Self-Training Large Language Models for Improved Visual Program Synthesis With Visual Reinforcement

Supplementary Material
In the appendix, we provide implementation details in Sec. 1, a failure analysis in Sec. 2, more qualitative examples in Sec. 3, and prompts in Sec. 4.

1. Implementation Details
We use ViperGPT [?] as our “backbone”. We follow their implementation of the ImagePatch API almost exactly. We remove some modules and functions that were not necessary for the tasks we explore (e.g. llm_query) is not necessary for our test datasets.

1.1. Grow Step
During the Grow step, we use nucleus sampling to stochastically sample programs from the language model. We prompt the language model with the ImagePatch API description in Sec. 4. In the Huggingface library, this corresponds to the following configuration. We use a top_p value of 0.9, which allows the model to consider the most probable tokens that cumulatively make up 90% of the probability mass. We set top_k was set to 0, disabling the top-k filtering and relying solely on nucleus sampling. The temperature parameter was set to 0.7. Temperature effects the randomness of token selection, with values lower than 1 resulting in less random selections. We increased the max_new_tokens from 180 to 320 to accommodate longer outputs, addressing the issue of premature truncation in programmatic responses. Because the codeLLama-7b model did not include a <PAD> token, we re-use the <EOS> token as the pad token.

1.2. Improve Step
During each Improve step, we train the language model using LoRA [?] for a single epoch. Following [?], we apply LoRA to all fully-connected layers in CodeLlama. In the HuggingFace Transformers library, this corresponds to fc1, fc2, k_proj, v_proj, q_proj, out_proj in each transformer block. This corresponds to the MLP blocks and the QKV matrices in the transformer. We use a LoRA rank of 16, set α = 32, and set the LoRA dropout to 0.05. During training, we use a batch size of 4 and the AdamW [?] optimizer. We use an initial learning rate of 0.0002 and apply a linear learning rate scheduler with a warmup ratio of 0.1.

During training, we use the following instruction-following template for language modeling:

```
<s>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.

Query: <QUERY GOES HERE>
Program: <PROGRAM GOES HERE>
</s>
```

Note that the first half of the instruction following template (up to Program:) is identical to the end of the prompt used during the Grow step (Sec. 4). We only apply the language modeling loss to the tokens of the program, rather than the “instruction”.

1.3. Evaluation Step
Hyperparameters and prompts are identical to the Grow step. Only the datasets change. We use the same prompt (Sec. 4), the same set of in-context examples, and the same hyperparameters.

2. Failure Analysis
2.1. Why does accuracy decrease on some question types?
In ??, we show that self-training allows the language model to improve on almost all question types. What is happening on question types that the language model does not improve on? In Tab. 1, we list those problematic question types and examples of questions from each of the problematic question types. Almost all of them tend to have boolean answers or provide a choice between several categories. To understand why self-training can fail on these questions, consider the scenario of a dataset of entirely boolean questions with possible answers \{yes, no\} where each answer occurs with equal probability. Now consider a
Table 1. Examples of question types from ?? which suffer reduced accuracy after self-training. Almost all of them are either boolean, or require choosing between several categories. In such cases, self-training can reward incorrect reasoning.

2.2. Failure Modes

2.2.1 Incoherent Reasoning

In Fig. 1, we show an example of a severe failure mode. This failure mode occurs with all evaluated LLMs, including gpt-3.5-turbo. First, the LLM makes an unjustified assumption, checking to see if the color of the tag is red. Second, it compares the .category attribute of the tag to the string “bed”. This comparison is irrelevant to the question. Surprisingly, this failure mode occurs even though the LLM is capable of answering other questions of the same question type which require similar reasoning. We hypothesize that in situations where the LLM generates completely incoherent reasoning but is able to answer similar questions correctly, further iterations of reinforced self-training will gradually erase this failure mode. The LLM already “knows” how to synthesize the correct program, but needs additional reinforcement. In situations where the LLM generates completely incoherent reasoning and is not able to answer similar questions correctly, we hypothesize that further

<table>
<thead>
<tr>
<th>Question Type</th>
<th>Answer Type</th>
<th>Example</th>
</tr>
</thead>
<tbody>
<tr>
<td>stateChoose</td>
<td>Categorical</td>
<td>How is the water today, still or wavy?</td>
</tr>
<tr>
<td>twoDifferent</td>
<td>Boolean</td>
<td>Is the vest different in color than the seat?</td>
</tr>
<tr>
<td>existAttrNotC</td>
<td>Boolean</td>
<td>Is there a truck in the scene that is not green?</td>
</tr>
<tr>
<td>diffAnimals</td>
<td>Boolean</td>
<td>Are these animals of different types?</td>
</tr>
<tr>
<td>sameGender</td>
<td>Boolean</td>
<td>Are both the people of the same gender?</td>
</tr>
<tr>
<td>existThatNotC</td>
<td>Boolean</td>
<td>Is there a bird in the picture that is not walking?</td>
</tr>
<tr>
<td>positionVerifyC</td>
<td>Boolean</td>
<td>Is the man on the right of the picture?</td>
</tr>
<tr>
<td>verifyAttrC</td>
<td>Boolean</td>
<td>Is the towel blue and rectangular?</td>
</tr>
<tr>
<td>existC</td>
<td>Boolean</td>
<td>Are there sheep in this picture?</td>
</tr>
<tr>
<td>existMaterialC</td>
<td>Boolean</td>
<td>Is there a bottle that is made of glass?</td>
</tr>
<tr>
<td>relO</td>
<td>Open-Ended</td>
<td>The horse is where?</td>
</tr>
<tr>
<td>twoCommon</td>
<td>Boolean</td>
<td>What do both the shoes and the shorts have in common?</td>
</tr>
<tr>
<td>existAttrNot</td>
<td>Boolean</td>
<td>Is there a fire hydrant in the picture that is not white?</td>
</tr>
<tr>
<td>exist</td>
<td>Boolean</td>
<td>Are there tomatoes?</td>
</tr>
<tr>
<td>sameAnimals</td>
<td>Boolean</td>
<td>Are the animals sheep?</td>
</tr>
<tr>
<td>materialChoose</td>
<td>Categorical</td>
<td>What makes up the lid, plastic or stainless steel?</td>
</tr>
</tbody>
</table>
iterations of reinforced self-training will not erase this failure mode. One solution in this case is to provide human-written examples of correct reasoning. As we show in ??, this stabilizes the self-training process.

