Synthesizing Visual Concepts as Vision-Language Programs
Supplementary Material

The following appendix provides supplementary materials referenced in the main text. We begin with an overview of the datasets used in the evaluations (Suppl. A), followed by a discussion of qualitative examples (Suppl. B). Additional supporting experiments are presented in Suppl. C, and failure cases are analyzed in Suppl. E. We discuss limitations of the dataset Bongard-OpenWorld in Suppl. F. Finally, we describe the DSL used for VLP (Suppl. G) and list the prompts employed in the experiments (Suppl. H).

A. Datasets

In the following, we present the datasets used in the evaluations in more detail, along with representative samples.

Bongard-HOI. The Bongard-HOI dataset [13] is one of the real-world image datasets used for evaluating our method, specifically focusing on logical rules related to Human-Object Interactions (HOI). This dataset requires the model to induce rules based on how *people are interacting with objects* in an image. Like all Bongard datasets used, each task provides 12 samples from which the target rule should be induced: 6 positive examples that conform to the latent rule (the rule set) and 6 negative examples that violate the rule (the anti-rule set). We evaluate on all 166 test concepts of the four different test splits. For Tab. 1, we take the first sample of each rule. For RQ3 we reduce the test concepts to 67, only keeping those concepts that have enough few-shot examples (50 or more for positive and negative set respectively). To achieve this, we collect all available images per rule and discard the problems that have fewer than 100 unique few-shot examples. The number of test samples is set to 4. We ensure that test examples are not present in the few-shot examples. This reduces the dataset from 166 to 67 problems. An example concept that is tested in RQ1 and RQ3 both is depicted in Fig. 6.

Bongard-OpenWorld. The Bongard-OpenWorld dataset [44] is based on real-world images and concepts, designed to extend the challenge of rule induction to broader, open-world concepts and complex relational patterns. It shares the fundamental Bongard problem structure, where the goal is to induce a hidden logical rule from a small, balanced set of visual examples. Each task consists of 12 samples in total: 6 positive examples conforming to the latent rule, and 6 negative examples that violate it. An example is depicted in Fig. 7. We evaluate our method on all 200 test samples of the dataset.

Bongard-RWR The Bongard-RWR dataset [19] is also based on real-world images, specifically focusing on abstract visual concepts derived from the original Bongard problems. It is utilized to test the induction of abstract and complex rules within natural, diverse imagery. Following the standard format, each task instance provides 12 samples (6 positives and 6 negatives) from which the underlying logical rule must be determined. We evaluate on all 60 concepts of the dataset and take the first sample per concept (same as Bongard-HOI). Examples of the dataset are provided in Fig. 8 and Fig. 9.

COCOLogic. The COCOLogic dataset [36] is selected for its inclusion of complex logical rules applied to real-world images. Built upon the widely used COCO dataset, it introduces logic-based classification tasks that require reasoning over object co-presence and object counts. We construct ten distinct tasks from COCOLogic, one for each dataset class. Positive samples are drawn from the target class to ensure all relevant objects appear at least once in the few-shot examples, while negative samples are taken from the remaining classes. Each few-shot task includes 20 balanced samples. For testing, we select up to ten query samples per task for both positive and negative classes, although some classes contain fewer available examples. For RQ3, similar to Bongard-HOI, we use a subset of 8 tasks to ensure a sufficient number of few-shot examples. Overall, COCOLogic provides a challenging benchmark for evaluating model performance on complex logical reasoning tasks within diverse, natural image contexts.

CLEVR-Hans3 For the synthetic dataset component, we utilize CLEVR-Hans3 [34]. This dataset is a variant of the CLEVR [14] dataset, specifically constructed to enforce complex logical rules. We leverage the three primary classes present in CLEVR-Hans3 to construct three corresponding tasks, each requiring the induction of complex logical rules related to object color, shape, size, and material. For the support set, we take 20 balanced samples per task, same for the query samples. We utilize the original confounded validation set of CLEVR-Hans3 in our evaluations as a test set, thus turning the originally confounded classification task into a standard one. An example is provided in Fig. 11.

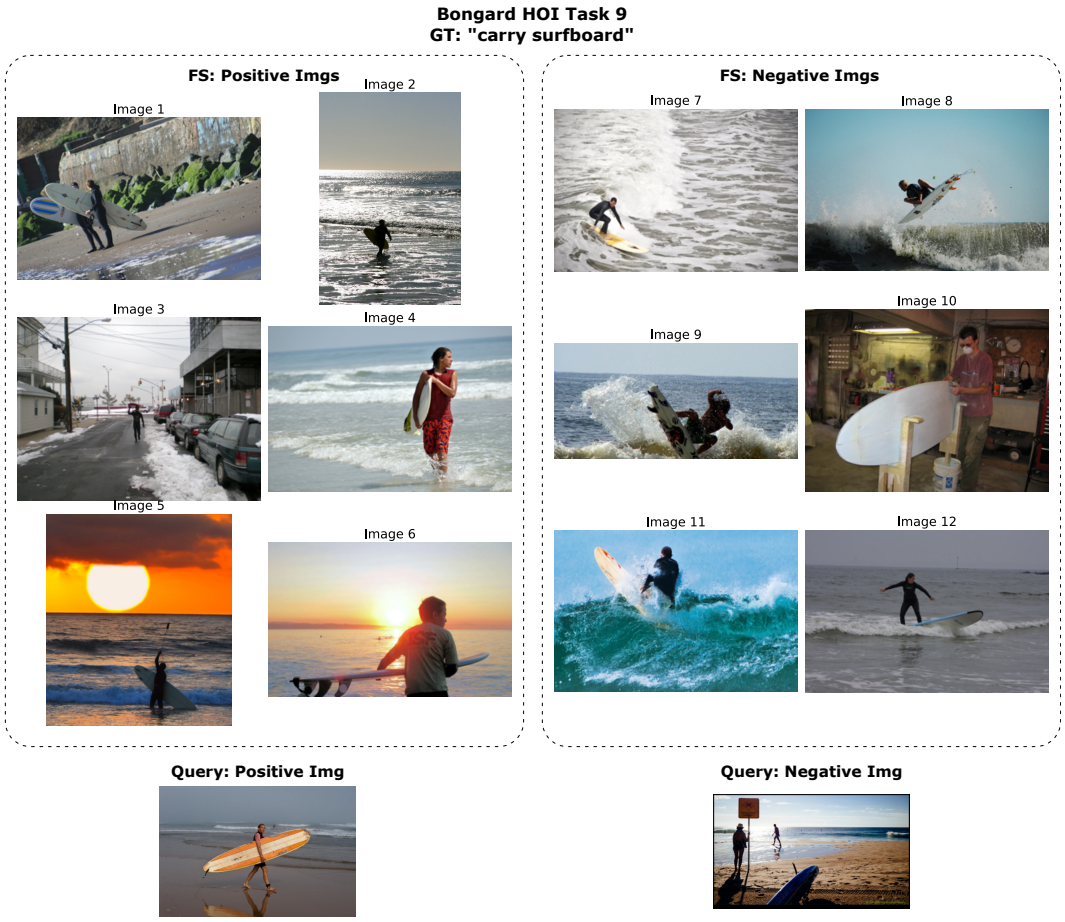

Figure 6. Example task from Bongard-HOI (Task 9, GT: "carry surfboard"). Bongard-HOI focuses on human-object interactions in natural images, requiring models to distinguish specific actions or relationships between people and objects. Positive examples show people carrying surfboards, while negative examples depict other surfboard-related activities such as surfing or standing near surfboards without carrying them.

B. Qualitative Examples

In the following we show an example of results from VLP on a Bongard-HOI task (Sec. B.1).

B.1. Bongard-HOI Example

VLP is not entirely free from shortcut learning. However, we consider this to be a limitation inherent to the nature of the underlying problems, such as the scarcity of hard negative examples and the presence of ambiguous concepts. We illustrate this with a qualitative example of shortcuts identified in VLP below.

For the concept "carry surfboard" in Bongard-HOI, which was analyzed in RQ1 and RQ3, we observed that with only 12 and 20 support samples, our method retrieved the following program in one run with InternVL-14B:

```
(exists_action(get_actions IMG) holding)
```

This program checks if the positive examples all include the action "holding" and correctly classifies all few-shot examples but fails to specify the object being held. InternVL3-8B, discovers a composition of two actions, holding or walking:

```
(or (exists_action (get_actions IMG) holding)
    (exists_action (get_actions IMG) walking))
```

