ViUniT: Visual Unit Tests for More Robust Visual Programming

Supplementary Material

A. Data

The three compositional reasoning datasets used in this work are GQA [3], SugarCREPE [2], and WinoGround [11]. Table 1 shows examples from each dataset, and Table 2 summarizes the dataset statistics. For GQA validation we sample 5 questions from each of the 102 question groups from the `balanced-val` split with a total of 502 examples. For testing, we sample 10 questions per group from the `balanced-train` split yielding 1022 examples. Note that some groups such as `typeVerifyC`, `stateChoose`, and `companyVerify` do not have a sufficient amount of questions, so we sample the whole group. For SugarCREPE, we utilize 788 examples for training by subsampling 10% of the dataset balanced across the 7 question types, excluding our validation split. This validation subset consists of 560 examples and includes both positive and negative image-text pairings from 40 samples for each of the 7 question types. The full Winoground dataset is used, encompassing all possible positive and negative pairings for a total of 1600 examples, with SugarCREPE employed for training.

B. Unit Test Sampling Pseudocode

For clarity, Algorithm 1 presents the pseudocode for the unit test coverage sampling method described in Section 3.

C. Program Generation and Execution

In this section, we outline the implementation details for program generation and execution.

C.1. Generation Details

For program generation we use in context examples both in of-the-shelf inference, and finetuned model inference. Generation is conducted using VLLM with the following generation parameters: `temperature=1.0`, `top_p=0.9`, `top_k=0.0`, `max_new_tokens=320`, and `num_beams=1`. We set the temperature at a high value to ensure diversity in generated programs. For CodeLLaMA we prefix the prompt with `<s>`, and for CodeGemma we enclose it in `<bos><start_of_turn>[..]<end_of_turn>`

C.2. Image Patch API

We present the `ImagePatch` API in Code 1 which we adapt the from Khan et al. [4] which is in turn adapted from ViperGPT Surís et al. [10]. We implement object detection using IDEA-Research/grounding-dino-base [6] with `text_threshold=box_threshold=0.2`, image-text-matching using openai/clip-vit-large-patch14-336 [8] using 0.8 similarity threshold for detection, and the underlying visual question answering module is Salesforce/blip2-flan-t5-xxl [5] loaded in 8-bits using Bit-

Image	Question	Answer
	GQA	
	Are there any guys to the right of the brown horse?	no
	Which direction is the animal that looks white and brown looking at?	forward
	What type of animal is that fence behind of, an elephant or a giraffe?	giraffe
	SugarCREPE	
	Is there a white pitcher holding flowers in a window sill?	yes
	Are a cat and a dog napping together under a blanket on the couch?	no
	Is a dog sitting in front of a laptop on top of a bed?	yes
	WinoGround	
	Verify image matches text="two humans and one wheel"	yes
	Verify image matches text="red building with white shutters"	no
	Verify image matches text="the person with the white collared shirt waters the plant while the other holds it"	yes

Table 1. Dataset Samples

# Samples	# Images	# Questions	# Answers	# Question Types	# Questions/Type
GQA					
1022/502	1014/487	937/474	176/122	105/102	10/5
WinoGround					
-/1600	-/800	-/800	-/2	-/70	-/8
SugarCREPE					
788/560	335/260	765/557	2/2	7/7	52/80

Table 2. Dataset Statistics: Values are shown in {train/test} format. For SugarCREPE and WinoGround, both positive and negative image-text pairings are included. In GQA, question types are divided by the data field `group`, and in WinoGround by the data field `tag`. The training data for WinoGround consists of SugarCREPE.

Algorithm 1 Unit Test Sampling Algorithm

```
Require: $T = \{t_1, t_2, \ldots, t_n\}$, the set of texts
Require: $A = \{a_1, a_2, \ldots, a_m\}$, the set of answers
Require: $f : T \to A$, a function mapping each text to an
    answer
Require: $E(t)$, embedding function for text $t$
Require: $k$, number of samples
Require: use_answers, a boolean flag
Ensure: $S$, a subset of $T$ of size $k$
function SAMPLETEXTS(T, A, f, E, k, use_answers)
    Initialize $S \leftarrow \emptyset$
    if use_answers = True then
        for each $a_i \in A$ do
            Select $t$ from $T$ such that $f(t) = a_i$
            $S \leftarrow S \cup \{t\}$
            $T \leftarrow T \setminus \{t\}$
        end for
    else
        Select a random $t$ from $T$
        $S \leftarrow \{t\}$
        $T \leftarrow T \setminus \{t\}$
    end if
    while $|S| < k$ do
        $s_{\text{new}} \leftarrow \arg\max_{t \in T} \min_{s \in S} \|E(t) - E(s)\|$
        $S \leftarrow S \cup \{s_{\text{new}}\}$
        $T \leftarrow T \setminus \{s_{\text{new}}\}$
    end while
    return $S$
end function
```

sAndBytes with a maximum batch size of 4 and generation hyperparameters `length_penalty=-1`, `num_beams=5`, `max_length=10`,`min_length=1`,`do_sample=False`, `top_p=0.9`, `repetition_penalty=1.0`, and `temperature=1` for QA and set `length_penalty=1` and `max_length=30` for captioning. All models are served by HuggingFace.

C.3. In-Context Examples

We present the in-context examples used for visual question answering and image-text matching in Codes 2 and 3 respectively. Code execution is handled using multiprocessing with a batch size of 30, and a timeout of 120 seconds, after which a `TimeOutException` is raised if execution exceeds the limit.

D. Unit Test Generation

D.1. Implementation Details

To generate the unit test image descriptions and expected answers we prompt meta-llama/Meta-Llama-3-8B-Instruct, executed via VLLM with the following generation parameters: `temperature=0.7`, `top_p=0.9`, `top_k=0.0`, `max_new_tokens=512`, and `num_beams=1`. We return 3 output sequences, from which we extract the unit tests, deduplicate them, and filter answers longer than five words since they are out of distribution to the task before feeding them to the sampling module.

D.2. In-Context Examples

We prompt the LLM with the system prompt presented below, as well as in-context examples presented in Codes 6 and 7 for VQA and ITM respectively.

```
You are a skilled AI assistant
specialized in generating test
cases for programs that respond
to queries about images.
```

D.3. Unit Test Candidate Generation

We experiment with two prompting methodologies for the unit test generation: `Query-Only` and `Query+Implementation`. The former only takes into account the user query to generate the unit-tests, while the latter takes into account also each generated program. We prompt the Visual Program Generator in the same way, but instead also include implementation examples and the current implementation as shown in Code 8.

D.4. Image Generation

To generate the images we use the diffusers library, and prompt each of the models with generation hyperparameters `guidance_scale=16.0` and `num_inference_steps=50`. In the case of NSFW image generation, we update the seed by 1 and regenerate an image up to 10 times. Effectively, all unit tests have a corresponding image. We use the following implementations: CompVis/stable-diffusion-v1-4 for SDv1.4, longlian/lmd_plus for LM Guided Diffusion, and stabilityai/stable-diffusion-xl-base-1.0 for SDXL3.

D.4.1. LM Grounded Diffusion

To generate the bounding boxes and phrases for LM Grounded Diffusion we prompt meta-llama/Meta-Llama-3-8B-Instruct, executed via VLLM with the following generation parameters: `temperature=1.0`, `top_p=0.9`, `top_k=0.0`, `max_new_tokens=320`, and `num_beams=1`. We return 5 candidate sequences to collect multiple candidates since we notice that often the extracted phrases can be empty, leading to failure in image generation. We present the prompt and in-context examples used for this part in Code 9.

E. Strategies for Visual Unit Test Generation

E.1. Unit Test Sampler σ

Figure 1 illustrates the impact of different sampling strategies with varying the number of unit tests and program configurations. Our results indicate that 'Coverage by Answer then Input', consistently outperforms other methods. To gain deeper insights, we categorize the questions into three groups: `Spatial`, `Attribute`, and `Other`. For GQA, we classify any question groups containing `Attr` as `Attribute` and those mentioning `location` or `position` as `Spatial`. Figure 2 presents the average performance across scenarios with at least five unit tests and three program configurations. Notably, the Coverage by Answer Then Input strategy emerges as the most effective for questions in the `Attribute` category.

E.2. Image Generator M

Figure 3 shows the impact of various diffusion models across different numbers of unit tests and program configurations. Our analysis reveals that LM-Guided diffusion consistently outperforms other methods, particularly in scenarios with more programs, where the likelihood of finding a suitable program for execution is higher. To gain deeper insights, figure 2 presents the average performance across scenarios with at least three unit tests and two program configurations on the categories introduced in the previous subsection. To provide a deeper understanding, Figure 4 illustrates the average performance across scenarios involving at least three unit tests and two program configurations, focusing on the categories defined earlier. Notably, LM-Guided diffusion proves most effective for questions in the `Spatial` category, highlighting the advantages of more controllable generation in achieving higher spatial fidelity.

E.3. Scoring function h

Figure 5 highlights the impact of error penalties across varying configurations of unit tests and programs. While their effect becomes negligible in higher-resource configurations with more programs and unit tests, error penalties prove beneficial in lower-resource settings. In these scenarios, they help prioritize the selection of executable programs, thereby improving performance. Notably, runtime error penalties are more impactful for GQA, whereas compilation error penalties play a larger role in WinoGround. This difference likely stems from the higher complexity of WinoGround programs,

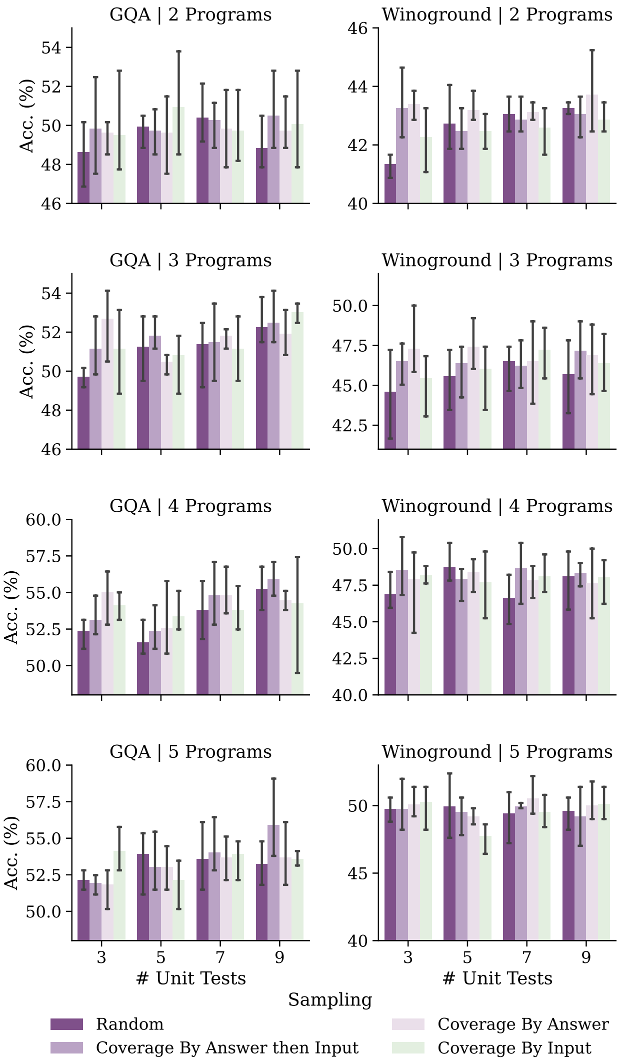


Figure 1. Effect of sampling methods on performance across varying numbers of unit tests and program configurations.

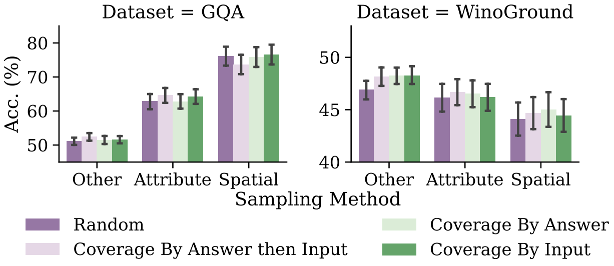


Figure 2. Performance of sampling methods across question categories. Results are averaged over scenarios with at least five unit tests and three program configurations.

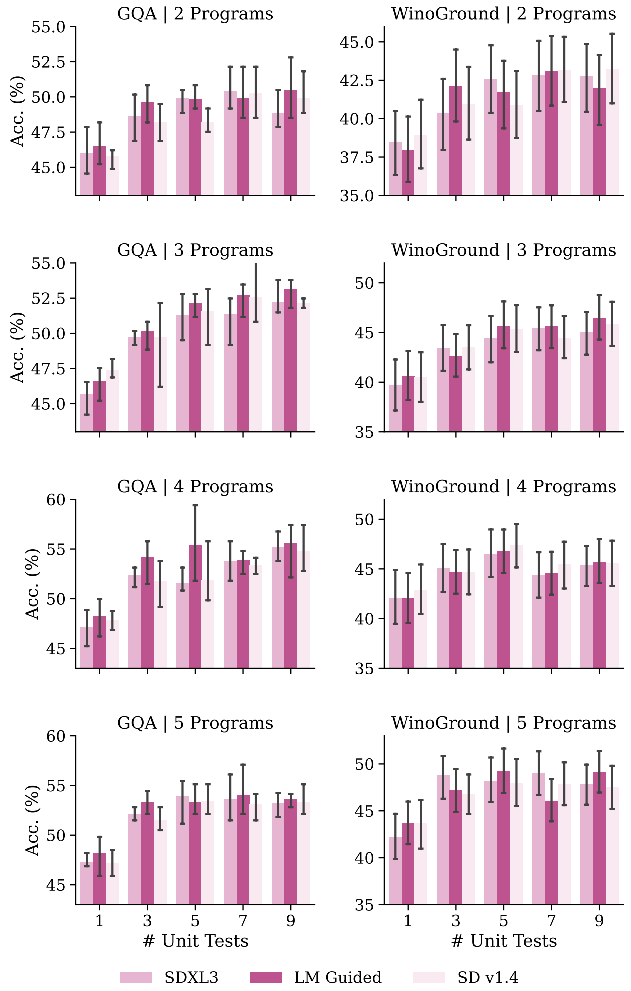


Figure 3. Effect of diffusion model on performance across varying numbers of unit tests and program configurations.

