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 $\alpha = 32$, 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

Question Type	Answer Type	Example
stateChoose	Categorical Choose	How is the water today, still or wavy?
twoDifferent	Boolean	Is the vest different in color than the seat?
existAttrNotC	Boolean	Is there a truck in the scene that is not green?
diffAnimals	Boolean	Are these animals of different types?
sameGender	Boolean	Are both the people of the same gender?
existThatNotC	Boolean	Is there a bird in the picture that is not walking?
positionVerifyC	Boolean	Is the man on the right of the picture?
verifyAttrsC	Boolean	Is the towel blue and rectangular?
existC	Boolean	Are there sheep in this picture?
existMaterialC	Boolean	Is there a bottle that is made of glass?
relO	Open-Ended	The horse is where?
twoCommon	Boolean	What do both the shoes and the shorts have in common?
existAttrNot	Boolean	Is there a fire hydrant in the picture that is not white?
exist	Boolean	Are there tomatoes?
sameAnimals	Boolean	Are the animals sheep?
materialChoose	Categorical Choose	What makes up the lid, plastic or stainless steel?

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.

language model policy π_θ that synthesizes programs that result in *yes* half the time, and programs that result in *no* half the time. In such a case, the policy will receive a non-zero reward approximately 25% of the time, regardless of whether the reasoning in the program was correct or not. This can reinforce incorrect patterns of reasoning.

2.2. Failure Modes

2.2.1 Incoherent Reasoning

Is the color of the tag different from the color of bed?

unjustified assumption

```
def execute_command(image) -> str:
    image_patch = ImagePatch(image)
    tags = image_patch.find("tag")
    for tag_patch in tags:
        if tag_patch.verify_property("tag", "red"):
            return bool_to_yesno(tag_patch.category != "bed")
    return bool_to_yesno(False)
```

irrelevant reasoning

Label: yes

Result: no

Figure 1. An example of a failure mode in which the LLM employs a line of reasoning which is completely incorrect, using both unjustified assumptions and reasoning irrelevant to the question. This was produced by CodeLlama-7b, but similar errors occur with all LLMs tested.

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 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

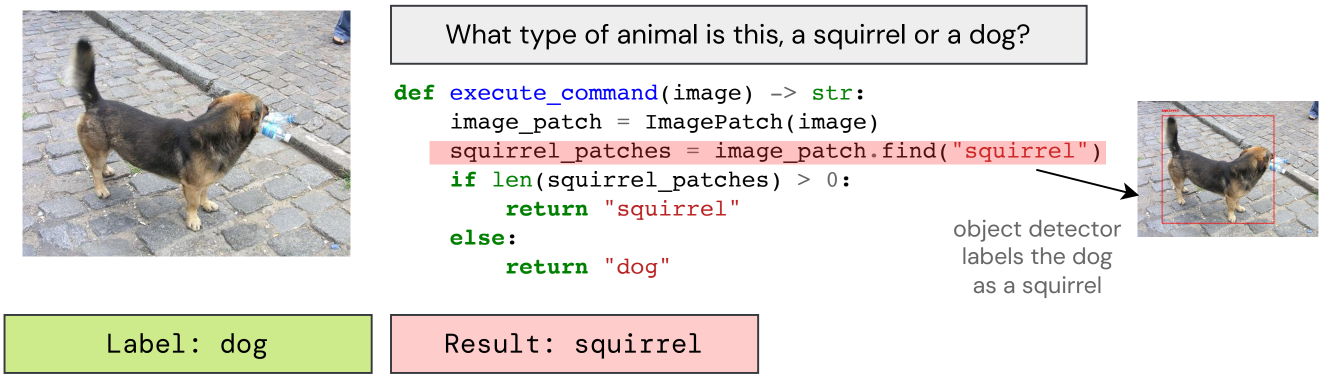


Figure 2. An example of a failure mode in which a perception module is unreliable on a simple input.