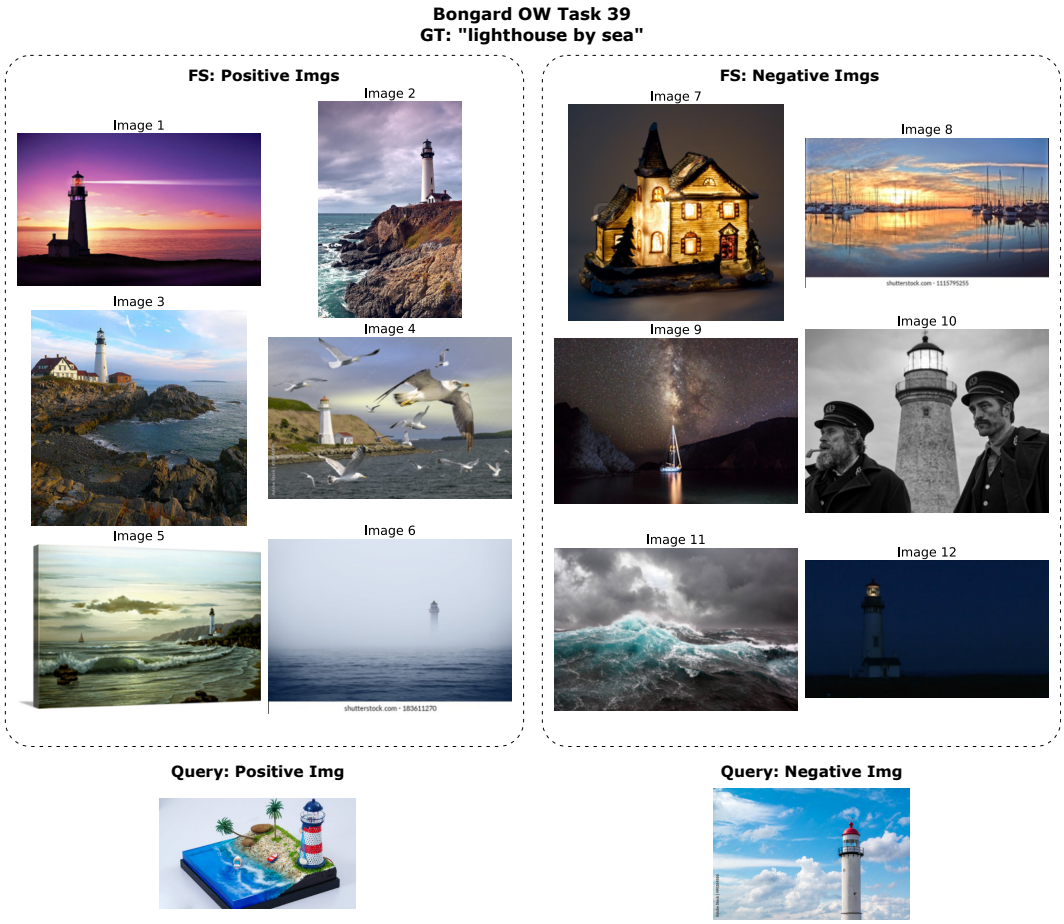

Figure 7. Example task from Bongard-OpenWorld (Task 39, GT: "lighthouse by sea"). Each task consists of 6 positive few-shot images (left) demonstrating the target concept, 6 negative few-shot images (right) showing contrasting examples, and two query images (bottom) to be classified. Models must induce the visual rule from the few-shot examples and apply it to novel queries.

In the RQ3 experiments, we increase the number of few-shot examples, thereby providing stronger evidence for the target concept. Consequently, the retrieved programs become more refined. For example, with 100 few-shot examples, InternVL3-14B retrieves:

```
(or (exists_action_with_object (get_actions var0) holding surfboard)
  (exists_action_with_object (get_actions var0) standing surfboard))
```

checking, if there is either the action *holding surfboard* or *standing* with *surfboard* present. InternVL3-8B retrieves the program:

```
(or (exists_action_with_object (get_actions IMG) walking surfboard)
  (exists_action_with_object (get_actions IMG) holding surfboard)).
```

which checks if there is the action *walking* with *surfboard* or *holding surfboard*.

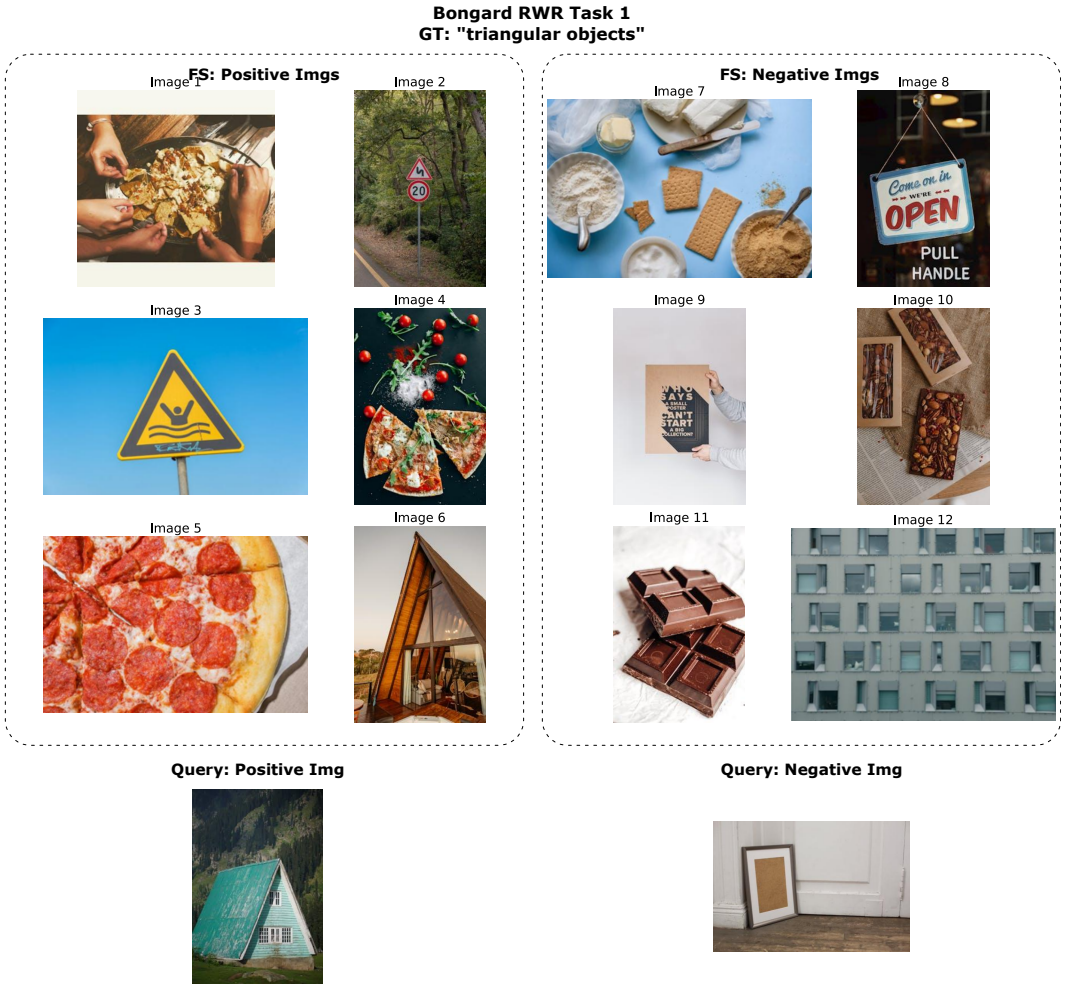

Figure 8. Example task from Bongard-RWR (Task 1, GT: “triangular objects”). Similar to Bongard-OW, each task provides 6 positive and 6 negative few-shot examples, followed by two query images to classify. Bongard-RWR features real-world photographic images with more complex visual scenes and contextual variation compared to simplified geometric tasks.

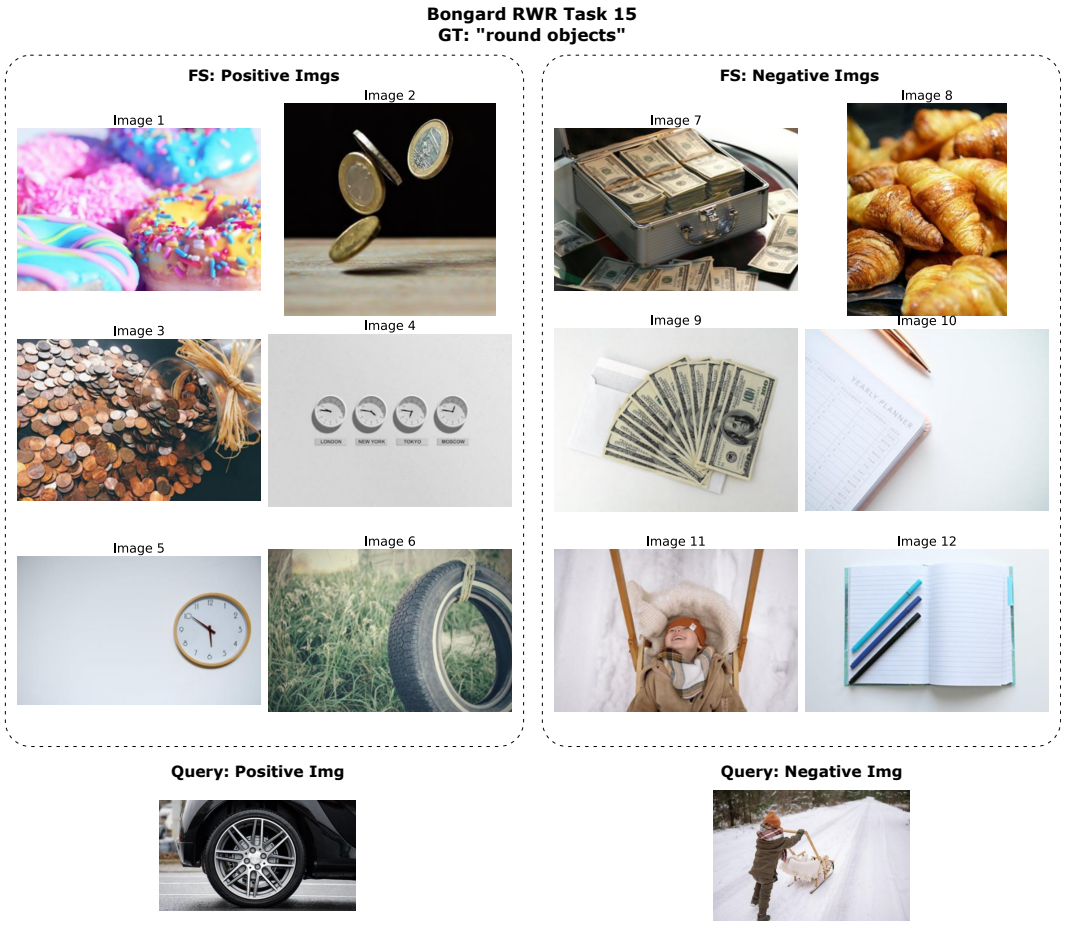

Figure 9. Bongard-RWR Task 15 (GT: "round objects"). Positive examples feature prominent circular or round objects (donuts, coins, clocks, wheels), while negative examples show rectangular or non-round items (money cases, croissants, notebooks). The task requires identifying roundness as the distinguishing visual property across diverse object categories and contexts.