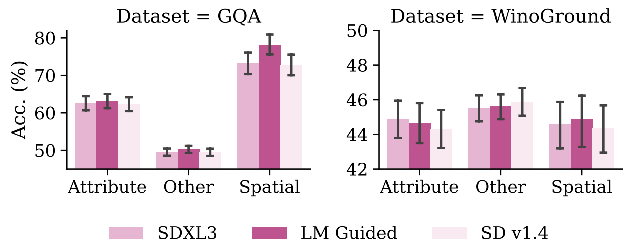


Figure 4. Performance of different diffusion models across question categories. Results are averaged over scenarios with at least three unit tests and two program configurations.

which are more prone to compilation errors.

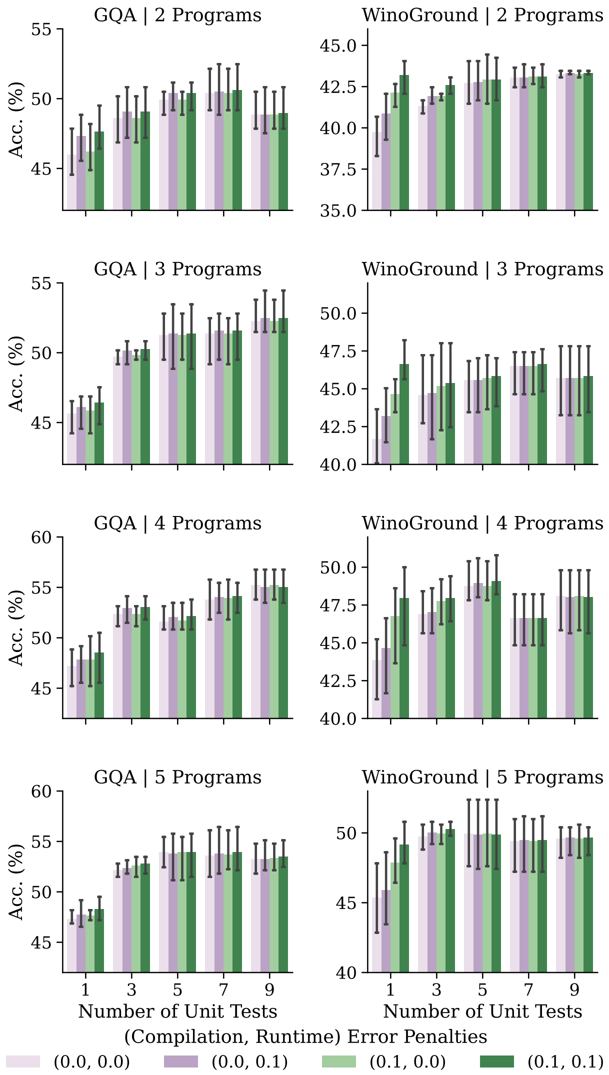


Figure 5. Effect of error penalties on accuracy.

E.4. Aggregate Scorer H

Figure 6 illustrates the impact of various aggregator functions on accuracy. Among these, mean score aggregation consistently outperforms other methods, particularly in configurations with a higher number of programs. In the case of WinoGround, however, max aggregation also performs competitively, occasionally surpassing mean aggregation. This is likely due to the binary nature of the answers in WinoGround and the increased likelihood of selecting correct for incorrect reasons programs.

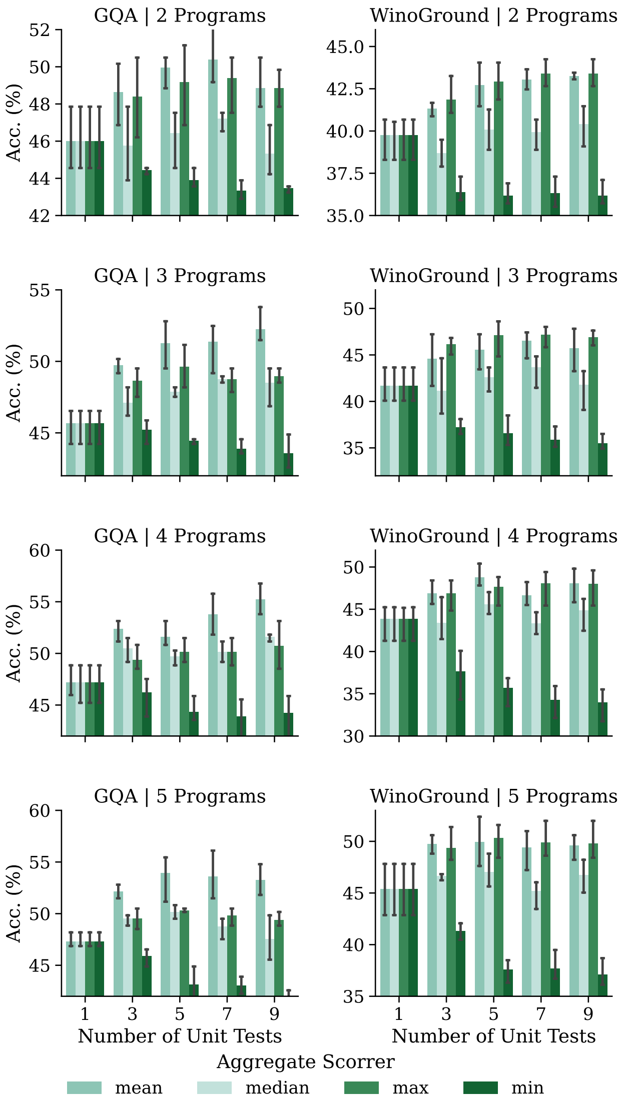


Figure 6. Effect of aggregator function on accuracy.

F. Visual Unit Test Utilization Methods

F.1. Best Program Selection

Table 3 shows additional results on best program selection with varrying number of programs.

F.2. Answer Refusal

Figure 7 shows additional statistics on answer refusal, in particular the accuracy of selecting programs that will provide the correct final answer and the programs that succeed on the unit tests at different thresholds.

			VQA	Image-Text Matching		
LLM	# Prog	# UT	GQA	Winoground	SugarCREPE	Avg.
			Base Setup			
gpt-4o-mini	1	0	42.03$_{\pm 1.21}$	44.98$_{\pm 0.75}$	38.75$_{\pm 0.47}$	41.92$_{\pm 0.81}$
CodeLlama-7B	1	0	35.99$_{\pm 2.94}$	38.83$_{\pm 0.45}$	30.54$_{\pm 0.99}$	35.12$_{\pm 1.46}$
CodeGemma-7B	1	0	41.83$_{\pm 2.26}$	39.60$_{\pm 1.38}$	42.56$_{\pm 1.52}$	41.33$_{\pm 1.72}$
			Most Common Answer Setup			
CodeLlama-7B	2	0	27.76$_{\pm 0.41}$	36.19$_{\pm 0.66}$	32.02$_{\pm 2.25}$	31.99$_{\pm 1.11}$
CodeLlama-7B	3	0	35.99$_{\pm 0.70}$	42.40$_{\pm 0.85}$	37.26$_{\pm 2.70}$	38.55$_{\pm 1.42}$
CodeLlama-7B	4	0	38.71$_{\pm 1.61}$	42.12$_{\pm 0.60}$	39.17$_{\pm 2.01}$	40.00$_{\pm 1.41}$
CodeLlama-7B	5	0	42.50$_{\pm 1.50}$	45.85$_{\pm 0.77}$	41.67$_{\pm 1.79}$	43.34$_{\pm 1.35}$
CodeGemma-7B	2	0	31.87$_{\pm 0.80}$	33.04$_{\pm 0.67}$	36.37$_{\pm 1.62}$	33.76$_{\pm 1.03}$
CodeGemma-7B	3	0	40.31$_{\pm 1.00}$	40.50$_{\pm 1.33}$	44.58$_{\pm 0.55}$	41.80$_{\pm 0.96}$
CodeGemma-7B	4	0	40.44$_{\pm 0.53}$	43.06$_{\pm 1.89}$	44.46$_{\pm 1.17}$	42.66$_{\pm 1.20}$
CodeGemma-7B	5	0	43.89$_{\pm 0.98}$	46.04$_{\pm 1.48}$	46.67$_{\pm 1.69}$	45.53$_{\pm 1.38}$
			ViUniT Setup (Ours)			
CodeLlama-7B	2	5	41.90$_{\pm 1.74}$	46.65$_{\pm 1.63}$	40.24$_{\pm 0.82}$	42.93$_{\pm 1.40}$
CodeLlama-7B	3	5	45.68$_{\pm 0.94}$	48.54$_{\pm 0.37}$	43.93$_{\pm 1.09}$	46.05$_{\pm 0.80}$
CodeLlama-7B	4	5	49.07$_{\pm 2.39}$	50.17$_{\pm 0.54}$	45.65$_{\pm 1.22}$	48.30$_{\pm 1.38}$
CodeLlama-7B	5	5	**49.27**$_{\pm 1.13}$	49.73$_{\pm 0.73}$	47.02$_{\pm 1.19}$	48.67$_{\pm 1.02}$
CodeGemma-7B	2	5	44.02$_{\pm 0.72}$	49.27$_{\pm 0.57}$	46.73$_{\pm 2.30}$	46.67$_{\pm 1.20}$
CodeGemma-7B	3	5	46.08$_{\pm 0.41}$	51.17$_{\pm 1.98}$	48.93$_{\pm 1.86}$	48.73$_{\pm 1.42}$
CodeGemma-7B	4	5	47.88$_{\pm 1.36}$	**52.25**$_{\pm 1.35}$	50.83$_{\pm 1.32}$	50.32$_{\pm 1.34}$
CodeGemma-7B	5	5	48.01$_{\pm 1.05}$	51.92$_{\pm 0.90}$	**51.85**$_{\pm 2.16}$	**50.59**$_{\pm 1.37}$

Table 3. Accuracy on Best Program Selection with varying number of programs. **Bold** is best.

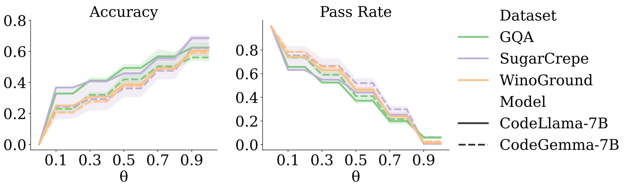


Figure 7. Accuracy and Program Pass Rate for different thereshold values for answer refusal.

F.3. Re-prompting

F.3.1. Implementation Details

We consider an application of the unit tests to generate different candidate programs if the generated program falls below a threshold. To do so, we maintain the same hyperparameters in the program generator and adapt the prompt to include the outputs of the unit tests as well as use suitable in context examples as shown in Codes 4 and 5 for VQA and ITM respectively.

Error Reprompting Baseline We employ the same model and hyperparamters as ViUniT reprompting, but instead adapt the prompt to take into account the error messages instead of the unit tests as shown in Codes 10 and 11 for VQA and ITM respectively.

F.3.2. Additional Results

Table 4 presents the results of an additional reprompting iteration, highlighting that while ViUniT continues to achieve higher performance overall, there is a slight drop in accuracy compared to the previous iteration. This decline can be attributed to its attempts to refine programs that may already produce correct answers for the wrong reasons. Such corrections can inadvertently cause shifts in the generated answers, leading to decreased accuracy despite the method's focus on improving program fidelity.

				VQA	Image-Text Matching		
LLM	Iter.	# Prog	# UT	GQA	Winoground	SugarCREPE	Avg.
				Base Setup (Iteration = 0)			
CodeLlama-7B	0	1	0	$35.99_{\pm 2.94}$	$38.83_{\pm 0.45}$	$30.54_{\pm 0.99}$	$35.12_{\pm 1.46}$
CodeGemma-7B	0	1	0	$41.83_{\pm 2.26}$	$39.60_{\pm 1.38}$	$42.56_{\pm 1.52}$	$41.33_{\pm 1.72}$
				Error Reprompting			
CodeLlama-7B	1	1	0	$37.92_{\pm 2.68}$	$42.46_{\pm 0.57}$	$33.21_{\pm 0.64}$	$37.86_{\pm 1.30}$
CodeLlama-7B	2	1	0	$38.78_{\pm 2.22}$	$44.58_{\pm 0.44}$	$37.08_{\pm 1.08}$	$40.15_{\pm 1.25}$
CodeGemma-7B	1	1	0	$42.63_{\pm 2.42}$	$42.42_{\pm 1.91}$	$44.52_{\pm 1.05}$	$42.63_{\pm 2.42}$
CodeGemma-7B	2	1	0	$42.90_{\pm 2.65}$	$43.08_{\pm 1.73}$	$45.30_{\pm 0.92}$	$42.90_{\pm 2.65}$
				ViUniT Reprompting $\theta = 0.7$ (Ours)			
CodeLlama-7B	1	1	5	**46.68**$_{\pm 2.52}$	**51.85**$_{\pm 0.40}$	**47.68**$_{\pm 2.17}$	**48.74**$_{\pm 1.69}$
CodeLlama-7B	2	1	5	**46.95**$_{\pm 1.33}$	**52.04**$_{\pm 0.83}$	**48.04**$_{\pm 1.64}$	**49.01**$_{\pm 1.26}$
CodeGemma-7B	1	1	5	**45.75**$_{\pm 0.30}$	**48.19**$_{\pm 2.28}$	**48.21**$_{\pm 1.12}$	**47.38**$_{\pm 1.23}$
CodeGemma-7B	2	1	5	**44.42**$_{\pm 1.00}$	**49.25**$_{\pm 2.66}$	**48.81**$_{\pm 1.19}$	**47.49**$_{\pm 1.62}$

Table 4. Accuracy of different re-prompting methods with an additional iteration. **Bold** is best.