### 2.2.2 Unreliable Perception

![Figure 2. An example of a failure mode in which a perception module is unreliable on a simple input.](image)

What type of animal is this, a squirrel or a dog?

```python
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    squirrel_patches = image_patch.find("squirrel")
    if len(squirrel_patches) > 0:
        return "squirrel"
    else:
        return "dog"
```

Result: squirrel

Label: dog

Another type of failure mode is one in which the perception modules are unreliable, as shown in Fig. 2. In the case of Fig. 2, the failure occurs in the find method, which uses GroundingDino as an open vocabulary object detector. The LLM depends on the find method to return an empty list when “squirrel” is not present. However, the object detector spuriously identifies the dog as a squirrel.

### 2.2.3 Complex Relationships

![Figure 3. Verifying / detecting complex relationships is challenging for the program synthesis paradigm.](image)

"the longer row of meatballs"

```python
def execute_command(image):
    image_patch = ImagePatch(image)
    meat_ball_patches = image_patch.find("meat ball")
    verification_question = "Is this a longer row of meatballs?"
    verified_patches = [
        patch for patch in meat_ball_patches
        if simple_query(verification_question) == "yes"
    ]
    # There could be multiple rows of meatballs here, # though the question is only asking for one. We pick a random # one and return it.
    return verified_patches[0]
```

Ground Truth

Ground Truth

Another failure mode is one in which the LLM must verify or detect a complex relationship that cannot be handled by the perception modules. As an example, consider the query in Fig. 3: “the longer row of meatballs”. Recovering the row structure of the meatballs from the detections is not straightforward. More generally, without a strong visual prior, it is difficult for the LLM to construct a programmatic heuristic for complex relationships.

### 3. Qualitative Examples

In Figs. 4 and 5, we show examples of visual questions taken from the GQA validation set in which gpt-3.5-turbo (ViperGPT) incorrectly answers queries, but CodeLlama-7B+ does not. In Figs. 6 and 7, we show examples in which a state-of-the-art open vocabulary object detector (GroundingDino) is not able to localize described objects, but CodeLlama-7B+ succeeds.
Figure 4. Qualitative examples on VQA (GQA) showing errors made by gpt-3.5-turbo (ViperGPT) that are fixed by CodeLlama-7B + VisReP.
Figure 5. Qualitative examples on VQA (GQA) showing errors made by gpt-3.5-turbo (ViperGPT) that are fixed by CodeLlama-7B + VisReP
All the pigeons that aren't fully white

The balloons without glasses

The closer cookie

Figure 6. Qualitative examples on object detection (Omnilabel) showing errors made by a state-of-the-art detector (GroundingDino) that are fixed by CodeLlama+.
Figure 7. Qualitative examples on object detection (Omnilabel) showing errors made by a state-of-the-art detector (GroundingDino) that are fixed by CodeLlama+. 
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."
        self.image = image
        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.crop((self.left, self.upper, self.right, self.lower))
        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)
```python
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"

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"

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 visualize(self):
    ""
    Visualizes the bounding box on the original image and annotates it with the
category name if provided.
    ""
    pass

def crop_left_of_bbox(self, left, upper, right, lower):
    ""
    Returns an ImagePatch object representing the area to the left of the given
Parameters
----------
left, upper, right, lower : 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.upper, foo_patch.right, foo_patch.lower
>>>     )
>>>     return bool_to_yesno(left_of_foo_patch.verify_property("bar", "quuxy"))
""
    pass

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

Parameters
----------
left, upper, right, lower : 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.upper, foo_patch.right, foo_patch.lower
>>>     )
>>>     return bool_to_yesno(right_of_foo_patch.verify_property("bar", "quuxy"))
""
    pass

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

Parameters
----------
left, upper, right, lower : 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.upper, foo_patch.right, foo_patch.lower  
>>>    )
>>>    return bool_to_yesno(below_foo_patch.verify_property("bar", "quuxy"))"
    pass

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

Parameters
----------
left, upper, right, lower : 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.upper, foo_patch.right, foo_patch.lower  
>>>    )
>>>    return bool_to_yesno(above_foo_patch.verify_property("bar", "quuxy"))"
    pass

def bool_to_yesno(bool_answer: bool) -> str:
    pass

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.
INSERT_IN_CONTEXT_EXAMPLES_HERE
Query: INSERT_QUERY_HERE