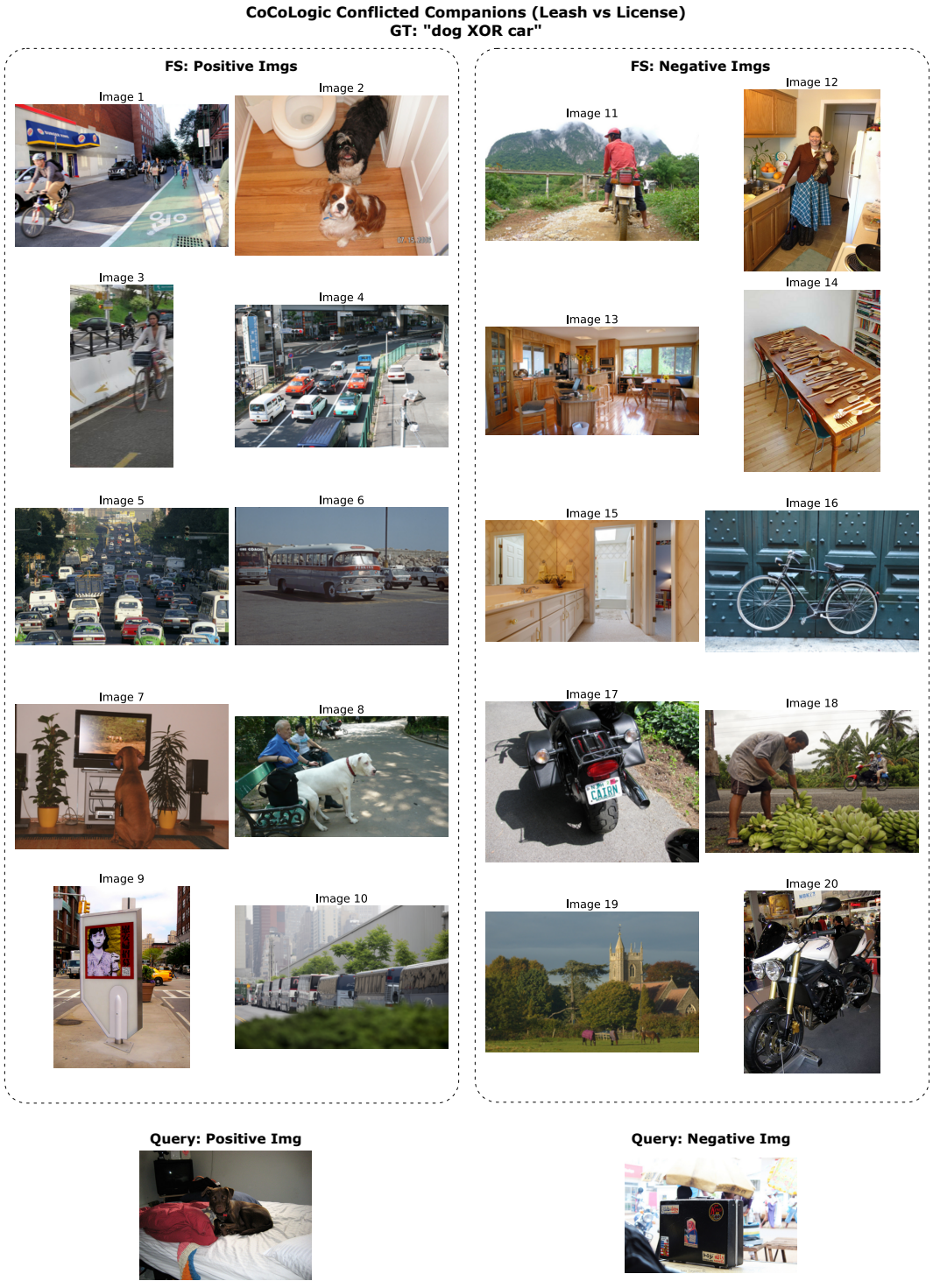

Figure 10. Example task from COCOLogic ("Conflicted Companions", GT: "dog XOR car"). Each task contains 10 positive and 10 negative few-shot examples (shown), plus 2 query images (shown at bottom). The dataset uses natural images from COCO to test logical reasoning with Boolean operators (AND, OR, XOR, NOT). This task requires identifying images containing either dogs or cars, but not both.

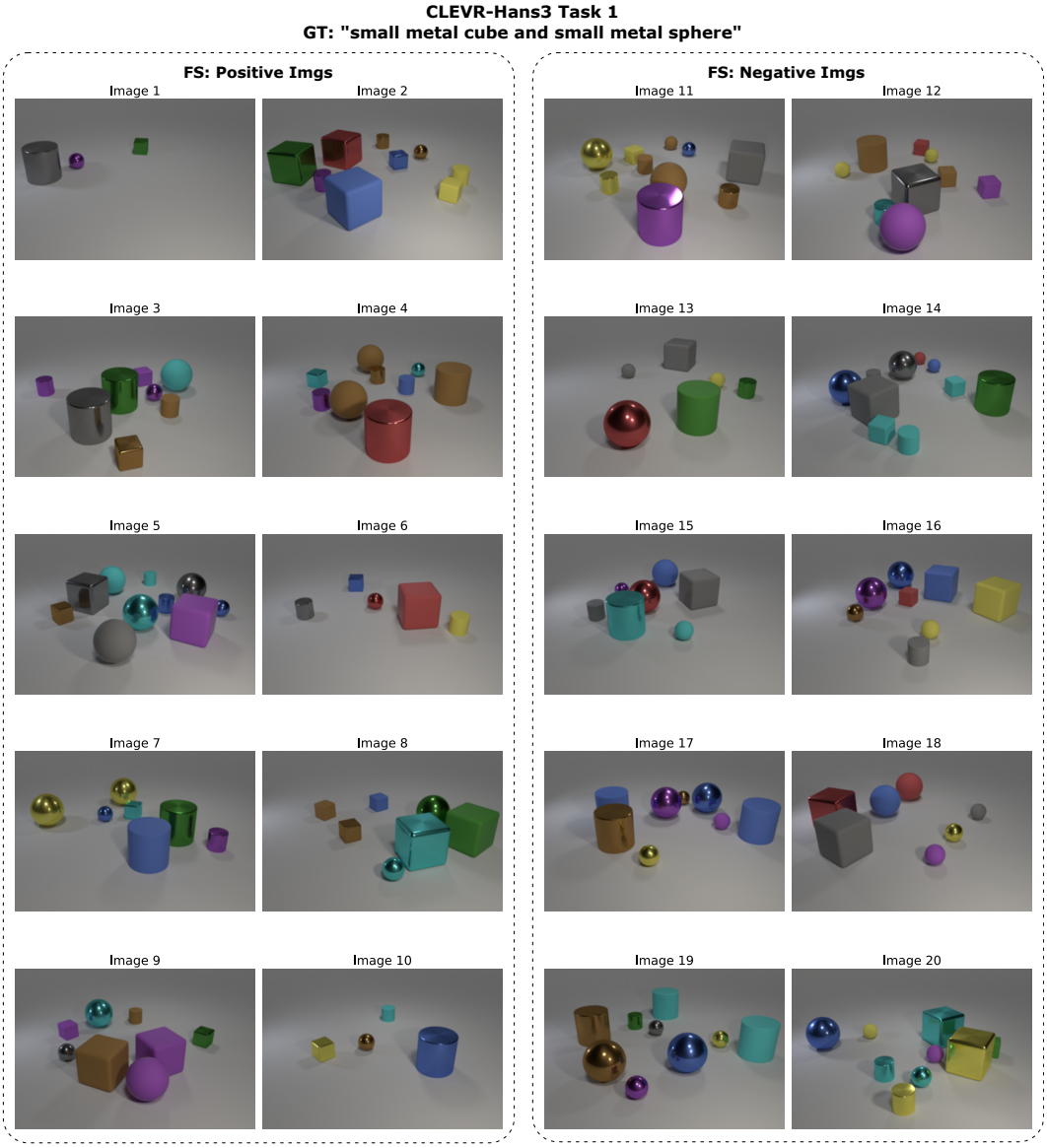

Figure 11. Example task from CLEVR-Hans3 (Task 1, GT: “small metal cube and small metal sphere”). Each task contains 10 positive and 10 negative few-shot examples (shown), plus 10 positive and 10 negative query images (not shown for space). The dataset uses synthetically rendered 3D scenes with controlled object properties (size, material, shape, color), enabling systematic evaluation of compositional reasoning with multi-attribute conjunctive rules. A shortcut that is discovered by Qwen3-VL in these few-shot examples is the rule ”*red object XOR not gold metallic cylinder*” (true for all images except for image 6).

C. Additional Experiments

To complement the results presented in the main paper, this appendix provides a set of additional experiments designed to deepen our understanding of how VLP behaves under different design choices and ablations. Together, these studies allow us to isolate the contribution of individual components, quantify their effect on performance, and verify the robustness of VLP across modeling and decoding settings. The following subsections report these results and discuss their implications.

C.1. Ablation of Program Synthesis

We examine whether structured symbolic representations of images provide an advantage over reasoning directly with the images. To achieve this, we introduce a second baseline in which VLMs receive the structured representations generated by the VLM functions of VLP, rather than the raw images. This setup allows a more direct comparison of rule induction, since both approaches operate on the same underlying information. The results are presented in Tab. 3 as "w/ VLM functions" together with the original results from (RQ1). For Qwen3-VL we report one seed only, since the model got stuck in reasoning traces and was not able to formulate a final response in over 60% of the cases.

Interestingly, this alternative increases performance slightly in some cases, but also performs worse than the standard baseline in others. On average, it is not able to come close to the performance of VLP. Overall, the results indicate that rules induced by VLP are by far more reliable than those obtained by prompting the model directly, even when structured symbolic representations are provided.

Table 3. **Comparison of base VLMs, base VLMs with VLM functions and VLMs with VLP (averaged over three runs).** Balanced accuracy (%) on 6-shot Bongard tasks and 10-shot logical reasoning benchmarks. Best results per model are shown in **bold**; overall best results per dataset column are underlined. *Qwen3-VL with one seed only, generation did not terminate.