F.4. Reward Design for Reinforcement Learning

F.4.1. Implementation Details

Table 5 contains additional hyperparameters used for training. Each RL epoch requires about 30 minutes with correctness reward, and 90 minutes with ViUniT reward since it requires execution of unit tests.

Parameter	Value
`warmup_ratio`	0.1
`max_grad_norm`	0.3
`lr_scheduler_type`	`linear`
`learning_rate`	2e-4
`lora_config.r`	16
`lora_config.lora_alpha`	32
`lora_config.lora_dropout`	0.05
`lora_config.bias`	`none`

`lora_config.target_modules`	`k_proj` `v_proj`
	`q_proj` `o_proj`

Table 5. RL training hyperparameters.

F.4.2. Additional Analysis

Table 6 highlights the reduced error rates—measured as the number of programs leading to exceptions—achieved using the ViUniT reward. Additionally, Table 7 presents the results of cross-task and cross-dataset generalization on policies trained with GQA, following the approach of [4]. For VQAv2 [1], we sample 10 questions for each of the 50 most common answers from the validation split of the compositional subset curated by [9], similar to [4]. For OKVQA [7], we sample 10 questions per question type, resulting in a total of 110 questions. The results indicate that while both reward types demonstrate strong generalization across tasks and datasets, the ViUniT reward consistently delivers superior performance.

			VQA	Image-Text Matching		
LLM	# Prog	# UT	GQA	Winoground	SugarCREPE	Avg.
			Supervised Correctness Reward			
CodeLlama-7B	1	0	$15.14_{\pm 7.74}$	**8.21**$_{\pm 1.72}$	$20.06_{\pm 3.62}$	$14.47_{\pm 4.36}$
CodeGemma-7B	1	0	$9.10_{\pm 9.35}$	$13.25_{\pm 6.30}$	$12.86_{\pm 4.41}$	$11.73_{\pm 6.69}$
			Unsupervised ViUniT Reward (Ours)			
CodeLlama-7B	1	0	**9.56**$_{\pm 2.13}$	$10.31_{\pm 1.55}$	**15.42**$_{\pm 3.03}$	**11.76**$_{\pm 2.24}$
CodeGemma-7B	1	0	**1.99**$_{\pm 0.91}$	**5.81**$_{\pm 0.49}$	**6.25**$_{\pm 1.02}$	**4.68**$_{\pm 0.80}$

Table 6. Comparison of *Error Rates* in models trained with supervised correctness rewards versus unsupervised unit-test-based rewards. Lower is better. **Bold** is best.

			X-Dataset Generalization		X-Task Generalization	
LLM	# Prog	# UT	VQAv2	OK-VQA	Winoground	SugarCREPE
			Base Setup			
CodeLlama-7B	1	0	$25.67_{\pm 2.20}$	$16.09_{\pm 2.02}$	$30.54_{\pm 0.99}$	$35.12_{\pm 1.46}$
CodeGemma-7B	1	0	$36.40_{\pm 1.44}$	$27.58_{\pm 2.48}$	$42.56_{\pm 1.52}$	$41.33_{\pm 1.72}$
			Supervised Correctness Reward			
CodeLlama-7B	1	0	$34.33_{\pm 7.82}$	$24.12_{\pm 5.98}$	$41.02_{\pm 3.05}$	$37.14_{\pm 6.48}$
CodeGemma-7B	1	0	$42.47_{\pm 6.03}$	$28.12_{\pm 6.20}$	$47.98_{\pm 4.98}$	$39.94_{\pm 11.58}$
			Unsupervised ViUniT Reward (Ours)			
CodeLlama-7B	1	0	**35.87**$_{\pm 2.31}$	**25.64**$_{\pm 0.91}$	**43.63**$_{\pm 2.89}$	**44.35**$_{\pm 3.18}$
CodeGemma-7B	1	0	**44.00**$_{\pm 4.20}$	**36.85**$_{\pm 3.48}$	**51.78**$_{\pm 0.41}$	**49.23**$_{\pm 2.54}$

Table 7. GQA policy generalization across tasks and datasets

G. End-to-End Fallback Methods

G.1. Implementation Details

G.1.1. VQA

For VQA we revert to ask the query directly to Salesforce/blip2-flan-t5-xxl [5] loaded in 8-bits using BitsAndBytes with a maximum batch size of 4 and generation hyperparameters `length_penalty=-1`, `num_beams=5`, `max_length=10`,`min_length=1`,`do_sample=False`, `top_p=0.9`, `repetition_penalty=1.0`, and `temperature=1`.

G.1.2. Image-Text-Matching

For image-text-matching we revert to openai/clip-vit-large-patch14-336 [8] using 0.8 similarity threshold for positive match, and negative otherwise.

G.2. Results with Fallback Method on Exception

In this work, we report results without employing a fallback method on exceptions, treating such cases as failures to better assess the quality of programs generated by different methods. However, it is common in the literature to report accuracy with a fallback method applied on exceptions. In Table 8 we present the best program selection results using this fallback approach on error.

H. Human Evaluation

This section presents details on the human evaluations on the quality of unit tests, and program correctness. We used Google-Forms to conduct the evaluations.

H.1. Unit Test Evaluation

To assess the quality of unit tests we randomly sample 20 examples from each of the three datasets, each correspond-

			VQA	Image-Text Matching		
LLM	# Prog	# UT	GQA	Winoground	SugarCREPE	Avg.
			Base Setup			
gpt-4o-mini†	1	0	$43.76_{\pm 1.72}$	$\mathbf{51.94}_{\pm 0.56}$	$49.46_{\pm 1.25}$	$48.39_{\pm 1.17}$
CodeLlama-7B†	1	0	$44.75_{\pm 2.01}$	$51.65_{\pm 1.09}$	$48.57_{\pm 0.82}$	$48.32_{\pm 1.31}$
CodeGemma-7B†	1	0	$44.82_{\pm 2.30}$	$47.23_{\pm 2.26}$	$50.18_{\pm 0.71}$	$47.41_{\pm 1.76}$
			Most Common Answer Setup			
CodeLlama-7B†	5	0	$49.07_{\pm 2.79}$	$51.29_{\pm 0.87}$	$46.79_{\pm 1.29}$	$49.05_{\pm 1.65}$
CodeGemma-7B†	5	0	$46.61_{\pm 1.24}$	$49.10_{\pm 1.32}$	$49.17_{\pm 1.52}$	$48.29_{\pm 1.36}$
			ViUniT Setup (Ours)			
CodeLlama-7B†	5	5	$\mathbf{49.27}_{\pm 1.33}$	$49.73_{\pm 0.73}$	$47.02_{\pm 1.19}$	$\mathbf{48.67}_{\pm 1.08}$
CodeGemma-7B†	5	5	$\mathbf{48.14}_{\pm 1.02}$	$51.92_{\pm 0.90}$	$\mathbf{51.85}_{\pm 2.16}$	$\mathbf{50.63}_{\pm 1.36}$

Table 8. Accuracy on Best Program Selection using fallback method on exception (indicated by †). **Bold** is best.

ing to 5 unit tests, resulting in a total of 300 unit tests for evaluation. The unit tests were judged by three independent annotators, instructed with `Is the answer answer correct given the image?`, where `answer` was populated with the unit test expected answer, expecting binary yes/no answers. Table 9 breaks down the results showing that on average 75% of unit tests are correct. Then the annotators optionally annotated the reason of failure by selecting from "Missing Object", "Spatial Error", "Incomplete object", "Color Mismatch", or "Other". Figure 8 shows the break down by error type, highlighting "Missing Object" as the most common source of error.

GQA		WinoGround		SugarCREPE		Avg.	
Acc.	κ	Acc.	κ	Acc.	κ	Acc.	κ
68.00	0.39	75.00	0.70	82.00	0.67	75.00	0.58

Table 9. Human Evaluation of Unit Test Quality. Accuracy corresponds to how many unit tests from the total were accurate and κ is the mean Kohen Kappa across annotators.

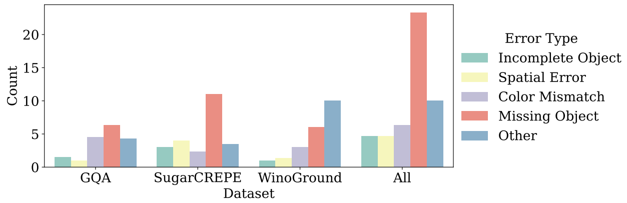


Figure 8. Human Evaluation of Unit Test Quality. Bars show the average number of times annotators selected a source of error.

H.2. Program Correctness Evaluation

To assess the improvements on program quality by applying ViUniT we conduct a human evaluation to rate GQA programs generated by the Base Setup and the programs selected from 5 candidate programs and 5 unit tests. Two annotators with 3+ years of Python experience graded programs using the following grading scheme: "Correct: The code accurately and fully answers the query." (0), "Partially Correct: The code answers the query but has some issues." (1), "Incorrect: The code does not answer the query correctly." (2), and "Irrelevant: The code is unrelated to the query." (3). In addition, they were optionally asked to select the source of error from "Missing Condition", "Incorrect Logic", "Irrelevant to the query", "Wrong Conditions", "Missing Checks (e.g. could get list index out of range)", "Performance Issues", "Other". Table 10 shows the breakdown of program correctness improvements using ViUniT and Figure 9 shows the error types identified in each method. ViUniT has "Missing Checks" as the most common error type, which mostly involves cases of not checking array length before accessing indices, typically still leading to correct solutions with reasonable programs, whereas the main culprit for program incorrectness in the base setup is "Incorrect Logic". This pattern of error redistribution occurs because unit tests disqualify programs likely correct for the wrong reasons. Irrelevant programs rarely pass multiple tests, while errors like wrong or missing conditions are less likely but possible. Missing checks (e.g., checking array length prior to access) often pass due to well-formatted inputs, which explains their persistence even post unit-testing.

	Base Setup	ViUniT Setup (Ours)
Fully Correct (≤ 1)	77%	**86%**
Partially Correct (< 2)	86%	**95%**
Incorrect (≥ 2)	14%	**5%**
Irrelevant (> 2)	4%	**0%**
κ	0.24	0.30
κ_{bin}	0.59	0.40

Table 10. Human Evaluation of Program Correctness. **Bold** is best.

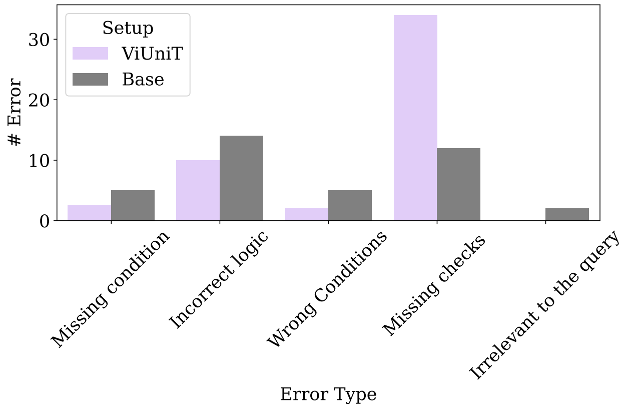


Figure 9. Human Evaluation of Program Quality.

