (*e.g.*, bedroom-2 refers to the folder containing all images of the second bedroom scene).

Each sample annotation consists of four components: the question, the answer, the corresponding image path, and the associated scene category. After all annotations are completed, they are reviewed by designated annotators. During review, each sample is examined for step validity and prerequisite correctness, with outcomes marked as either 'approved' or 'rejected, with reasons provided.' The annotation–review cycle is repeated twice to ensure that all annotations are accurate and logically sound.

## B. Evaluation Details

In this section, we present the prompts, hyperparameters, and other settings used in our evaluation. Specifically, Sec. B.1 describes the prompts and hyperparameters used when generating the matching matrices with GPT-4o, and Sec. B.2 details the prompts and hyperparameters used for evaluating the various VLMs.

### B.1. The Details of Generating the Matching Matrices

The prompt used to generate the matching matrix with GPT-4o is as follows:

> You are given an image and two list of step descriptions (Ground Truth List and Predicted List), each containing multiple sentences describing actions.
>
> Your task is to compare each sentence in Ground Truth List with each sentence in Predicted List, and determine whether they describe the same action in the context of the image. Two sentences are considered the same if they refer to the same objects and the same operations with no significant difference in meaning or execution.
>
> Please output an `m × n` matrix where:
> 1. `m` is the number of steps in Ground Truth List
> 2. `n` is the number of steps in Predicted List
> 3. Each cell `[i][j]` is either 1 (Yes) or 0 (No), indicating whether sentence `i` from Ground Truth List is equivalent in meaning to sentence `j` from Predicted List
>
> Ground Truth List: {ground_truth_steps}
> Predicted List: {predicted_steps}
>
> Please output an `m × n` matrix, with each element being either 1 or 0. You only need to return a list of lists wrapped with `<ans></ans>` tags (*e.g.*, `<ans>[[0, 1], [1, 1]]</ans>`).

where {ground_truth_steps} and {predicted_steps} denote the annotation result and the VLM's prediction, respectively. Regarding hyperparameters, we set the temperature to 0.

### B.2. The Details of Evaluating the Various VLMs

For open-source VLMs, we use the corresponding HuggingFace models for local inference. The version of transformers used matches that of VLMEvalKit. For models not covered by VLMEvalKit [22], we adopt the model versions recommended on HuggingFace. In addition, all hyperpa-

rameters for open-source models follow the recommended settings provided on HuggingFace. For proprietar VLMs, we uniformly set the temperature to 0.

The prompt used for evaluating the VLMs is as follows:

> Given a scene image and a task, please provide a concise and accurate list of steps to execute the task in order.
>
> Here is an example: {example}
>
> Each step in the answer consists of two parts: the action to be performed in this step ('content') and the steps that must be completed beforehand ('precondition').
>
> Please note the following:
> 1. The example above is only for illustrating the answer format and is not related to the question you need to answer.
> 2. All objects in the image are within your reach, so no movement is necessary.
> 3. Each step can only perform one action on a single object.
> 4. If you need to remove an object, use the action 'remove' directly.
> 5. Write your answer in the format above (the answer consists of multiple steps, and each step contains a 'content' and a 'precondition'), enclosed within <ans></ans> tags.
>
> Now answer the following question based on the input image: {question}. You only need to return an answer wrapped with <ans></ans> tags, and the answer should be in JSON format (as shown in the 'answer' field of the example).

Here, '{example}' is a JSON-formatted sample, as shown below (similarly for the rest):

```
{
    "question": "Pick up the laptop",
    "answer": {
        "step1":{
            "content": "Remove the stapler from
                the top of the book",
            "precondition":[]
        },
        "step2":{
            "content": "Remove the keys from the
                top of the book",
            "precondition":[]
        },
        "step3":{
            "content": "Remove the toliet paper
                from the top of the laptop",
            "precondition":[]
        },
        "step4":{
            "content": "Remove the book from the
                top of the laptop",
            "precondition":["step1", "step2"]
        },
        "step5":{
            "content": "Pick up the laptop",
            "precondition":["step3", "step4"]
        }
    }
}
```

## C. The Details of the Baselines and RSGAR

For both the baselines and our proposed method RSGAR, the temperature of GPT-4o is fixed at 0. In this section, we also present the prompts used by the baselines and by RSGAR. Specifically, Sec. C.1 and Sec. C.2 describe the prompts for the baselines and for RSGAR, respectively.