Model	Avg.	Bongard Tasks (6-shot)			Logical Reasoning (10-shot)	
		Bongard-HOI	Bongard-OW	Bongard-RWR	COCOLogic	CLEVR-Hans3
InternVL3-8B	57.4	60.5	59.2	47.2	71.5	48.3
w/ VLM functions	56.6	60.5	56.2	50.0	70.2	46.1
w/ **VLP**	**70.9** (+13.5)	**77.7** (+17.2)	**67.5** (+8.3)	**53.9** (+6.7)	**81.0** (+9.5)	**74.4** (+26.1)
InternVL3-14B	61.5	66.9	62.5	51.4	**78.4**	48.3
w/ VLM functions	61.5	70.8	62.5	50.0	74.0	50.0
w/ **VLP**	**68.3** (+6.8)	**74.3** (+7.4)	**64.7** (+2.2)	**52.8** (+1.4)	76.7 (-1.7)	**73.3** (+25.0)
Kimi-VL-A3B-Instruct	58.5	59.8	58.6	46.4	**77.9**	50.0
w/ VLM functions	55.9	57.3	54.2	53.3	62.2	52.2
w/ **VLP**	**65.5** (+7.0)	**69.4** (+9.6)	**59.4** (+0.8)	**52.5** (+6.1)	70.1 (-7.8)	**76.1** (+26.1)
Qwen2.5-VL-7B-Instruct	60.1	65.2	**66.2**	**49.7**	73.2	46.1
w/ VLM functions	56.9	57.3	55.5	53.9	61.9	56.1
w/ **VLP**	**69.5** (+9.4)	**68.8** (+3.6)	62.9 (-3.3)	49.2 (-0.5)	**80.5** (+7.3)	**86.1** (+40.0)
Qwen3-VL-30B-A3B-Instruct	63.4	69.0	**68.5**	55.8	73.9	50.0
w/ VLM functions*	56.0	60.2	57.5	50.8	61.5	50.0
w/ **VLP**	**68.9** (+5.5)	**74.5** (+5.5)	66.3 (-2.2)	**58.3** (+2.5)	**79.1** (+5.2)	**66.1** (+16.1)

C.2. Uniform vs. Occurrence-based weighted distribution

In this section, we investigate whether our proposed occurrence-based symbol weighting improves the performance of the VLP compared to a uniform symbol distribution. Tab. 4 reports the corresponding delta values. Overall, the weighted distribution leads to performance gains in most settings, indicating that emphasizing frequently observed symbols in the positive examples helps the model induce more accurate rules.

C.3. Comparing Deterministic and Sampling Based Decoding

In Tab. 5, we re-evaluate the setups from Tab. 1 using greedy decoding instead of sampling, both for the baseline model and for the VLP prompts. The relative performance pattern across models remains largely unchanged, indicating that the improvements from VLP do not depend on stochastic decoding but arise from the structure of the prompting itself.

Table 4. Difference of using occurence-based weighting instead of uniform for symbols.

	Average	Bongard-HOI	Bongard-OW	Bongard-RWR	COCOLogic	CLEVR-Hans3
InternVL3-8B	+1.1	+0.5	+1.0	0.0	+0.3	+3.8
InternVL3-14B	+0.2	+1.3	+0.4	−0.8	−0.3	+0.5
Kimi-VL-A3B-Instruct	+0.3	+1.2	+0.3	+2.5	+1.8	−4.5
Qwen2.5-VL-7B-Instruct	−0.9	−0.5	+0.7	−3.3	−1.0	−0.6
Qwen3-VL-30B-A3B-Instruct	+0.6	+0.1	0.0	+0.2	+2.1	+0.5

Greedy decoding yields a small decline for the baseline with Intern-VL3 and Qwen2.5-VL models, while Kimi-VL and Qwen3-VL benefit slightly from it. Interestingly, while the results for VLP with InternVL3-8B slightly decrease, for the other models, the results improve. Kimi-VL increases from an average of 65.5 to 67.7 under greedy decoding, and Qwen3-VL even gains an improvement from 68.9 to 71.4. These gains suggest that VLP interacts well with deterministic generation, likely because the structured prompts restrict the model's search space and reduce the chance of drifting into suboptimal continuations.

However, the previous best score of InternVL3-8B under sampling at 70.9 reduces to 67.3 in this setting. This indicates that, for some models, the small amount of exploration introduced by sampling might still be beneficial and allows the decoder to escape overly conservative predictions, *e.g.*, during symbol grounding.

In general, greedy decoding produces more stable outputs when applying VLM functions, while sampling can provide useful diversity during symbol grounding by generating a broader set of candidate symbols. A balanced combination of both approaches may therefore offer the strongest performance, and exploring such hybrid strategies is an interesting direction for future work.

Table 5. **Comparison of baseline and VLP prompting with greedy decoding.** Accuracy (%) across Bongard benchmarks and logical reasoning tasks. Improvements (green / gray) denote changes relative to the baseline. Best results per model are shown in **bold**; overall best results per dataset column are underlined.

Model	**Avg.**	**Bongard Tasks**			**Logical Reasoning**	
		Bongard-HOI	Bongard-OP	Bongard-RWR	COCOLogic	CLEVR-Hans3
InternVL3-8B	56.7	58.4	56.2	**47.5**	74.6	46.7
w/ **VLP**	**67.3** (+10.6)	**79.2** (+20.8)	**67.8** (+11.6)	42.5 (-5.0)	**78.6** (+4.0)	**68.3** (+21.6)
InternVL3-14B	59.6	67.2	63.5	48.3	70.5	48.3
w/ **VLP**	**69.1** (+9.5)	**75.0** (+7.8)	**66.2** (+2.7)	**51.7** (+3.4)	**81.1** (+10.6)	**71.7** (+23.4)
Kimi-VL-A3B-Instruct	59.2	59.3	**58.5**	50.0	**78.4**	50.0
w/ **VLP**	**67.2** (+8.0)	**70.2** (+10.9)	58.0 (-0.5)	**53.3** (+3.3)	72.8 (-5.6)	**81.7** (+31.7)
Qwen2.5-VL-7B-Instruct	60.0	63.3	**65.0**	**51.7**	75.0	45.0
w/ **VLP**	**69.6** (+9.6)	**67.2** (+3.9)	63.5 (-1.5)	48.3 (-3.4)	**80.7** (+5.7)	**88.3** (+43.3)
Qwen3-VL-30B-A3B-Instruct	63.7	72.0	**69.5**	50.0	76.8	50.0
w/ **VLP**	**71.4** (+7.7)	**74.4** (+2.4)	68.5 (-1.0)	**60.0** (+10.0)	**80.8** (+4.0)	**73.3** (+23.3)

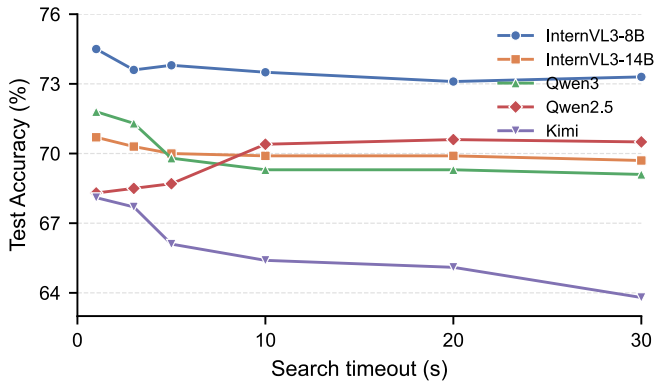

Figure 12. Performance trend across search timeouts.

Model	COCOLogic Seconds (s)	CLEVR-Hans3 Seconds (s)
Qwen3 w/ Think	6534.0	6468.6
Qwen3 w/ **VLP**	**1376.8**	**357.3**
Kimi w/ Think	**3787.3**	3112.1
Kimi w/ **VLP**	4603.6	**752.1**
GPT-5 w/ Think	3702.0	3208.4
GPT-5 w/ **VLP**	**1363.4**	**261.5**

Table 6. Execution time comparison between thinking models and VLP.

C.4. The Influence of the Search Budget

We ablate the influence of search time on the average test performance across datasets under varying search budgets in Fig. 12 for one seed. Interestingly, we observe that VLP achieves strong performance already with very small budgets and tends to overfit with longer search times. This highlights that the search budget is a tunable hyperparameter that can be optimized via distinct validation sets.

C.5. Latency of VLP

The primary computational bottleneck in VLP is not the search time during program synthesis, but rather the frequency of VLM function calls. Formally, the total runtime of VLP can be approximated as:

$$T_{\text{run}} \approx t_{\text{VLM}} \cdot N_{\text{img}} \cdot N_{\text{func}} + B_{\text{search}} \tag{1}$$

where t_{VLM} denotes the average VLM inference latency, N_{img} is the number of task images, N_{func} represents the number of VLM function evaluations per image, and B_{search} is the fixed search budget. Under this formulation, execution costs scale linearly with both the image count and the per-image function calls. Prompting the baselines in Tab. 1 is naturally faster than our method, since they require only a single prompt, whereas VLP uses multiple prompts. Depending on the model, baseline evaluation takes 6 to 21 seconds per task.

Table 6 characterizes the computational overhead of VLP relative to iterative *thinking* models. We find that VLP offers a superior latency profile, outperforming thinking models that often require upwards of 10 minutes per task. This highlights VLP's more efficient reasoning in comparison to thinking models.

D. Towards an automatic DSL

While our paper includes a specific implementation of the DSL for our evaluations, the general framework of VLP is not limited to these exact functions. The overall idea of the work is to integrate VLMs into a DSL to enable flexible functions that can execute on complex and diverse images. In fact, one of our contributions is to make the DSL more automated by having the symbols (objects, properties, actions) dynamically discovered by the VLM (Sec. 3.2) and in doing so, show that our approach provides strong improvements across diverse domains.