I. Limitations and Social Ethics Impact

I.1. Limitations

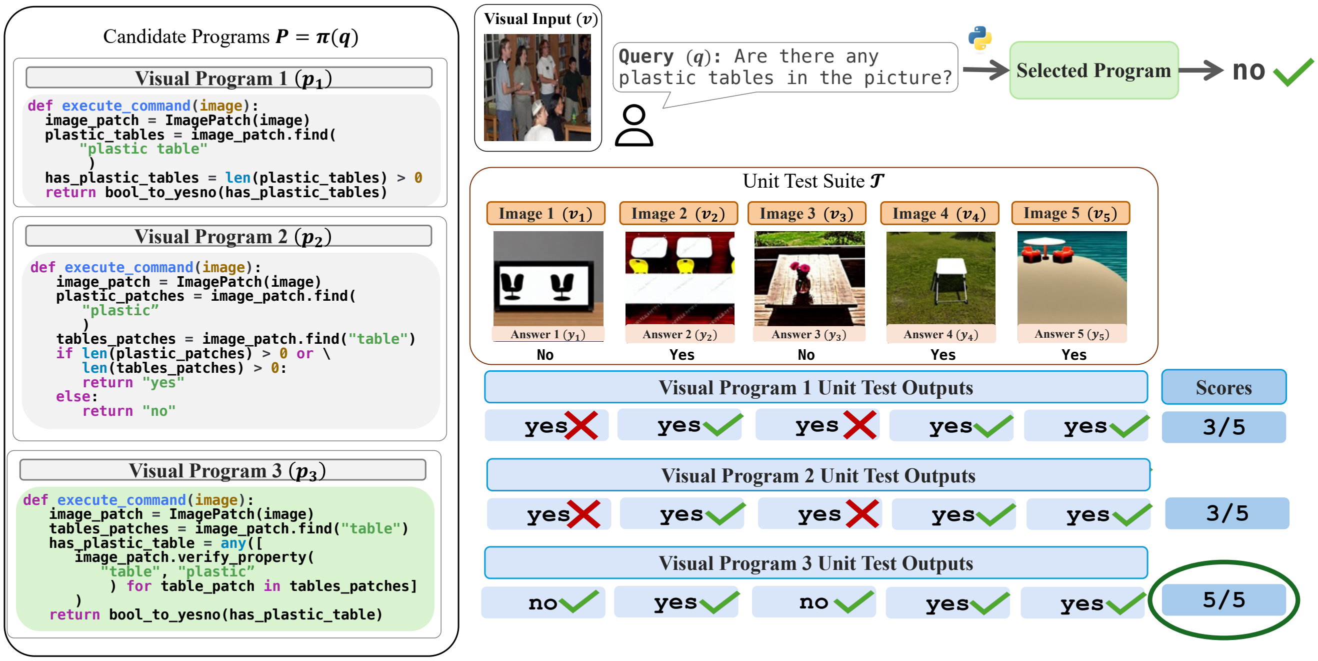
While ViUniT provides significant advancements in the logical correctness and robustness of visual programs, our framework has several limitations that present opportunities for future enhancement. First, although ViUniT improves program selection and execution by leveraging unit tests, it does not fully eliminate the issue of programs being correct for the wrong reasons, as shown by the human evaluation in Table 10. Our approach does not provide a formal guarantee of logical correctness, as it relies on automatically generated tests to evaluate candidate programs. Addressing this challenge opens avenues for integrating formal verification methods and more sophisticated testing strategies to further enhance program correctness. Second, while we optimize for maximizing input and output coverage during unit test generation, it is possible that the generated tests do not fully capture the space of edge cases or subtle logical errors in complex programs. This limitation highlights the potential for future work to develop more comprehensive coverage metrics and testing methodologies, possibly incorporating code-line execution coverage or other verifiable metrics. Third, the improved accuracy and robustness achieved by ViUniT , as seen in Table 1, come with an increase in computational effort. Generating candidate programs, sampling unit tests, and executing them on generated images introduce additional overhead. This trade-off between accuracy and efficiency presents an exciting challenge for future research to optimize the framework for real-time or resource-constrained applications, possibly through algorithmic improvements or efficient execution strategies. Additionally, enhancing the explainability of program failures remains an area for further development. Providing clear and interpretable feedback when a program is rejected or not selected due to poor performance on unit tests can improve user trust and facilitate debugging. Future work could focus on combining unit test outputs to offer detailed explanations of program failures. Finally, while ViUniT has demonstrated effectiveness on VQA and ITM tasks, exploring its applicability to other domains or tasks involving different modalities or reasoning paradigms presents an opportunity to extend its impact. Adapting the framework to diverse domains can unlock new possibilities and broaden its utility. Despite these limitations, the advancements introduced by ViUniT lay a strong foundation for future innovations in visual programming. By addressing these challenges, we can further enhance the robustness, efficiency, and applicability of the framework.

I.2. Social Ethics Impact

ViUniT enhances the robustness and correctness of visual programming, with applications in critical domains like autonomous driving, healthcare, and education. By reducing instances where programs are correct for the wrong reasons, it helps build more trustworthy AI systems. However, ethical considerations are crucial for its responsible deployment: First, ViUniT relies on pre-trained models, which may propagate biases (e.g., gender, racial, or cultural). Future work should focus on integrating bias detection and correction into unit test generation to promote fairness. Second, computational demands may limit access for resource-constrained organizations. Advancing efficiency and optimization can broaden accessibility and foster inclusivity. Third, increased computational needs may raise energy consumption. Optimizing for energy efficiency and using renewable energy can reduce the environmental impact, while improved AI reliability could deliver long-term sustainability benefits. Finally, in sensitive domains such as healthcare or legal decision-making, while ViUniT has the potential to enhance the correctness of visual programs, it is crucial to carefully communicate the framework's limitations and ensure rigorous validation. By proactively addressing ethical challenges and focusing on responsible development, we can maximize the positive societal impact of ViUniT, paving the way for more reliable, fair, and trustworthy AI systems.

J. Qualitative Examples

We present two program selection examples in Figures 10 and 11. While all programs may pass some unit tests, those that pass a greater number of tests tend to more effectively capture the intent of the question. In Figure 11, the selected program does not pass all unit tests—some of which cover edge-cases not handled by the program. Nevertheless, it is chosen because it sufficiently addresses the core intent of the user's query.

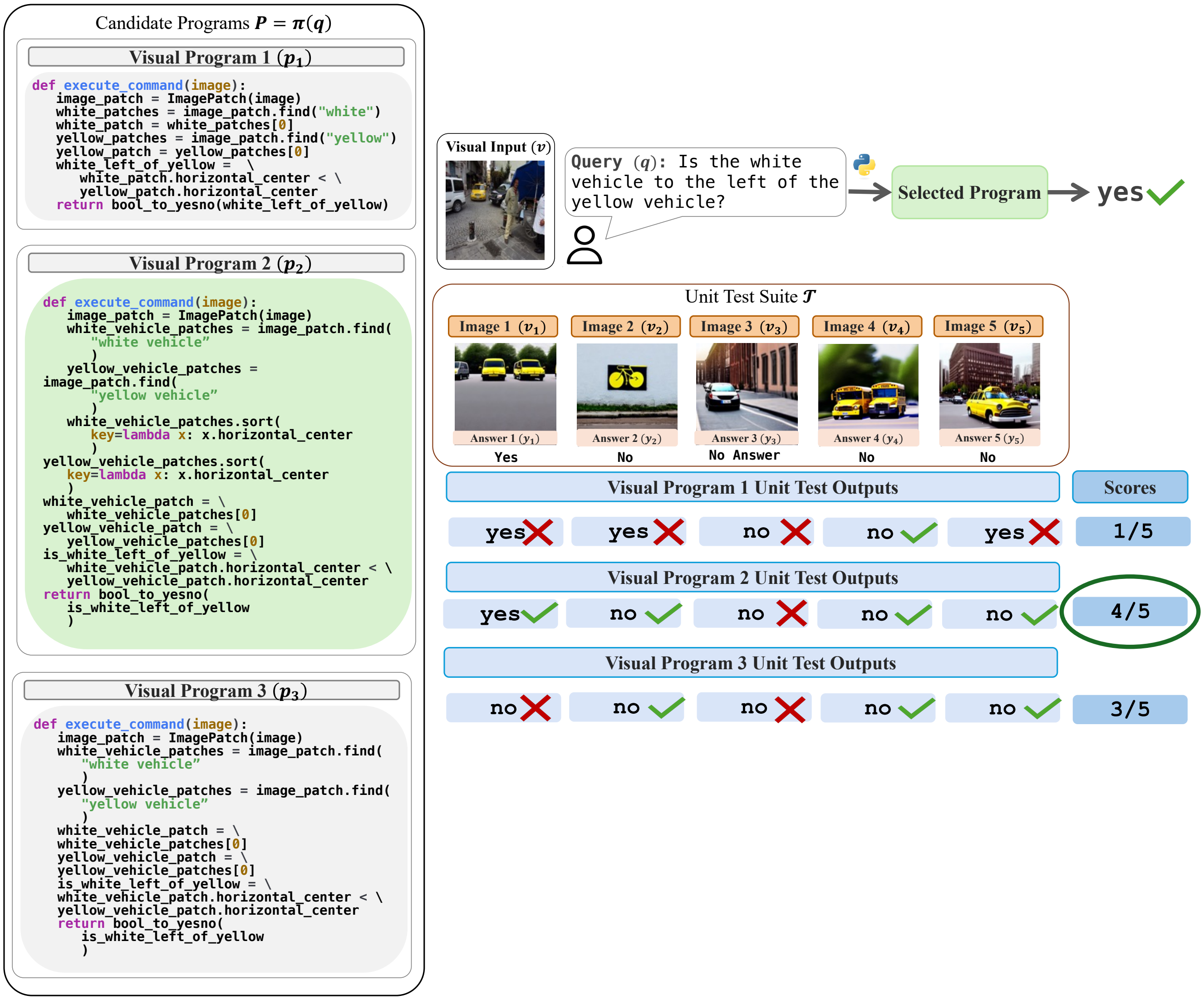

Candidate Programs $P = \pi(q)$
Visual Program 1 (p_1)
Visual Program 2 (p_2)
Visual Program 3 (p_3)
Visual Input (v)
Query (q): Are there any plastic tables in the picture?
Selected Program
no ✓
Unit Test Suite $\mathcal{T}$
Image 1 (v_1)
Image 2 (v_2)
Image 3 (v_3)
Image 4 (v_4)
Image 5 (v_5)
Answer 1 (y_1)
No
Answer 2 (y_2)
Yes
Answer 3 (y_3)
No
Answer 4 (y_4)
Yes
Answer 5 (y_5)
Yes
Visual Program 1 Unit Test Outputs
Scores
yes ✗ yes ✓ yes ✗ yes ✓ yes ✓
3/5
Visual Program 2 Unit Test Outputs
yes ✗ yes ✓ yes ✗ yes ✓ yes ✓
3/5
Visual Program 3 Unit Test Outputs
no ✓ yes ✓ no ✓ yes ✓ yes ✓
5/5

```python
def execute_command(image):
  image_patch = ImagePatch(image)
  plastic_tables = image_patch.find(
      "plastic table"
       )
  has_plastic_tables = len(plastic_tables) > 0
  return bool_to_yesno(has_plastic_tables)
```

```python
def execute_command(image):
   image_patch = ImagePatch(image)
   plastic_patches = image_patch.find(
      "plastic”
      )
   tables_patches = image_patch.find("table")
   if len(plastic_patches) > 0 or \
      len(tables_patches) > 0:
      return "yes"
   else:
      return "no"
```

```python
def execute_command(image):
   image_patch = ImagePatch(image)
   tables_patches = image_patch.find("table")
   has_plastic_table = any([
      image_patch.verify_property(
         "table", "plastic”
          ) for table_patch in tables_patches]
     )
   return bool_to_yesno(has_plastic_table)
```

Figure 10. Program Selection Example


Candidate Programs $\boldsymbol{P} = \boldsymbol{\pi}(\boldsymbol{q})$
Visual Program 1 ($\boldsymbol{p}_1$)
Visual Program 2 ($\boldsymbol{p}_2$)
Visual Program 3 ($\boldsymbol{p}_3$)
Visual Input (v)
Query (q): Is the white vehicle to the left of the yellow vehicle?
Selected Program
yes ✓
Unit Test Suite $\mathcal{T}$
Image 1 (v_1)
Image 2 (v_2)
Image 3 (v_3)
Image 4 (v_4)
Image 5 (v_5)
Answer 1 (y_1)
Yes
Answer 2 (y_2)
No
Answer 3 (y_3)
No Answer
Answer 4 (y_4)
No
Answer 5 (y_5)
No
Visual Program 1 Unit Test Outputs
yes ✗ yes ✗ no ✗ no ✓ yes ✗
Scores
1/5
Visual Program 2 Unit Test Outputs
yes ✓ no ✓ no ✗ no ✓ no ✓
4/5
Visual Program 3 Unit Test Outputs
no ✗ no ✓ no ✗ no ✓ no ✓
3/5

```
def execute_command(image):
   image_patch = ImagePatch(image)
   white_patches = image_patch.find("white")
   white_patch = white_patches[0]
   yellow_patches = image_patch.find("yellow")
   yellow_patch = yellow_patches[0]
   white_left_of_yellow =  \
      white_patch.horizontal_center < \
      yellow_patch.horizontal_center
   return bool_to_yesno(white_left_of_yellow)
```

```
def execute_command(image):
   image_patch = ImagePatch(image)
   white_vehicle_patches = image_patch.find(
      "white vehicle”
      )
   yellow_vehicle_patches =
image_patch.find(
      "yellow vehicle”
      )
   white_vehicle_patches.sort(
      key=lambda x: x.horizontal_center
      )
yellow_vehicle_patches.sort(
   key=lambda x: x.horizontal_center
   )
white_vehicle_patch = \
   white_vehicle_patches[0]
yellow_vehicle_patch = \
   yellow_vehicle_patches[0]
is_white_left_of_yellow = \
   white_vehicle_patch.horizontal_center < \
   yellow_vehicle_patch.horizontal_center
return bool_to_yesno(
   is_white_left_of_yellow
   )
```

```
def execute_command(image):
   image_patch = ImagePatch(image)
   white_vehicle_patches = image_patch.find(
      "white vehicle”
      )
   yellow_vehicle_patches = image_patch.find(
      "yellow vehicle”
      )
   white_vehicle_patch = \
   white_vehicle_patches[0]
   yellow_vehicle_patch = \
   yellow_vehicle_patches[0]
   is_white_left_of_yellow = \
   white_vehicle_patch.horizontal_center < \
   yellow_vehicle_patch.horizontal_center
   return bool_to_yesno(
      is_white_left_of_yellow
      )
```