### C.1. The Details of Baselines

The prompt used for '+ depth' is as follows:

> Given a scene image, the depth map of the image and a task, please provide a concise and accurate list of steps to execute the task in order.
>
> Here is an example: {example}
>
> Each step in the answer consists of two parts: the action to be performed in this step ('content') and the steps that must be completed beforehand ('precondition').
>
> Please note the following:
> 1. The example above is only for illustrating the answer format and is not related to the question you need to answer.
> 2. All objects in the image are within your reach, so no movement is necessary.
> 3. Each step can only perform one action on a single object.
> 4. If you need to remove an object, use the action 'remove' directly.
> 5. Write your answer in the format above (the answer consists of multiple steps, and each step contains a 'content' and a 'precondition'), enclosed within <ans></ans> tags.
>
> Now answer the following question based on the input image: {question}. You only need to return an answer wrapped with <ans></ans> tags, and the

answer should be in JSON format (as shown in the 'answer' field of the example).

The prompt used for '+ seg' is as follows:

Given a scene image, the segmentation map of the image and a task, please provide a concise and accurate list of steps to execute the task in order.

Here is an example: {example}

Each step in the answer consists of two parts: the action to be performed in this step ('content') and the steps that must be completed beforehand ('precondition').

Please note the following:
1. The example above is only for illustrating the answer format and is not related to the question you need to answer.
2. All objects in the image are within your reach, so no movement is necessary.
3. Each step can only perform one action on a single object.
4. If you need to remove an object, use the action 'remove' directly.
5. Write your answer in the format above (the answer consists of multiple steps, and each step contains a 'content' and a 'precondition'), enclosed within <ans></ans> tags.

Now answer the following question based on the input image: {question}. You only need to return an answer wrapped with <ans></ans> tags, and the answer should be in JSON format (as shown in the 'answer' field of the example).

The prompt used for '+ seg&depth' is as follows:

Given a scene image, the depth map of the image, the segmentation map of the image and a task, please provide a concise and accurate list of steps to execute the task in order.

Here is an example: {example}

Each step in the answer consists of two parts: the action to be performed in this step ('content') and the steps that must be completed beforehand ('precondition').

Please note the following:
1. The example above is only for illustrating the answer format and is not related to the question you need to answer.
2. All objects in the image are within your reach, so no movement is necessary.
3. Each step can only perform one action on a single object.
4. If you need to remove an object, use the action 'remove' directly.
5. Write your answer in the format above (the answer consists of multiple steps, and each step contains a 'content' and a 'precondition'), enclosed within <ans></ans> tags.

Now answer the following question based on the input image: {question}. You only need to return an answer wrapped with <ans></ans> tags, and the answer should be in JSON format (as shown in the 'answer' field of the example).

The prompt used for 'CoT' is as follows:

Given a scene image and a task, please provide a concise and accurate list of steps to execute the task in order.

Here is an example: {example}

Each step in the answer consists of two parts: the action to be performed in this step ('content') and the steps that must be completed beforehand ('precondition').

Please note the following:
1. The example above is only for illustrating the answer format and is not related to the question you need to answer.
2. All objects in the image are within your reach, so no movement is necessary.
3. Each step can only perform one action on a single object.
4. If you need to remove an object, use the action 'remove' directly.
5. Write your answer in the format above (the answer consists of multiple steps, and each step contains a 'content' and a 'precondition'), enclosed within <ans></ans> tags.

You should first think step by step to reason about the task, identifying all relevant objects, dependen-

cies, and the logical order of actions. After finishing your reasoning, summarize only the final structured answer in JSON format, wrapped with <ans></ans> tags, without including your intermediate reasoning in the output.
Now answer the following question based on the input image: {question}. You only need to return an answer wrapped with <ans></ans> tags, and the answer should be in JSON format (as shown in the 'answer' field of the example).

For PhysAgent [18], we follow the inference procedure described in the original paper, and all hyperparameters are kept consistent with the original settings.

## C.2. The Details of RSGAR

The prompt used in the first round of scene-graph generation is as follows:

You are given an image (along with its depth map and segmentation map) and a natural language task instruction. You have two tasks, **Target Object Identification** and **Scene Graph Generation**

**1. Target Object Identification**
Based on the task instruction, identify the source objects that are directly involved in accomplishing the task. For example:
a. 'Pour a glass of water' → both the kettle (or teapot) and the glass are source objects.
b. 'Pick up the teapot' → the teapot is a source object.

**2. Scene Graph Generation**
Based on the input image and source objects, build a scene graph centered on the source objects. For each source object, list:
a. Its category/name.
b. Include only target objects that have a direct spatial relationship (direct contact or adjacency) or a direct functional relationship with the source object.