However, an exciting direction for further DSL automation and adaption is to use a VLM to propose functions. We explore this on two exemplary spatial reasoning tasks of the famous Bongard problems (#36 and #37)[2][3] with the ground truth rule "triangle above circle". Here, InternVL3-8B, both without and with VLP, struggled to discover the ground truth rule, as spatial relationships where not explicitly represented in our initial DSL. Next, we present the base DSL together with 10 images of BP#36 to gpt-5 and ask for 5 new functions to extend the DSL. It adds VLM-functions like `left_of`, `above` and `below`. With this updated DSL, VLP can induce the underlying rules

```
(below IMG circle triangle)
```

[2] https://www.foundalis.com/res/bps/bongard/p036.htm
[3] https://www.foundalis.com/res/bps/bongard/p037.htm

for BP#36 and

```
(above IMG triangle circle)
```

for BP#37, using the invented VLM-functions `below` and `above`. With the automatically extended DSL VLP can solve both problems. This demonstrates that the DSL in VLP can easily be extended when required for the domain at hand, even automatically. VLP thus offer an exiting opportunity to enable adaptation to diverse domains without manual engineering of standard approaches that utilize DSLs.

E. Failure Cases

In this section we discuss potential failure cases of our method. These failures occur primarily for two reasons: (1) the VLM produces incorrect symbolic representations or misinterprets visual content (Sec. E.1), or (2) the VLM fails to retrieve the relevant image elements during symbol grounding (Sec. E.2).

E.1. VLM Generation Quality

We evaluate the syntactic validity of the symbolic representations produced by the VLMs for `get_objects` and `get_actions` in Tab. 7. Models such as InternVL3-14B reliably generate parsable outputs, whereas others, like Kimi, produce numerous non-parsable representations, particularly on COCOLogic. These failures typically arise when the model enters a repetitive loop, repeating list elements without closing the list. In such cases, we automatically repair the list, though the resulting content is often less informative and may contain hallucinations. This behavior also helps explain why Kimi is the model that performs worst when paired with VLP.

Table 7. **Object** and **Action** Parse Rates Across Models and Datasets

Dataset	Model	Object parse rate	Action parse rate
bongard-hoi	InternVL3-14B	1.00	1.00
	InternVL3-8B	1.00	1.00
	Kimi-VL-A3B-Instruct	0.96	0.95
	Qwen2.5-VL-7B-Instruct	0.98	0.99
	Qwen3-VL-30B-A3B-Instruct	1.00	1.00
bongard-op	InternVL3-14B	1.00	1.00
	InternVL3-8B	1.00	1.00
	Kimi-VL-A3B-Instruct	0.94	0.93
	Qwen2.5-VL-7B-Instruct	0.98	0.98
	Qwen3-VL-30B-A3B-Instruct	0.99	1.00
bongard-rwr	InternVL3-14B	1.00	1.00
	InternVL3-8B	1.00	1.00
	Kimi-VL-A3B-Instruct	0.97	0.73
	Qwen2.5-VL-7B-Instruct	0.98	1.00
	Qwen3-VL-30B-A3B-Instruct	0.99	1.00
cocologic	InternVL3-14B	1.00	1.00
	InternVL3-8B	1.00	1.00
	Kimi-VL-A3B-Instruct	0.87	0.87
	Qwen2.5-VL-7B-Instruct	0.94	0.97
	Qwen3-VL-30B-A3B-Instruct	0.97	1.00
CLEVR-Hans3	InternVL3-14B	1.00	-
	InternVL3-8B	0.99	-
	Kimi-VL-A3B-Instruct	0.99	-
	Qwen2.5-VL-7B-Instruct	1.00	-
	Qwen3-VL-30B-A3B-Instruct	1.00	-

E.2. Symbol Discovery Quality

VLP can only reason about symbols that have been retrieved during the *symbol grounding* stage. Therefore, one obvious failure case of VLP is that one or more relevant symbols for the visual concept are not retrieved in this initial step. To analyze how often this occurs, we conduct an experiment, checking the quality of the grounded symbols for each model and dataset. Hereby, we proceed as follows. If a dataset has objects, properties, or actions explicitly specified in the ground truth rule, such as Bongard-HOI, COCOLogic, and CLEVR-Hans, we leverage them to check the grounded symbols against the elements present in the rule. For the datasets Bongard-OpenWorld and Bongard-RWR, the visual concepts are more vague and not always bound to objects, *e.g.* "*living room*" or "*aerial view*". Here, we ask an LLM to extract objects, properties, and actions from the rule to serve as a comparison to the grounded symbols of VLP, as an approximation.

Since there might be multiple terms with the same or close semantic meaning, we do not perform exact equality checking on the retrieved symbols but rather ask an LLM to judge whether the ground truth symbol is present in the symbols retrieved by VLP. For both LLM usages described, we take the model "gpt-4o-2024-08-06".

The results of this analysis are displayed in Fig. 13 for objects, properties, and actions, respectively. Datasets for a symbol type are only considered if the symbol type is part of the ground truth concept.

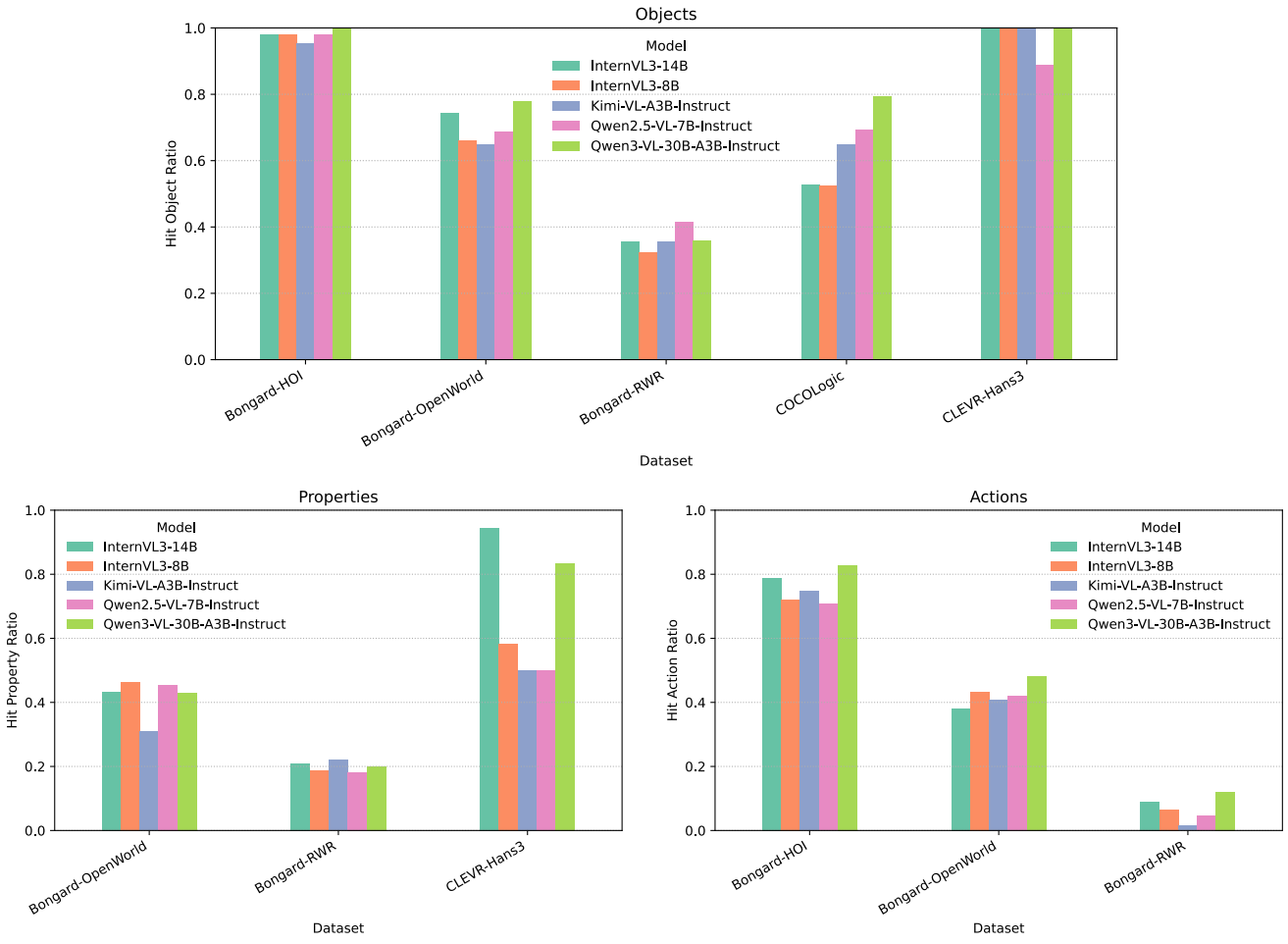

Figure 13. Hit ratios of the discovered object, property, and action symbols across all datasets. These values measure how often the symbols extracted by the VLM match the ground truth concepts. Datasets on which VLP performs well in rule induction also show higher symbol quality in this analysis.

We observe that datasets on which VLP achieves the strongest results also tend to exhibit higher quality ratios for the discovered symbols. This relationship is expected, as strong downstream performance implies that the model can reliably

detect the visual aspects relevant to the ground-truth concept. Conversely, low symbol quality is likely a contributing factor to weak performance in the rule induction task. For example, Bongard-RWR shows the lowest quality ratios across all components, which aligns with its status as the most challenging dataset in our main evaluation.

While the ground truth symbols in this analysis are approximations generated by an LLM from the official Bongard solutions[4], the overall trend remains plausible. The original Bongard concepts are well known to be difficult for modern VLMs, even when translated into real-world imagery [19, 46].

F. Dataset Quality Issues in Bongard-OpenWorld

During our experiments, we identified systematic annotation errors in the Bongard-OpenWorld dataset that fundamentally compromise both learning and evaluation. These issues fall into three categories: (1) mislabeled few-shot examples that contradict the ground-truth rule, (2) query images that violate the stated concept, and (3) inconsistent application of conjunctive rules. We document representative examples below to illustrate how these errors prevent reliable performance assessment.

F.1. Inconsistent Few-Shot Labeling

Partial Rule Satisfaction in Conjunctive Concepts. Task 23 (GT: "wooden floor living room") exemplifies how annotators sometimes apply only partial matching to multi-component rules. As shown in Figure 14, Images 4 and 5 in the positive set depict a dining area and a staircase respectively—neither of which are living rooms, despite having wooden floors. This creates fundamental ambiguity: should models learn that *both* conditions must be satisfied (wooden floor AND living room), or that *either* condition suffices? Such errors make it impossible to determine whether poor model performance reflects genuine conceptual limitations or simply confusion induced by contradictory training signals.