Figure 11. Program Selection Example

Listing 1. API Prompt

```
import math

class ImagePatch:
    pass

    def __init__(
        self, image, left=None, lower=None, right=None, upper=None, category=None
    ):
        """Initializes an ImagePatch object by cropping the image at the given
        coordinates and stores the coordinates as attributes. If no coordinates are
        provided, the image is left unmodified, and the coordinates are set to the
        dimensions of the image.
        Parameters
        -------
        image : array_like
            An array-like of the original image.
        left, lower, right, upper : int
            An int describing the position of the (left/lower/right/upper) border of the
             crop’s bounding box in the original image.
        category : str
            A string describing the name of the object in the image."""

        # Rectangles are represented as 4-tuples, (x1, y1, x2, y2),
        # with the upper left corner given first. The coordinate
        # system is assumed to have its origin in the upper left corner, so
        # upper must be less than lower and left must be less than right.

        self.left = left if left is not None else 0
        self.lower = lower if lower is not None else image.height
        self.right = right if right is not None else image.width
        self.upper = upper if upper is not None else 0
        self.cropped_image = image[:, image.shape[1]-upper:image.shape[1]-lower, left:right]
        self.horizontal_center = (self.left + self.right) / 2
        self.vertical_center = (self.upper + self.lower) / 2
        self.category = category

    def from_bounding_box(cls, image, bounding_box):
        """Initializes an ImagePatch object by cropping the image at the given
        coordinates and stores the coordinates as attributes.
        Parameters
        -------
        image : array_like
```

```
            An array-like of the original image.
        bounding_box : dict
            A dictionary like {"box": [left, lower, right, upper], "category": str}."""
        pass

    @property
    def area(self):
        """
        Returns the area of the bounding box.

        Examples
        --------
        >>> # What color is the largest foo?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     foo_patches.sort(key=lambda x: x.area)
        >>>     largest_foo_patch = foo_patches[-1]
        >>>     return largest_foo_patch.simple_query("What is the color?")
        """
        pass

    def find(self, object_name):
        """Returns a list of ImagePatch objects matching object_name contained in the
        crop if any are found.
        Otherwise, returns an empty list.
        Parameters
        ----------
        object_name : str
            the name of the object to be found

        Returns
        -------
        List[ImagePatch]
            a list of ImagePatch objects matching object_name contained in the crop

        Examples
        --------
        >>> # return the foo
        >>> def execute_command(image) -> List[ImagePatch]:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     return foo_patches
        """
        pass

    def exists(self, object_name):
        """Returns True if the object specified by object_name is found in the image,
        and False otherwise.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.

        Examples
        -------
        >>> # Are there both foos and garply bars in the photo?
        >>> def execute_command(image)->str:
        >>>     image_patch = ImagePatch(image)
        >>>     is_foo = image_patch.exists("foo")
        >>>     is_garply_bar = image_patch.exists("garply bar")
        >>>     return bool_to_yesno(is_foo and is_garply_bar)
        """
        pass

    def verify_property(self, object_name, visual_property):
        """Returns True if the object possesses the visual property, and False otherwise.
        Differs from 'exists' in that it presupposes the existence of the object
        specified by object_name, instead checking whether the object possesses
        the property.
        Parameters
        -------
        object_name : str
            A string describing the name of the object to be found in the image.
        visual_property : str
            String describing the simple visual property (e.g., color, shape, material)
            to be checked.

        Examples
        -------
        >>> # Do the letters have blue color?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     letters_patches = image_patch.find("letters")
        >>>     # Question assumes only one letter patch
        >>>     return bool_to_yesno(letters_patches[0].verify_property("letters", "blue"))
        """
        pass

    def simple_query(self, question):
        """Returns the answer to a basic question asked about the image.
        If no question is provided, returns the answer to "What is this?".
        The questions are about basic perception, and are not meant to be used for
        complex reasoning or external knowledge.
        Parameters
        -------
        question : str
            A string describing the question to be asked.

        Examples
        -------

        >>> # Which kind of baz is not fredding?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     baz_patches = image_patch.find("baz")
        >>>     for baz_patch in baz_patches:
        >>>         if not baz_patch.verify_property("baz", "fredding"):
        >>>             return baz_patch.simple_query("What is this baz?")

        >>> # What color is the foo?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patches = image_patch.find("foo")
        >>>     foo_patch = foo_patches[0]
        >>>     return foo_patch.simple_query("What is the color?")

        >>> # Is the second bar from the left quuxy?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     bar_patches = image_patch.find("bar")
        >>>     bar_patches.sort(key=lambda x: x.horizontal_center)
        >>>     bar_patch = bar_patches[1]
        >>>     return bar_patch.simple_query("Is the bar quuxy?")
        """
        pass

    def crop_left_of_bbox(self, left, lower, right, upper):
        """Returns an ImagePatch object representing the area to the left of the given
        bounding box coordinates.

        Parameters
        ----------
        left, lower, right, upper : int
            The coordinates of the bounding box.

        Returns
        -------
        ImagePatch
            An ImagePatch object representing the cropped area.

        Examples
        --------
        >>> # Is the bar to the left of the foo quuxy?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patch = image_patch.find("foo")[0]
        >>>     left_of_foo_patch = image_patch.crop_left_of_bbox(
        >>>         foo_patch.left, foo_patch.lower, foo_patch.right, foo_patch.upper
        >>>     )
        >>>     return bool_to_yesno(left_of_foo_patch.verify_property("bar", "quuxy"))
        """
        pass

    def crop_right_of_bbox(self, left, lower, right, upper):
        """Returns an ImagePatch object representing the area to the right of the given
        bounding box coordinates.
        Parameters
        ----------
        left, lower, right, upper : int
            The coordinates of the bounding box.

        Returns
        -------
        ImagePatch
            An ImagePatch object representing the cropped area.

        Examples
        --------
        >>> # Is the bar to the right of the foo quuxy?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patch = image_patch.find("foo")[0]
        >>>     right_of_foo_patch = image_patch.crop_right_of_bbox(
        >>>         foo_patch.left, foo_patch.lower, foo_patch.right, foo_patch.upper
        >>>     )
        >>>     return bool_to_yesno(right_of_foo_patch.verify_property("bar", "quuxy"))
        """
        pass

    def crop_below_bbox(self, left, lower, right, upper):
        """Returns an ImagePatch object representing the area below the given
        bounding box coordinates.

        Parameters
        ----------
        left, lower, right, upper : int
            The coordinates of the bounding box.

        Returns
        -------
        ImagePatch
            An ImagePatch object representing the cropped area.

        Examples
        --------
        >>> # Is the bar below the foo quuxy?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patch = image_patch.find("foo")[0]
        >>>     below_foo_patch = image_patch.crop_below_bbox(
        >>>         foo_patch.left, foo_patch.lower, foo_patch.right, foo_patch.upper
        >>>     )
        >>>     return bool_to_yesno(below_foo_patch.verify_property("bar", "quuxy"))
        """
        pass

    def crop_above_bbox(self, left, lower, right, upper):
        """Returns an ImagePatch object representing the area above the given
        bounding box coordinates.

        Parameters
        ----------
        left, lower, right, upper : int
            The coordinates of the bounding box.

        Returns
        -------
        ImagePatch
            An ImagePatch object representing the cropped area.

        Examples
        --------
        >>> # Is the bar above the foo quuxy?
        >>> def execute_command(image) -> str:
        >>>     image_patch = ImagePatch(image)
        >>>     foo_patch = image_patch.find("foo")[0]
        >>>     above_foo_patch = image_patch.crop_above_bbox(
        >>>         foo_patch.left, foo_patch.lower, foo_patch.right, foo_patch.upper
        >>>     )
        >>>     return bool_to_yesno(above_foo_patch.verify_property("bar", "quuxy"))
        """
        pass

```

```
def best_image_match(list_patches: List[ImagePatch], content: List[str], return_index=False) ->
    Union[ImagePatch, int]:
    """Returns the patch most likely to contain the content.
    Parameters
    ----------
    list_patches : List[ImagePatch]
    content : List[str]
        the object of interest
    return_index : bool
        if True, returns the index of the patch most likely to contain the object

    Returns
    -------
    int
        Patch most likely to contain the object
    """
    return best_image_match(list_patches, content, return_index)

def bool_to_yesno(bool_answer: bool) -> str:
    return "yes" if bool_answer else "no"

Write a function using Python and the ImagePatch class (above) that could be executed to provide an
    answer to the query.

Consider the following guidelines:
- Use base Python (comparison, sorting) for basic logical operations, left/right/up/down, math, etc.

# Examples of how to use the API
INSERT_CONTEXT_HERE

Query: INSERT_QUERY_HERE
Program:
```

Listing 2. VQA In-Context Examples

```
# Query: Is the vehicle in the top of the image?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    # Assume there's only one vehicle patch.
    vehicle_patch = image_patch.find("vehicle")[0]
    vehicle_in_top_half = vehicle_patch.vertical_center > image_patch.vertical_center
    return bool_to_yesno(vehicle_in_top_half)

# Query: Are there trains or fences in this scene?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    trains = image_patch.find("train")
    fences = image_patch.find("fence")
    has_trains_or_fences = len(trains) > 0 or len(fences) > 0
    return bool_to_yesno(has_trains_or_fences)

# Query: Is the pillow in the top part or in the bottom of the picture?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    pillow_patches = image_patch.find("pillow")
    pillow_patch = pillow_patches[0]
    pillow_in_top_half = pillow_patch.vertical_center > image_patch.vertical_center
    if pillow_in_top_half:
        return "top"
    else:
        return "bottom"

# Query: What color is the curtain that is to the right of the mirror?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    mirror_patches = image_patch.find("mirror")
    mirror_patch = mirror_patches[0]
    right_of_mirror_patch = image_patch.crop_right_of_bbox(
        mirror_patch.left, mirror_patch.lower, mirror_patch.right, mirror_patch.upper
    )
    return right_of_mirror_patch.simple_query("What color is the curtain?")
```

Listing 3. ITM In-Context Examples

```
# Query: Verify image matches text="An airplane is flying in the sky, and birds are flying below it."
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    airplane_patches = image_patch.find("airplane")
    bird_patches = image_patch.find("bird")

    airplane_in_sky = any(
        airplane_patch.vertical_center > image_patch.height * 0.6
        for airplane_patch in airplane_patches
    )

    birds_below_airplane = any(
        bird_patch.upper <= airplane_patch.lower
        for bird_patch in bird_patches for airplane_patch in airplane_patches
    )

    return bool_to_yesno(airplane_in_sky and birds_below_airplane)

# Query: Verify image matches text="The bird is flying above the tree, and a cat is sitting under the
    tree."
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    bird_patches = image_patch.find("bird")
    tree_patches = image_patch.find("tree")
    cat_patches = image_patch.find("cat")

    bird_above_tree = any(
        bird_patch.lower >= tree_patch.upper and
        abs(bird_patch.horizontal_center - tree_patch.horizontal_center) < 50
        for bird_patch in bird_patches for tree_patch in tree_patches
    )

    cat_under_tree = any(
        cat_patch.upper <= tree_patch.lower and
        abs(cat_patch.horizontal_center - tree_patch.horizontal_center) < 50
        for cat_patch in cat_patches for tree_patch in tree_patches
    )

    return bool_to_yesno(bird_above_tree and cat_under_tree)

# Query: Verify image matches text="The apple is on top of the book, and the pen is beside the book."
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    apple_patches = image_patch.find("apple")
    book_patches = image_patch.find("book")
    pen_patches = image_patch.find("pen")

    apple_on_book = any(
        apple_patch.lower >= book_patch.upper and
        book_patch.left <= apple_patch.horizontal_center <= book_patch.right
        for apple_patch in apple_patches for book_patch in book_patches
    )

    pen_beside_book = any(
        abs(pen_patch.horizontal_center - book_patch.horizontal_center) < 50 and
        abs(pen_patch.vertical_center - book_patch.vertical_center) < 100
        for pen_patch in pen_patches for book_patch in book_patches
    )

    return bool_to_yesno(apple_on_book and pen_beside_book)

#Query: Verify image matches text="A man is riding a bicycle, and a dog is running beside him."
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    man_patches = image_patch.find("man")
    bicycle_patches = image_patch.find("bicycle")
    dog_patches = image_patch.find("dog")

    man_on_bicycle = any(
        man_patch.left <= bicycle_patch.right and man_patch.right >= bicycle_patch.left and
        man_patch.lower <= bicycle_patch.upper and man_patch.upper >= bicycle_patch.lower
        for man_patch in man_patches for bicycle_patch in bicycle_patches
    )

    dog_beside_man = any(
        abs(dog_patch.horizontal_center - man_patch.horizontal_center) < 100 and
        abs(dog_patch.vertical_center - man_patch.vertical_center) < 50
        for dog_patch in dog_patches for man_patch in man_patches
    )

    return bool_to_yesno(man_on_bicycle and dog_beside_man)
```