Here is an example: {example}

The example above is only for illustrating the answer format and is not related to the question you need to answer. Ensure that the 'source' in the scene graph is exclusively the source object. 'relation' is not limited to the given example and can be any word.

Now finish the **Target Object Identification** and **Scene Graph Generation** based on the original image (maps for reference only) and question '{question}'. You only need to return an answer wrapped with <ans></ans> tags.

Here, '{example}' is a JSON-formatted sample, as shown below (similarly for the rest):

```
{
    "source_objects": [
        {
            "name": "teapot",
            "attributes": ["silver", "full of
                water", "lid closed"],
            "reason": "Required to pour water"
        },
        {
            "name": "cup",
            "attributes": ["ceramic", "empty", "
                handle on right side"],
            "reason": "Receives water"
        }
    ],
    "scene_graph": [
        {"source": "teapot", "relation": "on", "
            target": "tray"},
        {"source": "cup", "relation": "next to",
            "target": "teapot"},
        {"source": "teapot", "relation": "under",
             "target": "box"}
    ]
}
```

The prompt used for generating the scene graph after the first round is as follows:

Given an image (along with its depth map and segmentation map) and several source objects with their scene graphs in history outputs, you are required to take the target objects ('target') in the scene graph of history outputs as the new source objects, and generate a new scene graph based on these source objects.

Here is the history outputs: {history_outputs}

For each source object in this stage, list:
a. Its category/name.
b. Only the target objects that have a direct spatial relationship (direct contact or adjacency) or a direct functional relationship with the new source object.

Here is an example: {example}

The example above is only for illustrating the an-

swer format and is not related to the question you need to answer. Ensure that the 'source' in the scene graph is exclusively the new source object. 'relation' is not limited to the given example and can be any word. 'target' must be something that has not appeared in the scene graph of history outputs.

Finally, you only need to output the source objects and their scene graphs for this stage, following the format of the example. The answer should be wrapped with <ans></ans> tags.

Here, '{history_outputs}' refers to the scene graphs generated in the previous rounds.

The prompt used when incorporating the scene graph for assisted reasoning is as follows:

Given a scene image (along with its scene graph, where the scene graph depicts spatial relationships among certain objects in the image) and a task, please provide a concise and accurate list of steps to execute the task in order.

Here is the scene graph of the input image: {scene_graph}

Here is an answer example: {example}

Each step in the answer consists of two parts: the action to be performed in this step ('content') and the steps that must be completed beforehand ('precondition').

Please note the following:
1. The example above is only for illustrating the answer format and is not related to the question you need to answer.
2. All objects in the image are within your reach, so no movement is necessary.
3. Each step can only perform one action on a single object.
4. If you need to remove an object, use the action 'remove' directly.
5. Write your answer in the format above (the answer consists of multiple steps, and each step contains a 'content' and a 'precondition'), enclosed within <ans></ans> tags.

Now answer the following question based on the input image and scene graph (the scene graph may not be accurate and is for reference only): {question}.

You only need to return an answer wrapped with <ans></ans> tags, and the answer should be in JSON format (as shown in the 'answer' field of the example).

Here, '{scene_graph}' refers to the previously generated scene graph.

## D. Efficiency Analysis

We measured the time required for RSGAR and the other baselines to perform a single round of reasoning over the entire SpatiaLQA dataset. All experiments were conducted on an NVIDIA GeForce RTX 4090 and the VLM used was GPT-4o. The prompts and hyperparameters for each method are provided in Sec. C. The results are shown below:

| Method | $\tau$ | $F_c$ | $F_p$ |
|---|---|---|---|
| Vanilla Reasoning | 27.4h | 67.4 | 25.1 |
| + depth | 27.9h | 64.1 | 19.0 |
| + seg | 29.4h | 50.3 | 14.7 |
| + seg&depth | 30.1h | 50.5 | 14.5 |
| PhysAgent | 31.5h | 64.7 | 22.2 |
| CoT | 27.5h | 67.6 | 27.0 |
| RSGAR ($T = 1$) | 57.9h | 68.5 | 27.4 |
| RSGAR ($T = 5$) | 174.5h | 69.8 | 28.1 |

Table A1. Efficiency analysis. $\tau$ is the time taken by each method to perform one validation on SpatiaLQA.

The results in Tab. A1 show that although RSGAR requires longer inference time, it achieves better performance than the other baselines. This is because RSGAR is task-oriented and progressively transforms the original image into an interpretable scene graph, providing the VLM with additional information during the reasoning phase, thereby improving overall performance.