Contradictory Labels Within the Same Task. Task 47 (GT: "tomato dishes") contains multiple logical inconsistencies that render the rule unlearnable (Figure 15). Image 12, labeled as a negative example, clearly shows a tomato-based dish. Meanwhile, Image 4 in the positive set shows a pizza, yet the negative query image also depicts a pizza with visible tomatoes. These contradictions make it impossible for any model—or human—to extract a coherent rule from the provided examples.

F.2. Query Image Mislabeling

Attribute Violations in Query Sets. Task 76 (GT: "gift box pink ribbon") demonstrates systematic query set errors (Figure 16). While the few-shot examples correctly distinguish pink ribbons (positive) from other colors (negative), both query images contain white ribbons. Since neither query satisfies the ground-truth rule, their assigned labels are arbitrary, making evaluation metrics meaningless for this task.

Categorical Mismatches. Task 92 (GT: "statue buddha intricate carvings") reveals more severe errors where query images belong to entirely different conceptual categories (Figure 17). The positive query depicts Hindu deity statues rather than Buddha statues—a fundamental categorical error, not merely a boundary case. Such mismatches indicate inadequate quality control in dataset construction and invalidate any assessment of model generalization.

F.3. Implications for Performance Evaluation

These examples are not isolated incidents but represent systematic issues that affect multiple tasks in the dataset. When ground-truth labels are internally contradictory or violate the stated rules, performance drops cannot be meaningfully interpreted. A model that fails on Task 47, for instance, may actually be learning the correct concept but struggling with the dataset's logical inconsistencies. Conversely, high accuracy on Task 76 may reflect memorization of spurious correlations rather than genuine rule understanding. These quality issues must be considered when interpreting any quantitative results on Bongard-OpenWorld.

[4] https://www.foundalis.com/res/bps/bongard_problems_solutions.htm

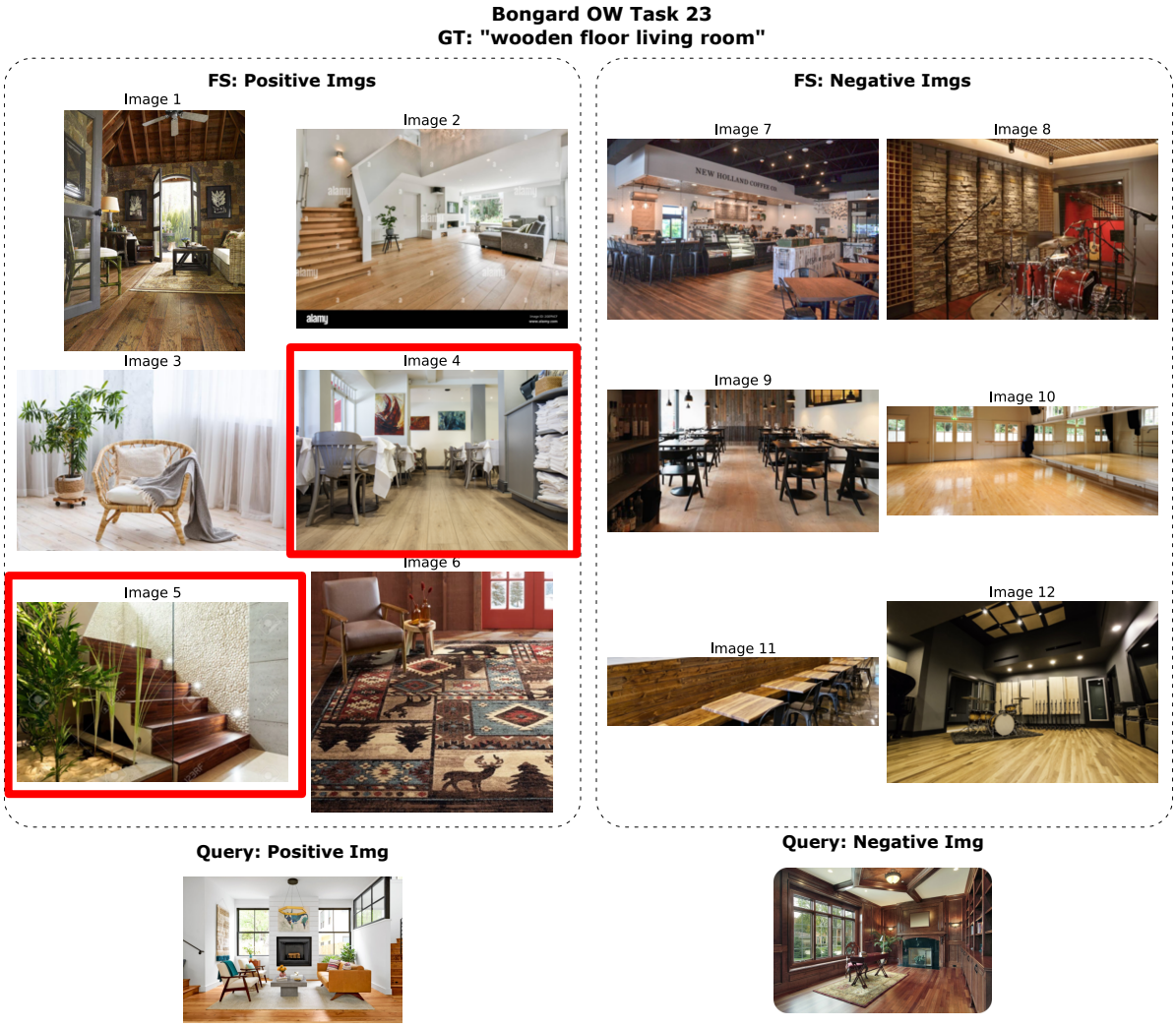

Figure 14. Annotation errors in Bongard-OW Task 23 reveal inconsistent rule application (GT: "wooden floor living room"). Positive few-shot examples include a dining space (Image 4) and a staircase (Image 5, red boxes), which contain wooden floors but are not living rooms.

Figure 15. Annotation inconsistencies in Bongard-OW Task 47 (GT: “tomato dishes”). The dataset contains contradictory labels: Image 12 (negative few-shot) clearly shows a tomato-based dish, while the negative query image shows a pizza with tomatoes despite Image 4 (positive few-shot) also being a pizza. These inconsistencies (highlighted in red) make the underlying rule ambiguous and evaluation unreliable.

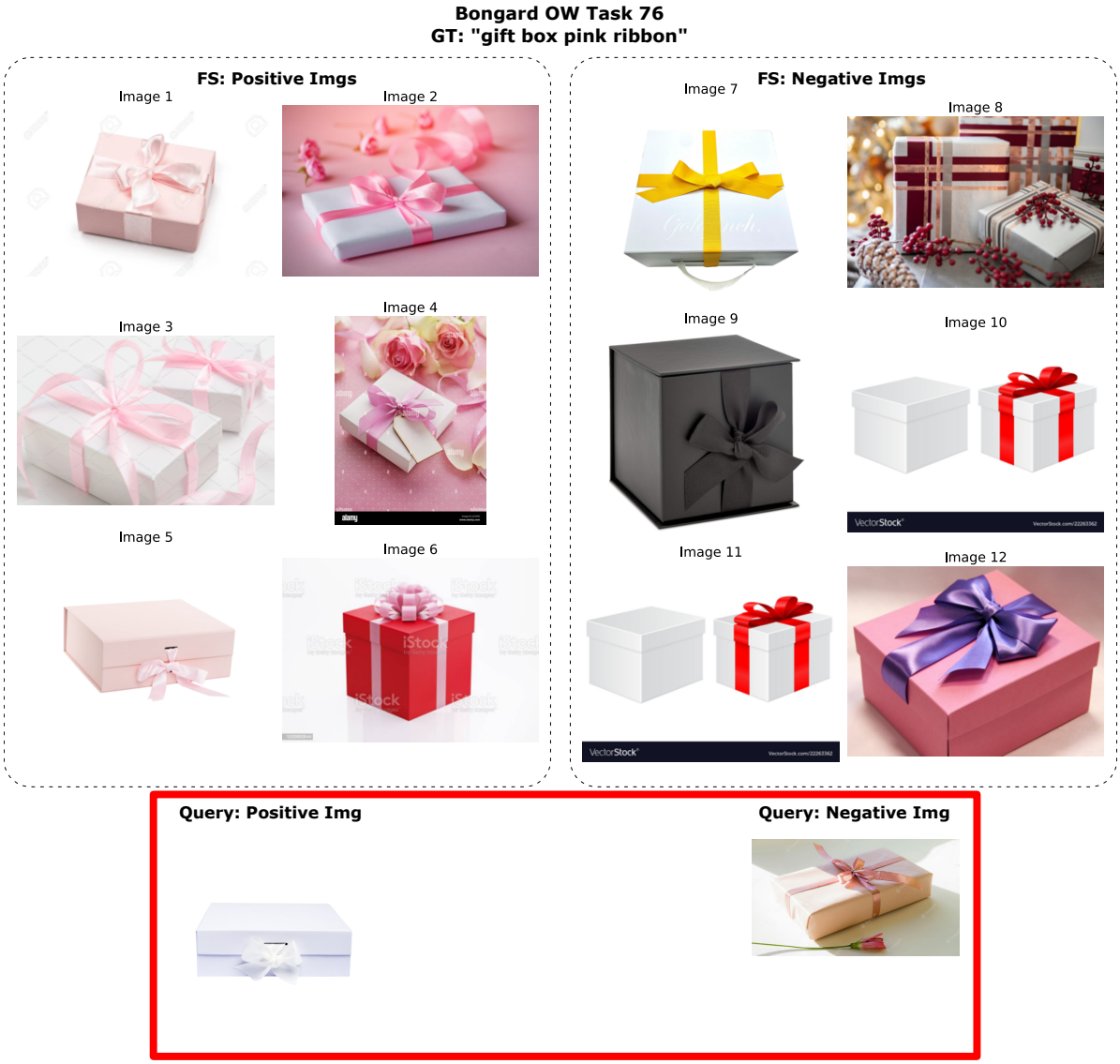

Figure 16. Query image mislabeling in Bongard-OW Task 76 (GT: "gift box pink ribbon"). While the few-shot examples correctly distinguish pink ribbons (positive) from non-pink ribbons (negative), both query images (highlighted in red) contain white ribbons. This violates the ground-truth rule and renders the queries unanswerable based on the provided examples.