Listing 4. Reprompting with Unit Tests VQA

```
INSERT_IMAGE_PATCH_API

You are provided a Python program that answers a query about an image, with a set of tests with the
    corresponding outputs and exected responses.
Correct the Python program such that it passes the tests.
- Ensure the corrected program is different than the incorrect program provided.

Query: Is there a blue chair in the image?
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    blue_chair = image_patch.find("chair")
    if not blue_chair:
        return "No"
    is_blue = any([chair.verify_property("blue") for chair in blue_chair])
    return "Yes" if is_blue else "No"
Test Cases:
Test A
Image Content: "A room with a red chair"
Ground Truth Answer: "No"
Program Output: "Error: verify_property() missing 1 required positional argument: ’visual_property’"
Test B
Image Content: "A room with a blue chair under the window"
Ground Truth Answer: "Yes"
Program Output: "Error: verify_property() missing 1 required positional argument: ’visual_property’"
Test C
Image Content: "An empty room"
Ground Truth Answer: "No"
Program Output: "No"
Test D
Image Content: "A garden with a blue chair"
Ground Truth Answer: "Yes"
Program Output: "Error: verify_property() missing 1 required positional argument: ’visual_property’"
Test E
Image Content: "A room with several chairs, all red"
Ground Truth Answer: "No"
Program Output: "Error: verify_property() missing 1 required positional argument: ’visual_property’"
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    chair_patches = image_patch.find("chair")
    if not chair_patches:
        return "No"  # No chairs found
    blue_chair_found = any(chair.verify_property("chair", "blue") for chair in chair_patches)
    return "Yes" if blue_chair_found else "No"

Query: "Are there any flowers to the left of the house?"
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    house_patches = image_patch.find("house")
    if not house_patches:
        return "No house found"
    left_of_house_patch = image_patch.crop_left_of_bbox(
        house_patches.left, house_patches.lower, house_patches.right, house_patches.upper
    )  # Incorrect attribute access
    return "Yes" if left_of_house_patch.exists("flower") else "No"
Test Cases:
Test A
Image Content: "An image of a garden without any buildings."
Ground Truth Answer: "No house found"
Program Output: "Error: ’list’ object has no attribute ’left’"
Test B
Image Content: "A house without a garden"
Ground Truth Answer: "No flowers found"
Program Output: "Error: ’list’ object has no attribute ’left’"
Test C
Image Content: "A house with many flowers around"
Ground Truth Answer: "Yes"
Program Output: "Error: ’list’ object has no attribute ’left’"
Test D
Image Content: "A house with flowers only on the right side"
Ground Truth Answer: "No"
Program Output: "Error: ’list’ object has no attribute ’left’"
Test E
Image Content: "An image with flowers but no house"
Ground Truth Answer: "No house found"
Program Output: "Error: ’list’ object has no attribute ’left’"
Corrected Program:
```

```
def execute_command(image):
    image_patch = ImagePatch(image)
    house_patches = image_patch.find("house")
    if not house_patches:
        return "No house found"
    for house_patch in house_patches:
        left_of_house_patch = image_patch.crop_left_of_bbox(
            house_patch.left, house_patch.lower, house_patch.right, house_patch.upper
        )
        flowers_found = left_of_house_patch.find("flower")
        if flowers_found:
            return "Yes"
    return "No"

Query: Who wears a green shirt?
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    people_patches = image_patch.find("person")
    if not people_patches:
        return "No one"
    person_wearing_green_shirt = None
    for index, person_patch in enumerate(people_patches):
        green_patches = person_patch.find("green")
        if green_patches:
            person_wearing_green_shirt = index
            break
    if person_wearing_green_shirt == None:
        return "No one"
    else:
        return people_patches[person_wearing_green_shirt].simple_query("Who is this?")
Test Cases:
Test A
Image Content: "An image of an empty room"
Ground Truth Answer: "No one"
Program Output: "No one"
Test B
Image Content: "A young girl wearing a green dress and a boy wearing a blue shirt."
Ground Truth Answer: "No one"
Program Output: "girl"
Test C
Image Content: "A man wearing a red shirt and a woman wearing a green shirt."
Ground Truth Answer: "woman"
Program Output: "woman"
Test D
Image Content: "A boy wearing a green shirt."
Ground Truth Answer: "boy"
Program Output: "boy"
Test E
Image Content: "Two people wearing green shirts: a man and a woman"
Ground Truth Answer: "man and woman"
Program Output: "man"
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    people_patches = image_patch.find("person")
    if not people_patches:
        return "No people found"
    people_wearing_green_shirts = []
    for index, person_patch in enumerate(people_patches):
        if person_patch.verify_property("clothing", "shirt") and person_patch.verify_property("color",
             "green"):
            people_wearing_green_shirts.append(index)
    if not people_wearing_green_shirts:
        return "No one"
    wearing_green_shirts = ’ and ’.join([people_patches[i].simple_query("Who is this?") for i in
        people_wearing_green_shirts])
    return wearing_green_shirts

Query: "Is the blue ball inside the box?"
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    ball_patches = image_patch.find("ball")
    if not ball_patches:
        return "No"
    blue_ball = [ball for ball in ball_patches if ball.verify_property("color", "blue")]
    if not blue_ball:
        return "No"  # No blue ball found
    box_patches = image_patch.find("box")
    if not box_patches:
        return "No"
    return "Yes"
Test Cases:
Test A

Image Content: "A blue ball is outside a box"
Ground Truth Answer: "No"
Program Output: "Yes"
Test B
Image Content: "A red ball is inside a box"
Ground Truth Answer: "No"
Program Output: "No"
Test C
Image Content: "A blue ball is inside a box"
Ground Truth Answer: "Yes"
Program Output: "Yes"
Test D
Image Content: "No balls or boxes in the image"
Ground Truth Answer: "No"
Program Output: "No"
Test E
Image Content: "Multiple blue balls, all outside boxes"
Ground Truth Answer: "No"
Program Output: "Yes"
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    ball_patches = image_patch.find("ball")
    if not ball_patches:
        return "No"  # No ball found
    blue_ball = [ball for ball in ball_patches if ball.verify_property("color", "blue")]
    if not blue_ball:
        return "No"  # No blue ball found
    box_patches = image_patch.find("box")
    if not box_patches:
        return "No"  # No box found
    blue_ball_patch = blue_ball[0]
    for box_patch in box_patches:
        if (box_patch.left <= blue_ball_patch.left and
            box_patch.right >= blue_ball_patch.right and
            box_patch.upper <= blue_ball_patch.upper and
            box_patch.lower >= blue_ball_patch.lower):
            return "Yes"
    return "No"

Query: INSERT_QUERY_HERE
Incorrect Program:
INSERT_CODE_HERE
Test Cases:
INSERT_UNIT_TEST_OUTPUTS_HERE
Corrected Program:
```

Listing 5. Reprompting with Unit Tests ITM

```
INSERT_IMAGE_PATCH_API

You are provided a Python program that answers a query about an image, with a set of tests with the
    corresponding outputs and exected responses.
Correct the Python program such that it passes the tests.
- Ensure the corrected program is different than the incorrect program provided.

Query: "Verify image matches text="An airplane is flying in the sky, and birds are flying below it.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    airplane = image_patch.find("airplane")
    birds = image_patch.find("birds")
    if not airplane or not birds:
        return "No"
    if airplane[0].vertical_center >= birds[0].vertical_center:
        return "Yes"
    return "No"
Test Cases:
Test A
Image Content: "An airplane flying high in the sky with birds below it."
Ground Truth Answer: "Yes"
Program Output: "Yes"
```

```
Test B
Image Content: "Birds are flying above and below an airplane in the sky."
Ground Truth Answer: "No"
Program Output: "Yes"
Test C
Image Content: "An airplane and birds flying side by side."
Ground Truth Answer: "No"
Program Output: "Yes"
Test D
Image Content: "Only an airplane is flying in the sky."
Ground Truth Answer: "No"
Program Output: "No"
Test E
Image Content: "Birds flying in the sky with no airplane present."
Ground Truth Answer: "No"
Program Output: "No"
Corrected Program::
def execute_command(image):
    image_patch = ImagePatch(image)
    airplane_patches = image_patch.find("airplane")
    bird_patches = image_patch.find("bird")
    if not airplane_patches or not bird_patches:
        return "No"
    airplane = airplane_patches[0]
    birds_below = all(bird.vertical_center > airplane.vertical_center for bird in bird_patches)
    return "Yes" if birds_below else "No"

Query: "Verify image matches text="The bird is flying above the tree, and a cat is sitting under the
    tree.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    tree = image_patch.find("tree")
    bird = image_patch.find("bird")
    cat = image_patch.find("cat")
    if not tree or not bird or not cat:
        return "No"
    if bird[0].vertical_center < tree[0].vertical_center and cat[0].vertical_center >
        tree[0].vertical_center:
        return "Yes"
    return "No"
Test Cases:
Test A
Image Content: "A bird flying above a tree and a cat under the tree."
Ground Truth Answer: "Yes"
Program Output: "Yes"
Test B
Image Content: "A cat sitting above the tree and a bird flying below it."
Ground Truth Answer: "No"
Program Output: "Yes"
Test C
Image Content: "A bird sitting in the tree with no cat around."
Ground Truth Answer: "No"
Program Output: "No"
Test D
Image Content: "A cat climbing the tree while a bird flies overhead."
Ground Truth Answer: "No"
Program Output: "Yes"
Test E
Image Content: "A bird flying above a tree with a dog under the tree."
Ground Truth Answer: "No"
Program Output: "No"
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    tree_patches = image_patch.find("tree")
    bird_patches = image_patch.find("bird")
    cat_patches = image_patch.find("cat")
    if not tree_patches or not bird_patches or not cat_patches:
        return "No"
    tree = tree_patches[0]
    bird_above = all(bird.vertical_center < tree.vertical_center for bird in bird_patches)
    cat_below = all(cat.vertical_center > tree.vertical_center for cat in cat_patches)
    return "Yes" if bird_above and cat_below else "No"

Query: "Verify image matches text="A car is parked near a tree, and a bird is sitting on the tree.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    car = image_patch.find("car")
    tree = image_patch.find("tree")
    bird = image_patch.find("bird")
    if not car or not tree or not bird:
        return "No"
    if car.horizontal_center - tree.horizontal_center < 100 and bird.vertical_center <
        tree.vertical_center:
        return "Yes"
    return "No"
Test Cases:
Test A
Image Content: "A car parked near a tree with a bird sitting on it."
Ground Truth Answer: "Yes"
Program Output: AttributeError: ’list’ object has no attribute ’horizontal_center’
Test B
Image Content: "A car far from a tree with a bird on the ground."
Ground Truth Answer: "No"
Program Output: AttributeError: ’list’ object has no attribute ’horizontal_center’
Test C
Image Content: "A tree with a bird on it but no car nearby."
Ground Truth Answer: "No"
Program Output: "No"
Test D
Image Content: "A car parked near a tree with no bird in sight."
Ground Truth Answer: "No"
Program Output: AttributeError: ’list’ object has no attribute ’horizontal_center’
Test E
Image Content: "A car and a bird but no tree present."
Ground Truth Answer: "No"
Program Output: AttributeError: ’list’ object has no attribute ’horizontal_center’
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    car_patches = image_patch.find("car")
    tree_patches = image_patch.find("tree")
    bird_patches = image_patch.find("bird")
    if not car_patches or not tree_patches or not bird_patches:
        return "No"
    car = car_patches[0]
    tree = tree_patches[0]
    bird = bird_patches[0]
    car_near_tree = abs(car.horizontal_center - tree.horizontal_center) < 100
    bird_on_tree = bird.vertical_center < tree.vertical_center
    return "Yes" if car_near_tree and bird_on_tree else "No"

Query: "Verify image matches text="A man is holding a red balloon, and a child is reaching up to grab
    it.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    man = image_patch.find("man")
    balloon = image_patch.find("balloon")
    child = image_patch.find("child")
    if not man or not balloon or not child:
        return "No"
    if balloon[0].verify_property("red") and child[0].vertical_center < balloon[0].vertical_center:
        return "Yes"
    return "No"
Test Cases:
Test A
Image Content: "A man holding a red balloon, with a child reaching up."
Ground Truth Answer: "Yes"
Program Output: TypeError: verify_property() missing 1 required positional argument: ’visual_property’
Test B
Image Content: "A man holding a blue balloon, with a child below him."
Ground Truth Answer: "No"
Program Output: TypeError: verify_property() missing 1 required positional argument: ’visual_property’
Test C
Image Content: "A man holding a flower, with a child next to him."
Ground Truth Answer: "No"
Program Output: "No"
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    man_patches = image_patch.find("man")
    balloon_patches = image_patch.find("balloon")
    child_patches = image_patch.find("child")
    if not man_patches or not balloon_patches or not child_patches:
        return "No"
    balloon = balloon_patches[0]
    is_red_balloon = balloon.verify_property("balloon", "red")
    child_below_balloon = all(child.vertical_center < balloon.vertical_center for child in
        child_patches)
    return "Yes" if is_red_balloon and child_below_balloon else "No"

Query: "Verify image matches text="A cat is sitting on the table, and a book is lying beside it.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    cat = image_patch.find("cat")
    book = image_patch.find("book")
    if not cat or not book:
        return "No"
    if abs(book[0].horizontal_center - cat[0].horizontal_center) < 50:
        return "Yes"
    return "No"
Test Cases:
Test A
Image Content: "A cat sitting on the table with a book beside it."
Ground Truth Answer: "Yes"
Program Output: "Yes"
Test B
Image Content: "A cat sitting on the floor with a book beside it."
Ground Truth Answer: "No"
Program Output: "Yes"
Test C
Image Content: "A cat sitting on the table with no book around."
Ground Truth Answer: "No"
Program Output: "No"
Test D
Image Content: "A book lying on the table with no cat in sight."
Ground Truth Answer: "No"
Program Output: "No"
Test E
Image Content: "A cat sitting on the table with a book on the floor."
Ground Truth Answer: "No"
Program Output: "Yes"
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    cat_patches = image_patch.find("cat")
    book_patches = image_patch.find("book")
    table_patches = image_patch.find("table")
    if not cat_patches or not book_patches or not table_patches:
        return "No"
    cat = cat_patches[0]
    book = book_patches[0]
    table = table_patches[0]
    is_cat_on_table = cat.vertical_center < table.vertical_center and abs(cat.horizontal_center -
        table.horizontal_center) < 50
    is_book_beside_cat = abs(book.horizontal_center - cat.horizontal_center) < 50
    return "Yes" if is_cat_on_table and is_book_beside_cat else "No"

Query: INSERT_QUERY_HERE
Incorrect Program:
INSERT_CODE_HERE
Test Cases:
INSERT_UNIT_TEST_OUTPUTS_HERE
Corrected Program:
```