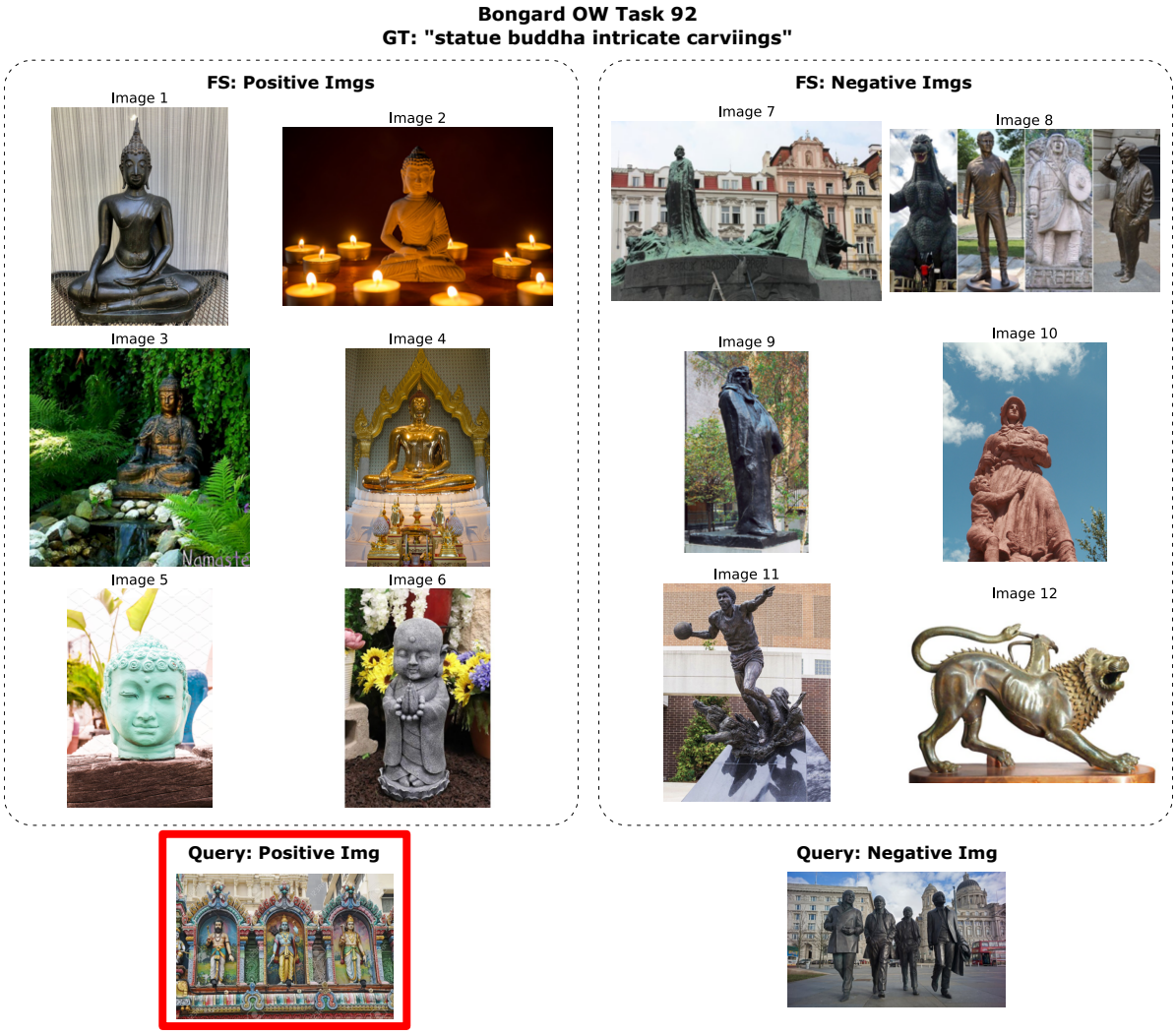

Figure 17. Annotation error in Bongard-OW Task 92 query set (GT: "statue buddha intricate carvings"). While all positive few-shot images correctly show Buddha statues with intricate carvings, the positive query image (red box) contains Hindu deity statues instead. This categorical mismatch undermines the task's validity and prevents meaningful evaluation of rule generalization.

G. DSL used for VLP

In Tab. 8, we present the DSL used in the experiments. While the functions are defined in a general form, they are adapted depending on the focus and characteristics of each dataset. For example, CLEVR-Hans contains synthetic objects that are not associated with actions, and therefore action related functions are unnecessary. Similarly, datasets with lower logical complexity do not require counting or numerical comparison operations, whereas Bongard-RWR and COCOLogic benefit from these additional functions due to their more demanding reasoning requirements.

Table 8. Overview of all primitive functions of the DSL available for each dataset, together with their corresponding type signatures. The table groups functions by dataset and specifies the input and output types using arrow based type notation.

Dataset	Primitive	Type
Bongard-HOI	`get_objects`	IMG → List(List(STRING))
	`get_actions`	IMG → List(List(STRING))
	`exists_object`	List(List(STRING)) → OBJECT → BOOL
	`exists_object_with_property`	List(List(STRING)) → OBJECT → PROPERTY → BOOL
	`exists_property`	List(List(STRING)) → PROPERTY → BOOL
	`exists_action`	List(List(STRING)) → ACTION → BOOL
	`exists_action_with_object`	List(List(STRING)) → ACTION → OBJECT → BOOL
	`and, or`	BOOL → BOOL → BOOL
	`not`	BOOL → BOOL
Bongard-OW	Same as Bongard-HOI +	
	`exists_properties`	List(List(STRING)) → PROPERTY → PROPERTY → BOOL
	`exists_object_with_properties`	List(List(STRING)) → OBJECT → PROPERTY → PROPERTY → BOOL
Bongard-RWR	Same as Bongard-OW +	
	`count_object_in_img`	List(List(STRING)) → OBJECT → INT
	`count_objects_with_property`	List(List(STRING)) → PROPERTY → INT
	`max_objects_of_same_type`	List(List(STRING)) → INT
	`count_all_objects`	List(List(STRING)) → INT
	`xor`	BOOL → BOOL → BOOL
	`gt?, eq?`	INT → INT → BOOL
	`0, 1, 2, 3, 4, 5, 6`	INT
COCOLogic	Same as Bongard-HOI +	
	`count_object_in_img`	List(List(STRING)) → OBJECT → INT
	`count_objects_with_property`	List(List(STRING)) → PROPERTY → INT
	`max_objects_of_same_type`	List(List(STRING)) → INT
	`count_all_objects`	List(List(STRING)) → INT
	`xor`	BOOL → BOOL → BOOL
	`gt?, eq?`	INT → INT → BOOL
	`0, 1, 2, 3, 4, 5, 6`	INT
CLEVR-Hans3	Similar to Bongard-OW, but without action predicates.	

H. Prompts

In the following we provide the used prompts for the baseline Sec. H.1. For VLP we provide the prompts for symbol grounding Sec. H.2 as well as the VLM functions of VLP Sec. H.3. Further we describe the VLM functions added in the course of RQ4 in Sec. H.4. And finally the prompts for the ablation on program synthesis are provided in Sec. H.5.

H.1. Baseline Prompts

Prompts for the baseline used in the experiments. First, the VLM is prompted for a rule that separates the images of the problem, and then it is prompted to evaluate the rule on the test images.

Baseline prompt for obtaining visual rule

You are given {n} images. Each image depicts a scene with specific objects, interactions, and environments.
Your task is to determine the underlying concept that distinguishes the positive examples from the negative examples, based on the objects, their properties, and the actions occurring in each scene.

- The first {m} images are positive examples.
- The remaining {o} images are negative examples.

Task

Identify the rule that defines the positive examples:
- The rule must apply to all positive examples.
- The rule must not apply to any negative example.

Step-by-Step Process

1. Image Analysis:
- Carefully describe each image, noting objects, their attributes, and conceptual features (such as relationships, actions, or settings).

2. Rule Derivation:
- From your analysis, infer the rule that uniquely characterizes the positive examples.
- Confirm that the rule does not hold for the negative examples.

Final Answer Format

Provide your final answer in the following format:

```python
answer = {
'rule': '[RULE]',
}
```

Ensure that the rule is clearly defined and concise.

Baseline prompt for evaluating direct result

Given the rule '{response}', determine if the image follows the rule or not. Answer with 'Yes' or 'No', nothing else.

H.2. Symbol Grounding

In the following the prompts used during symbol grounding are provided. The VLM receives the task images together with the prompt instructions to identify relevant *objects*, *properties* and *actions*.

For the amount of symbols requested, we use the parameters listed in Tab. 9 across all experiments.

Table 9. **Variables used per dataset in program search.** Number of objects, properties, and actions requested when generating programs for each dataset.

Dataset	#Objects	#Properties	#Actions
Bongard-HOI	10	5	10
Bongard-OW	10	10	3
Bongard-RWR	10	10	5
COCOLogic	10	10	3
CLEVR-Hans3	10	10	0

Objects

You are analyzing images to identify notable objects. Focus on clearly identifiable, semantically meaningful objects that appear across the image set, especially those present in some images but not others. Objects include persons, animals, and things. Avoid minor details like shadows or textures. Use specific, descriptive names (e.g., ”bicycle” not ”vehicle”). Return exactly {n} objects in a Python list.

Answer format:
“‘python
objects = [...]
“‘
No comments or explanations. If no objects found, return [].

Properties

Property Discovery Task

You are analyzing a set of images to identify important properties that describe objects in the image set.
Objective

Discover **notable properties** that characterize objects in the images. Focus on properties that meaningfully distinguish or describe objects (visual attributes, spatial relationships, geometric features, states).

Instructions

1. **Examine all images carefully** - Look for properties that apply to the relevant objects across the image set
2. **Identify important properties** - Focus on significant, clearly observable properties that meaningfully describe objects
3. **Consider property variation** - Properties that vary across images or objects may be particularly noteworthy
4. **Prioritize meaningful properties** - Choose properties that help distinguish or characterize objects (e.g., color, size, position, orientation, state)
5. **Return exactly n properties** - If fewer notable properties exist, return as many as available
6. **Use descriptive names** - Name properties clearly and specifically (e.g., ”red” rather than ”colored”, ”horizontal” rather than ”oriented”)

Relevant Objects
The objects to consider are: {objects}

Property Categories
- **Visual attributes**: color, texture, pattern, brightness
- **Spatial properties**: position (left/right, top/bottom, center), proximity, distance
- **Geometric attributes**: size, shape, orientation, symmetry
- **States**: filled/outlined, open/closed, active/inactive

Output Requirements
- Return a Python list assigned to variable ‘properties‘
- Include only the Python code, no explanations or comments
- If no notable properties are found, return an empty list ‘[]‘
- Use clear, specific property names (e.g., ”blue”, ”large”, ”leftmost”, ”vertical”)
- Be general (e.g., if there’s a yellow triangle, use ”yellow” not ”yellow triangle”)

Example Format
“‘python
properties = [”red”, ”large”, ”horizontal”, ”outlined”, ”centered”]
“‘

Actions

Action Discovery Task

You are analyzing a set of images to identify important actions performed by objects in the image set.

Objective

Discover **notable actions** that characterize objects in the images. Focus on actions that meaningfully distinguish or describe what objects are doing (movements, behaviors, states of activity).

Instructions

1. **Examine all images carefully** - Look for actions that apply to the relevant objects across the image set
2. **Identify important actions** - Focus on significant, clearly observable actions that meaningfully describe what objects are doing
3. **Consider action variation** - Actions that vary across images or objects may be particularly noteworthy (contrasting actions)
4. **Prioritize meaningful actions** - Choose actions that help distinguish or characterize object behaviors (e.g., running, jumping, standing, flying)
5. **Return exactly n actions** - If fewer notable actions exist, return as many as available
6. **Use descriptive names** - Name actions clearly and specifically (e.g., ”running” rather than ”moving”, ”sitting” rather than ”positioned”)

Relevant Objects

The objects to consider are: {objects}

Action Categories

- **Movement actions**: walking, running, jumping, flying, rolling
- **Positional actions**: standing, sitting, lying, hanging
- **Interactive actions**: holding, pushing, pulling, touching
- **State actions**: opening, closing, rotating, tilting

Output Requirements

- Return a Python list assigned to variable ‘actions‘
- Include only the Python code, no explanations or comments
- If no notable actions are found, return an empty list ‘[]‘
- Use clear, specific action names (e.g., ”jumping”, ”sitting”, ”rotating”, ”falling”)

Example Format

“‘python
actions = [”running”, ”jumping”, ”standing”, ”flying”, ”sitting”]
“‘

H.3. VLM Functions of the DSL

In the following the two VLM functions used for VLP are presented. These are executed on individual images to obtain a structured representation of the objects and actions in the image.

get_objects - VLM function for obtaining object-property representation

Task
Identify objects and their properties from the image using only the provided lists.

Objects: {objects}
Properties: {properties}

Rules
1. Only use objects/properties from the provided lists
2. Return empty list if no valid objects found
3. No explanations or additional text

Output Format
```python
objects = [
    ['object_name', 'property1', 'property2', ...],
    ['object_name', 'property1'],
...
]
```

If no valid objects: `objects = [[]]`

Examples

Example 1
- Objects: ["car", "person", "tree"]
- Properties: ["red", "tall", "small", "standing"]
- Image: Red car under tall tree with small standing person

```python
objects = [
    ['car', 'red'],
    ['tree', 'tall'],
    ['person', 'standing', 'small']
]
```

Example 2
- Objects: ["dog", "ball", "book", "chair"]
- Properties: ["blue", "sitting", "round"]
- Image: Dog sitting by round ball and blue chair

```python
objects = [
    ['dog', 'sitting'],
    ['ball', 'round'],
    ['chair', 'blue']
]
```

Example 3
- Objects: ["bicycle", "lamp", "table", "cup"]
- Properties: ["green", "broken", "wooden", "white"]
- Image: Table with laptop and cup

```python
objects = [[]]
```
Note: Even though 'table' and 'cup' are in the objects list and visible in the image, neither has properties from the provided list, so no valid object-property combinations exist

Analyze the image now:

get_actions - VLM function for obtaining action-object representation

```
## Task
Identify actions occurring in the image using only the provided lists.

**Actions:** actions
**Objects:** objects

## Rules
1. Only use actions/objects from the provided lists
2. Only detect actions that are actually happening in the image
3. Do not include actions from the list if they are not occurring in the image
4. If an action involves an object, include the object name
5. Return empty list if no valid actions found
6. No explanations or additional text

## Output Format
“‘python
actions = [
    [’action_name1’],
    [’action_name2’, ’object_name2’],
    [’action_name2’, ’object_name1’, ’object_name2’],
    ...
]
“‘

**If no valid actions:** ‘actions = [[]]‘

## Examples

**Example 1**
- Actions: [”running”, ”jumping”, ”sitting”, ”dancing”]
- Objects: [”chair”, ”ball”, ”person”, ”table”]
- Image: Person sitting on chair

“‘python
actions = [
    [’sitting’, ’person’, ’chair’]
]
“‘
*Note: ’running’, ’jumping’, and ’dancing’ are in the actions list but not happening in the image, so they’re excluded*

**Example 2**
- Actions: [”throwing”, ”catching”, ”walking”, ”reading”, ”sleeping”]
- Objects: [”ball”, ”book”, ”dog”, ”frisbee”]
- Image: Person throwing a ball while dog is walking

“‘python
actions = [
    [’throwing’, ’person’, ’ball’],
    [’walking’, ’dog’]
]
“‘
*Note: ’catching’, ’reading’, and ’sleeping’ are in the actions list but not occurring in the image, so they’re excluded*

**Example 3**
- Actions: [”swimming”, ”flying”, ”cooking”]
- Objects: [”pool”, ”bird”, ”kitchen”]
- Image: Person eating at a restaurant

“‘python
actions = [[]]
“‘
*Note: Even though actions are happening in the image, none match the provided actions list, so no valid actions exist*

**Analyze the image now:**
```

H.4. VLM Functions for property size

In the course of (RQ4) we add VLM functions to the DSL that explicitly ask the VLM for objects of small and large size. These are listed in the following.

exists_object_small_in_img

Does the image contain any '{obj}' that is relatively small in size compared to the other objects? Answer with 'YES' or 'NO'.

exists_object_large_in_img

Does the image contain any '{obj}' that is relatively large in size compared to the other objects? Answer with 'YES' or 'NO'.

exists_object_with_property_small_in_img

Does the image contain any '{obj}' with the property '{prop}' that is relatively small in size compared to the other objects? Answer with 'YES' or 'NO'.

exists_object_with_property_large_in_img

Does the image contain any '{obj}' with the property '{prop}' that is relatively large in size compared to the other objects? Answer with 'YES' or 'NO'.

H.5. Structure-based baseline

For the ablation study on program synthesis in Sec. C.1, we implement a structure-based approach that uses the VLM functions defined in the DSL (see above) to extract symbolic representations of the input images. These representations are then passed back to the VLM, which reasons over them to induce a rule. Below we provide the prompt used for this procedure, along with the prompt applied during evaluation when test images are assessed based on their structure-based representations.

Structure-based rule induction

Task: Concept Identification from Structured Image Representations

You will be given a sequence of structured image representations, where each image is labeled as either a **positive** or **negative** example of an underlying concept.

Structure of Each Image Representation

Objects
A list of objects, where each object is described together with its properties.

Format:
```
[[object_name, property1, property2, ...], [object_name, property1, ...], ...]
```
Example:
```
[[dog, brown, large], [cat, small, cute], [ball, red, round]]
```
Actions
A list of actions occurring in the image, including the entities involved.

Format:
```
[[action, participant1, participant2, ...], [action, participant], ...]
```
Example:
```
[[playing, dog, cat], [shining, sun], [rolling, ball]]
```
Your Objective
Identify the rule that defines the positive examples:
- The rule must apply to all positive examples.
- The rule must not apply to any negative example.

Step-by-Step Process
1. Image Analysis:
- Analyze the objects, their properties, and the actions across all examples to identify the concept that distinguishes positive examples from negative ones.

2. Rule Derivation:
- From your analysis, infer the rule that uniquely characterizes the positive examples.
- Confirm that the rule does not hold for the negative examples.

Structured Image Representations
Image: 1
Objects: {obj_repr}
Actions: {action_repr}
Label: 'Positive'

...

Image: {n}
Objects: {obj_repr}
Actions: {action_repr}
Label: 'Negative'

Final Answer Format
Provide your final answer in the following format:
```python
answer = {
'rule': '[RULE]',
}
```

Ensure that the rule is clearly defined and concise.

Structure-based evaluation

Task: Classify Image based on Structured Image Representations

You are given a structured image representation.

Structure of Each Image Representation

Objects
A list of objects, where each object is described together with its properties.

Format:
‘‘‘
[[object_name, property1, property2, ...], [object_name, property1, ...], ...]
‘‘‘
Example:
‘‘‘
[[dog, brown, large], [cat, small, cute], [ball, red, round]]
‘‘‘
Actions
A list of actions occurring in the image, including the entities involved.

Format:
‘‘‘
[[action, participant1, participant2, ...], [action, participant], ...]
‘‘‘
Example:
‘‘‘
[[playing, dog, cat], [shining, sun], [rolling, ball]]
‘‘‘
Your Objective
Given the rule ’{rule}’, determine if the image follows the rule or not. Answer with ’Yes’ or ’No’, nothing else.

Structured Image Representations
{representation}