Listing 6. VQA Unit Test Generation In Context Examples

```
Query: Is there a cat or dog in the image?
Tests:
1. Image Caption: "A grey tabby cat peacefully napping on a plush sofa" Answer: yes
2. Image Caption: "A lively golden retriever bounding across a grassy field in the park" Answer: yes
3. Image Caption: "Twin Siamese cats playfully swatting at a bright yellow ball" Answer: yes
4. Image Caption: "A cluster of wild horses trotting along the sandy shores of a sunlit beach" Answer:
    no
5. Image Caption: "An orange cat and a black Labrador playfully tugging on a rope toy" Answer: yes
6. Image Caption: "A modern living room featuring sleek furniture and devoid of any pets" Answer: no

Query: Is there a red truck or bus in the image?
Tests:
1. Image Caption: "A vibrant red Ford pickup parked beside a country road" Answer: yes
2. Image Caption: "A red double-decker bus navigating through a busy downtown street" Answer: yes
3. Image Caption: "A large blue semi-truck cruising down an interstate highway" Answer: no
```

```
4. Image Caption: "A quiet suburban street devoid of any large vehicles like buses or trucks" Answer:
    no
5. Image Caption: "A shiny red Ferrari speeding on a professional race track" Answer: no
6. Image Caption: "An array of red delivery trucks lined up in a distribution center parking lot"
    Answer: yes
7. Image Caption: "Several bright yellow school buses parked in a row at a local school" Answer: no

Query: What color is the largest car in the image?
Tests:
1. Image Caption: "A large blue Ford pickup truck driving on a busy highway" Answer: blue
2. Image Caption: "A city street empty of any large vehicles like buses or trucks" Answer: no answer
3. Image Caption: "A row of green food trucks serving lunch in an urban park" Answer: green
4. Image Caption: "A scene with a green public bus next to a smaller blue pickup at an intersection"
    Answer: green

Query: Is the vase to the left or right of the center?
Tests:
1. Image Caption: "A delicate porcelain vase positioned on the right end of a mahogany dining table"
    Answer: right
2. Image Caption: "A tall glass vase sitting on the left side of a neatly made bed in a sunlit room"
    Answer: left
3. Image Caption: "A ceramic vase centrally placed on a round table surrounded by chairs" Answer:
    center

Query: What is the highest object in the image?
Tests:
1. Image Caption: "A massive skyscraper dominating the skyline among lower city buildings" Answer:
    skyscraper
2. Image Caption: "A lone oak tree surpassing the height of the cottage it stands next to" Answer: tree
3. Image Caption: "Colorful balloons drifting above the treetops in a clear sky" Answer: balloons
4. Image Caption: "A commercial jet flying high above the city's tallest skyscrapers" Answer: plane
5. Image Caption: "A majestic eagle soaring high above a vast canyon landscape" Answer: eagle
6. Image Caption: "A figure standing on the peak of a grassy hill under a blue sky" Answer: person

Query: INSERT_QUERY_HERE
Tests:
```

Listing 7. ITM Unit Test Generation In Context Examples

```
Query: Is the drawing of a tree on the hill, and a river that flows at the bottom of the hill?
Tests:
1. Image Caption: "A solitary tree stands atop a gentle hill, with a flowing river winding below it."
    Answer: yes
2. Image Caption: "A tree on a grassy hill under a clear sky." Answer: no
3. Image Caption: "A river meandering through a dense forest of tall trees." Answer: no
4. Image Caption: "A panoramic view of rolling hills in the desert, with a river at the bottom."
    Answer: no
5. Image Caption: "A vast plain with a river running through fields of wildflowers." Answer: no
6. Image Caption: Image Caption: "A hill with multiple trees and a river flowing nearby." Answer: yes

Query: Is the drawing of an airplane flying in the sky, and birds flying below it?
Tests:
1. Image Caption:  "An airplane soars through the sky, with a flock of birds flying beneath it."
    Answer: yes
2. Image Caption: "Birds flying over a tranquil lake under a clear sky." Answer: no
3. Image Caption: "An airplane performing aerobatic maneuvers, with birds flying above it." Answer: no
4. Image Caption: "An airplane floating in the sea with birds flying above it." Answer: Yes
5. Image Caption: "An airplane in a clear sky" Answer: no

Query: Is the drawing of a girl holding an umbrella in the rain?
Tests:
1. Image Caption: "A girl holding an umbrella walks through a rainy street." Answer: yes
2. Image Caption: "A girl holds an umbrella under a bright sun in the park." Answer: no
3. Image Caption: "A girl stands in the rain wearing a colorful raincoat and holding flowers." Answer:
    no
4. Image Caption: "A girl walks her dog while holding an umbrella on a rainy day." Answer: yes

Query: Is the drawing of a person sitting at a desk with a computer monitor in front of them?
Tests:
1. Image Caption: "A person sitting at a desk, writing in a notebook with a lamp beside them." Answer:
    no
2. Image Caption: "Someone sitting at a desk cluttered with papers and a computer monitor." Answer: yes
3. Image Caption: "Someone sitting at a desk cluttered with papers and a computer monitor." Answer: yes
3. Image Caption: "A person with a big computer screen in the background" Answer: no


Query: Is the drawing of a man riding a bicycle, and a dog running beside him?
Tests:
1. Image Caption: "A man cycling alone on a mountain trail surrounded by trees." Answer: no
```

```
2. Image Caption: "A man rides a bicycle along the beach, his dog running beside him." Answer: yes
3. Image Caption: "A bicycle and a dog" Answer: no
4. Image Caption: "A dog next to a car" Answer: no
5. Image Caption: "A man walking his dog" Answer: no
6. Image Caption: "A man rides a bicycle down a sunny street with a dog running beside him." Answer:
    yes

Query: INSERT_QUERY_HERE
Tests:
```

Listing 8. VQA Unit Test Generation with Implementation In-Context Examples

```
# Query: Is there a cat or dog in the image?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    cats = image_patch.find("cat")
    dogs = image_patch.find("dog")
    has_cats_or_dogs = len(cats) > 0 or len(dogs) > 0
    return bool_to_yesno(has_cats_or_dogs)
Tests:
1. Image Caption: "A grey tabby cat peacefully napping on a plush sofa" Answer: yes
2. Image Caption: "A lively golden retriever bounding across a grassy field in the park" Answer: yes
3. Image Caption: "Twin Siamese cats playfully swatting at a bright yellow ball" Answer: yes
4. Image Caption: "A cluster of wild horses trotting along the sandy shores of a sunlit beach" Answer:
    no
5. Image Caption: "An orange cat and a black Labrador playfully tugging on a rope toy" Answer: yes
6. Image Caption: "A modern living room featuring sleek furniture and devoid of any pets" Answer: no

# Query: Is there a red truck or bus in the image?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    trucks = image_patch.find("truck")
    buses = image_patch.find("bus")
    red_trucks = [truck for truck in trucks if truck.verify_property("truck", "red")]
    red_buses = [bus for bus in buses if bus.verify_property("bus", "red")]
    has_red_trucks_or_buses = len(red_trucks) > 0 or len(red_buses) > 0
    return bool_to_yesno(has_red_trucks_or_buses)
Tests:
1. Image Caption: "A vibrant red Ford pickup parked beside a country road" Answer: yes
2. Image Caption: "A red double-decker bus navigating through a busy downtown street" Answer: yes
3. Image Caption: "A large blue semi-truck cruising down an interstate highway" Answer: no
4. Image Caption: "A quiet suburban street devoid of any large vehicles like buses or trucks" Answer:
    no
5. Image Caption: "A shiny red Ferrari speeding on a professional race track" Answer: no
6. Image Caption: "An array of red delivery trucks lined up in a distribution center parking lot"
    Answer: yes
7. Image Caption: "Several bright yellow school buses parked in a row at a local school" Answer: no


# Query: What color is the largest car in the image?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    car_patches = image_patch.find("car")
    if not car_patches:
        return "No cars found in the image."
    # Sort cars by their area to find the largest one
    car_patches.sort(key=lambda x: x.area, reverse=True)
    largest_car_patch = car_patches[0]
    color_of_largest_car = largest_car_patch.simple_query("What is the color?")
    return color_of_largest_car
Tests:
1. Image Caption: "A large blue Ford pickup truck driving on a busy highway" Answer: blue
2. Image Caption: "A city street empty of any large vehicles like buses or trucks" Answer: no answer
3. Image Caption: "A row of green food trucks serving lunch in an urban park" Answer: green
4. Image Caption: "A scene with a green public bus next to a smaller blue pickup at an intersection"
    Answer: green

# Query: Is the vase to the left or right of the center?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    vase_patches = image_patch.find("vase")
    if not vase_patches:
        return "No vases found in the image."
    vase_patch = vase_patches[0]
    vase_position = vase_patch.horizontal_center
    image_center = (image_patch.left + image_patch.right) / 2
    if vase_position < image_center:
        return "left"
    elif vase_position > image_center:
        return "right"
    else:
        return "center"
Tests:
1. Image Caption: "A delicate porcelain vase positioned on the right end of a mahogany dining table"
    Answer: right
2. Image Caption: "A tall glass vase sitting on the left side of a neatly made bed in a sunlit room"
    Answer: left
3. Image Caption: "A ceramic vase centrally placed on a round table surrounded by chairs" Answer:
    center

# Query: What is the highest object in the image?
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    possible_objects = ["car", "tree", "building", "person", "vase", "animal", "vehicle", "furniture"]
    all_patches = []
    for obj in possible_objects:
        all_patches.extend(image_patch.find(obj))
    if not all_patches:
        return "No objects found in the image."
    highest_patch = max(all_patches, key=lambda x: x.upper)
    highest_object_name = highest_patch.simple_query("What is this?")
    return highest_object_name
Tests:
1. Image Caption: "A massive skyscraper dominating the skyline among lower city buildings" Answer:
    skyscraper
2. Image Caption: "A lone oak tree surpassing the height of the cottage it stands next to" Answer: tree
3. Image Caption: "Colorful balloons drifting above the treetops in a clear sky" Answer: balloons
4. Image Caption: "A commercial jet flying high above the city’s tallest skyscrapers" Answer: plane
5. Image Caption: "A majestic eagle soaring high above a vast canyon landscape" Answer: eagle
6. Image Caption: "A figure standing on the peak of a grassy hill under a blue sky" Answer: person

Create test cases for the specified query and program using the format provided in the examples.
The test cases should consist of image captions and answers to the query.
The answers should be consice, limited to a single word.

Query: INSERT_QUERY_HERE
Program:
INSERT_PROGRAM_HERE
Tests:
```

Listing 9. Example Code

```
I will provide you with a caption for a photo, image, or painting.
Your task is to generate the bounding boxes for the objects mentioned in the caption, along with a
    background prompt describing the scene.
The images are of size 512x512. The top-left corner has coordinate [0, 0].
The bottom-right corner has coordinnate [512, 512].
The bounding boxes should not overlap or go beyond the image boundaries.
Each bounding box should be in the format of (object name, [top-left x coordinate, top-left y
    coordinate, box width, box height]) and should not include more than one object.
Do not put objects that are already provided in the bounding boxes into the background prompt. Do not
    include non-existing or excluded objects in the background prompt.
Use "A realistic scene" as the background prompt if no background is given in the prompt. If needed,
    you can make reasonable guesses.
Please refer to the example below for the desired format.

Caption: A realistic image of landscape scene depicting a green car parking on the left of a blue
    truck, with a red air balloon and a bird in the sky
Objects: [(’a green car’, [21, 281, 211, 159]), (’a blue truck’, [269, 283, 209, 160]), (’a red air
    balloon’, [66, 8, 145, 135]), (’a bird’, [296, 42, 143, 100])]
Background prompt: A realistic landscape scene
Negative prompt: None

Caption: A realistic top-down view of a wooden table with two apples on it
Objects: [(’a wooden table’, [20, 148, 472, 216]), (’an apple’, [150, 226, 100, 100]), (’an apple’,
    [280, 226, 100, 100])]
Background prompt: A realistic top-down view
Negative prompt: None

Caption: A realistic scene of three skiers standing in a line on the snow near a palm tree
Objects: [(’a skier’, [5, 152, 139, 168]), (’a skier’, [278, 192, 121, 158]), (’a skier’, [148, 173,
    124, 155]), (’a palm tree’, [404, 105, 103, 251])]
Background prompt: A realistic outdoor scene with snow
Negative prompt: None

Caption: An oil painting of a pink dolphin jumping on the left of a steam boat on the sea
Objects: [(’a steam boat’, [232, 225, 257, 149]), (’a jumping pink dolphin’, [21, 249, 189, 123])]
Background prompt: An oil painting of the sea
```

```
Negative prompt: None

Caption: A cute cat and an angry dog without birds
Objects: [('a cute cat', [51, 67, 271, 324]), ('an angry dog', [302, 119, 211, 228])]
Background prompt: A realistic scene
Negative prompt: birds

Caption: Two pandas in a forest without flowers
Objects: [('a panda', [30, 171, 212, 226]), ('a panda', [264, 173, 222, 221])]
Background prompt: A forest
Negative prompt: flowers

Caption: An oil painting of a living room scene without chairs with a painting mounted on the wall, a
    cabinet below the painting, and two flower vases on the cabinet
Objects: [('a painting', [88, 85, 335, 203]), ('a cabinet', [57, 308, 404, 201]), ('a flower vase',
    [166, 222, 92, 108]), ('a flower vase', [328, 222, 92, 108])]
Background prompt: An oil painting of a living room scene
Negative prompt: chairs

Caption: INSERT_PROMPT_HERE
Objects:
```

Listing 10. Reprompting with Errors VQA

```
INSERT_IMAGE_PATCH_API
You are provided a Python program that answers a query about an image, with a set of tests with the
    corresponding outputs and exected responses.
Correct the Python program such that it passes the tests.
- Ensure the corrected program is different than the incorrect program provided.

Query: Is there a blue chair in the image?
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    blue_chair = image_patch.find("chair")
    if not blue_chair:
        return "No"
    is_blue = any([chair.verify_property("blue") for chair in blue_chair])
    return "Yes" if is_blue else "No"
Error: verify_property() missing 1 required positional argument: 'visual_property
Corrected Program::
def execute_command(image):
    image_patch = ImagePatch(image)
    chair_patches = image_patch.find("chair")
    if not chair_patches:
        return "No"  # No chairs found
    blue_chair_found = any(chair.verify_property("chair", "blue") for chair in chair_patches)
    return "Yes" if blue_chair_found else "No"

Query: "Are there any flowers to the left of the house?"
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    house_patches = image_patch.find("house")
    left_of_house_patch = image_patch.crop_left_of_bbox(
        house_patches.left, house_patches.lower, house_patches.right, house_patches.upper
    )  # Incorrect attribute access
    return "Yes" if left_of_house_patch.exists("flower") else "No"
Error: 'list' object has no attribute 'left
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    house_patches = image_patch.find("house")
    if not house_patches:
        return "No house found"
    house_patch = house_patches[0]
    left_of_house_patch = image_patch.crop_left_of_bbox(
        house_patch.left, house_patch.lower, house_patch.right, house_patch.upper
    )
    flowers_found = left_of_house_patch.find("flower")
    return "Yes" if flowers_found else "No"


Query: Who wears a green shirt?
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    # Incorrectly calling find() with an extra argument, leading to an error
    people_patches = image_patch.find("person", "green")
    if not people_patches:
        return "No one"
    people_wearing_green_shirts = []
    for person_patch in people_patches:
        if person_patch.verify_property("clothing", "shirt") and person_patch.verify_property("color",
            "green"):
            people_wearing_green_shirts.append(person_patch)
    if not people_wearing_green_shirts:
        return "No one"
    wearing_green_shirts = ’, ’.join([person.simple_query("Who is this?") for person in
        people_wearing_green_shirts])
    return wearing_green_shirts
Error: find() takes 2 positional arguments but 3 were given
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    people_patches = image_patch.find("person")
    if not people_patches:
        return "No people found"
    people_wearing_green_shirts = []
    for index, person_patch in enumerate(people_patches):
        if person_patch.verify_property("clothing", "shirt") and person_patch.verify_property("color",
            "green"):
            people_wearing_green_shirts.append(index)
    if not people_wearing_green_shirts:
        return "No one"
    wearing_green_shirts = ’, ’.join([people_patches[i].simple_query("Who is this?") for i in
        people_wearing_green_shirts])
    return wearing_green_shirts

Query: "Is the blue ball inside the box?"
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    ball_patches = image_patch.find("ball")
    blue_ball = [ball for ball in ball_patches if ball.verify_property("color", "blue")]
    blue_ball_left = blue_ball[0].left
    box_patches = image_patch.find("box")
    box_left = box_patches[0].left  # Assuming there’s always a box present
    if not box_patches:
        return "No"
    return "Yes"
Error: IndexError: list index out of range
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    ball_patches = image_patch.find("ball")
    if not ball_patches:
        return "No"  # No ball found
    blue_ball = [ball for ball in ball_patches if ball.verify_property("color", "blue")]
    if not blue_ball:
        return "No"  # No blue ball found
    box_patches = image_patch.find("box")
    if not box_patches:
        return "No"  # No box found
    blue_ball_patch = blue_ball[0]
    for box_patch in box_patches:
        if (box_patch.left <= blue_ball_patch.left and
            box_patch.right >= blue_ball_patch.right and
            box_patch.upper <= blue_ball_patch.upper and
            box_patch.lower >= blue_ball_patch.lower):
            return "Yes"
    return "No"

Query: "Is the table bigger than the chair?"
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    table_patches = image_patch.find("table")
    chair_patches = image_patch.find("chair")
    if not table_patches or not chair_patches:
        return "No"
    if table_patch.area < chair_patch.area:
        return "Yes"
    return "No"
Error: name ’table_patch’ is not defined
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    table_patches = image_patch.find("table")
    chair_patches = image_patch.find("chair")
    if not table_patches or not chair_patches:
        return "No"
    table_patch = table_patches[0]
    chair_patch = chair_patches[0]
    if table_patch.area > chair_patch.area:
        return "Yes"
    return "No"

Query: "What is the color of the largest ball?"
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    ball_patches = image_patch.find("ball")[0]
    ball_patches.sort(key=lambda x: x.area)
    largest_ball = ball_patches[-1]  # Picks the smallest ball due to incorrect indexing
    return largest_ball.simple_query("What is the color?")
Error: ’ImagePatch’ object has no attribute ’sort’
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    ball_patches = image_patch.find("ball")
    ball_patches.sort(key=lambda x: x.area)
    largest_ball = ball_patches[-1]
    return largest_ball.simple_query("What is the color?")

Query: INSERT_QUERY_HERE
Incorrect Program:
INSERT_CODE_HERE
Error: INSERT_ERROR_HERE
Corrected Program:
```

Listing 11. Reprompting with Errors ITM

```
INSERT_IMAGE_PATCH_API

You are provided a Python program that answers a query about an image, with a set of tests with the
    corresponding outputs and exected responses.
Correct the Python program such that it passes the tests.
- Ensure the corrected program is different than the incorrect program provided.

Query: "Verify image matches text="An airplane is flying in the sky, and birds are flying below it.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    airplane = image_patch.find("airplane")
    birds = image_patch.find("birds")
    if airplane[0].vertical_center > birds[0].vertical_center:
        return "Yes"
    return "No"
Error: IndexError: list index out of range
Corrected Program::
def execute_command(image):
    image_patch = ImagePatch(image)
    airplane_patches = image_patch.find("airplane")
    bird_patches = image_patch.find("bird")
    if not airplane_patches or not bird_patches:
        return "No"
    airplane = airplane_patches[0]
    birds_below = all(bird.vertical_center > airplane.vertical_center for bird in bird_patches)
    return "Yes" if birds_below else "No"

Query: "Verify image matches text="The bird is flying above the tree, and a cat is sitting under the
    tree.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    tree = image_patch.find("tree")
    bird = image_patch.find("bird")
    cat = image_patch.find("cat")
    if not tree or not bird or not cat:
        return "No"
    if bird.vertical_center < tree.vertical_center and cat.vertical_center > tree.vertical_center:
        return "Yes"
    return "No"
Error: list has no attribute vertical_center
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    tree_patches = image_patch.find("tree")
    bird_patches = image_patch.find("bird")
    cat_patches = image_patch.find("cat")
    if not tree_patches or not bird_patches or not cat_patches:
        return "No"
    tree = tree_patches[0]
    bird_above = all(bird.vertical_center < tree.vertical_center for bird in bird_patches)
    cat_below = all(cat.vertical_center > tree.vertical_center for cat in cat_patches)
    return "Yes" if bird_above and cat_below else "No"

Query: "Verify image matches text="A man is riding a bicycle, and a dog is running beside him.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    man = image_patch.find("man")
    bicycle = image_patch.find("bicycle")
    dog = image_patch.find("dog")
    if not man or not bicycle or not dog:
        return "No"
    if abs(man[0].center_x - dog[0].center_x) < 50:
        return "Yes"
    return "No"
Error: ImagePatch has no attribute center_x
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    man_patches = image_patch.find("man")
    bicycle_patches = image_patch.find("bicycle")
    dog_patches = image_patch.find("dog")
    if not man_patches or not bicycle_patches or not dog_patches:
        return "No"
    man = man_patches[0]
    bicycle = bicycle_patches[0]
    dog_beside = any(abs(dog.horizontal_center - man.horizontal_center) < 100 for dog in dog_patches)
    return "Yes" if dog_beside else "No"

Query: "Verify image matches text="A man is holding a red balloon, and a child is reaching up to grab
    it.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    man = image_patch.find("man")
    balloon = image_patch.find("balloon")
    child = image_patch.find("child")
    if not man or not balloon or not child:
        return "No"
    if balloon[0].verify_property("red") and child[0].vertical_center < balloon[0].vertical_center:
        return "Yes"
    return "No"
Error: verify_property() missing 1 required positional argument: ’visual_property’
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    man_patches = image_patch.find("man")
    balloon_patches = image_patch.find("balloon")
    child_patches = image_patch.find("child")
    if not man_patches or not balloon_patches or not child_patches:
        return "No"
    balloon = balloon_patches[0]
    is_red_balloon = balloon.verify_property("balloon", "red")
    child_below_balloon = all(child.vertical_center < balloon.vertical_center for child in
        child_patches)
    return "Yes" if is_red_balloon and child_below_balloon else "No"

Query: "Verify image matches text="A cat is sitting on the table, and a book is lying beside it.""
Incorrect Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    cat_patches = image_patch.find("cat")
    book_patches = image_patch.find("book")
    if not cat_patches or not book_patches:
        return "No"
    if abs(cat.horizontal_center - book.horizontal_center) < 50:
        return "Yes"
    return "No"
Error: name ’cat’ is not defined
Corrected Program:
def execute_command(image):
    image_patch = ImagePatch(image)
    cat_patches = image_patch.find("cat")
    book_patches = image_patch.find("book")
    table_patches = image_patch.find("table")
    if not cat_patches or not book_patches or not table_patches:
        return "No"
    cat = cat_patches[0]
    book = book_patches[0]
    table = table_patches[0]
    is_cat_on_table = cat.vertical_center < table.vertical_center and abs(cat.horizontal_center -
        table.horizontal_center) < 50
    is_book_beside_cat = abs(book.horizontal_center - cat.horizontal_center) < 50
    return "Yes" if is_cat_on_table and is_book_beside_cat else "No"

Query: INSERT_QUERY_HERE
Incorrect Program:
INSERT_CODE_HERE

Error: INSERT_ERROR_HERE
```

References

[1] Yash Goyal, Tejas Khot, Douglas Summers-Stay, Dhruv Batra, and Devi Parikh. Making the v in vqa matter: Elevating the role of image understanding in visual question answering. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 6904–6913, 2017. 6

[2] Cheng-Yu Hsieh, Jieyu Zhang, Zixian Ma, Aniruddha Kembhavi, and Ranjay Krishna. Sugarcrepe: Fixing hackable benchmarks for vision-language compositionality. *Advances in neural information processing systems*, 36, 2024. 1

[3] Drew A Hudson and Christopher D Manning. Gqa: A new dataset for real-world visual reasoning and compositional question answering. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 6700–6709, 2019. 1

[4] Zaid Khan, Vijay Kumar BG, Samuel Schulter, Yun Fu, and Manmohan Chandraker. Self-training large language models for improved visual program synthesis with visual reinforcement. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 14344–14353, 2024. 1, 6

[5] Junnan Li, Dongxu Li, Silvio Savarese, and Steven Hoi. Blip-2: Bootstrapping language-image pre-training with frozen image encoders and large language models. In *International conference on machine learning*, pages 19730–19742. PMLR, 2023. 1, 6

[6] 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. In *European Conference on Computer Vision*. Springer, 2024. 1

[7] Kenneth Marino, Mohammad Rastegari, Ali Farhadi, and Roozbeh Mottaghi. Ok-vqa: A visual question answering benchmark requiring external knowledge. In *Proceedings of the IEEE/cvf conference on computer vision and pattern recognition*, pages 3195–3204, 2019. 6

[8] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. Learning transferable visual models from natural language supervision. In *International conference on machine learning*, pages 8748–8763. PMLR, 2021. 1, 6

[9] Ramprasaath R Selvaraju, Purva Tendulkar, Devi Parikh, Eric Horvitz, Marco Tulio Ribeiro, Besmira Nushi, and Ece Kamar. Squinting at vqa models: Introspecting vqa models with sub-questions. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 10003–10011, 2020. 6

[10] 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. 1

[11] Tristan Thrush, Ryan Jiang, Max Bartolo, Amanpreet Singh, Adina Williams, Douwe Kiela, and Candace Ross. Winoground: Probing vision and language models for visio-linguistic compositionality. In *2022 IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*, pages 5228–5238. IEEE Computer Society, 2022. 1