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

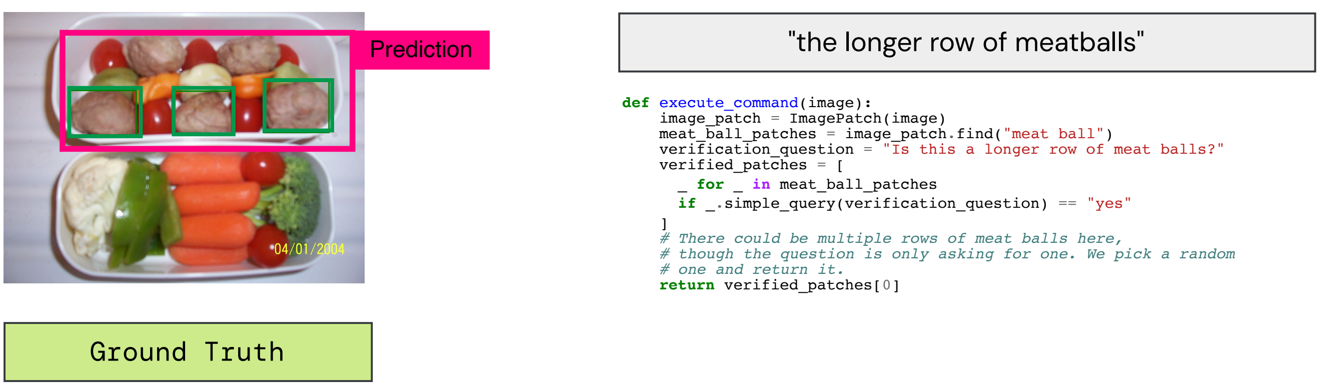


Figure 3. Verifying / detecting complex relationships is challenging for the program synthesis paradigm.

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.

Query: What is the color of the man hat?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
man_patches = image_patch.find("man")
hat_patches = [patch for patch in man_patches if patch.ex
ists("hat")]
hat_patch = hat_patches[0]
return hat_patch.simple_query("What is the color?")
```

result: "green" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
man_hat_patches = image_patch.find("man hat")
man_hat_patch = man_hat_patches[0]
return man_hat_patch.simple_query(
  "What is the color of the man hat?"
)
```

result: "brown" correct

Query: What is the color of the boat to the left of the umbrella?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
umbrella_patches = image_patch.find("umbrella")
umbrella_patch = umbrella_patches[0]
left_of_umbrella_patch = umbrella_patch.crop_left_of_bbox(
umbrella_patch.left, umbrella_patch.upper, umbrella_patch.right, u
mbrella_patch.lower
)
boat_patches = left_of_umbrella_patch.find("boat")
boat_patch = boat_patches[0]
return boat_patch.simple_query("What is the color?")
```

result: "blue" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
umbrella_patches = image_patch.find("umbrella")
umbrella_patch = umbrella_patches[0]
left_of_umbrella_patch = image_patch.crop_left_of_bbox(
  umbrella_patch.left, umbrella_patch.upper,
  umbrella_patch.right, umbrella_patch.lower)
return left_of_umbrella_patch.simple_query(
  "What is the color of the boat?")
```

result: "white" correct

Query: Is the weather partly cloudy or cloudless?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
partly_cloudy = image_patch.exists("partly cloudy")
cloudless = image_patch.exists("cloudless")
if partly_cloudy and not cloudless:
  return "partly cloudy"
elif cloudless and not partly_cloudy:
  return "cloudless"
else:
  return "unknown"
```

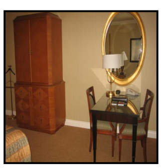
result: "unknown" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
return image_patch.simple_query(
  "Is the weather partly cloudy or cloudless?"
)
```

result: "partly cloudy" correct

Query: Are there either tables or cabinets that are not dark brown?



gpt-3.5-turbo

```
image_patch = ImagePatch(image)
table_patches = image_patch.find("tables")
cabinet_patches = image_patch.find("cabinets")
for table_patch in table_patches:
  if not table_patch.verify_property("it", "dark brown"):
    return "yes"
for cabinet_patch in cabinet_patches:
  if not cabinet_patch.verify_property("it", "dark brown"):
    return "yes"
return "no"
```

result: "yes" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
tables = image_patch.find("table")
tables_in_top_half = tables[0].verify_property("table", "dark brown")
if not tables_in_top_half:
  return "yes"
else:
  return "no"
cabinets = image_patch.find("cabinet")
for cabinet_patch in cabinets:
  if not cabinet_patch.verify_property("cabinet", "dark brown"):
    return "yes"
return "no"
```

result: "no" correct

Query: Is he a baseball player?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
baseball_player_patches = image_patch.find(
  "baseball player"
)
is_baseball_player = len(baseball_player_patches) > 0
return bool_to_yesno(is_baseball_player)
```

result: "yes" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
people = image_patch.find("person")
for person_patch in people:
  if person_patch.verify_property("person", "baseball player"):
    return "yes"
return "no"
```

result: "no" correct

Figure 4. Qualitative examples on VQA (GQA) showing errors made by gpt-3.5-turbo (ViperGPT) that are fixed by .

Query: What is the animal on the gray street?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
animal_patches = image_patch.find("animal")
gray_street_patches = image_patch.find("gray street")
for animal_patch in animal_patches:
  if animal_patch.verify_property("animal", "gray"):
    for street_patch in gray_street_patches:
      if animal_patch.left >= street_patch.left and animal_patch.right <= str
eet_patch.right and animal_patch.upper >= street_patch.upper and animal_patc
h.lower <= street_patch.lower:
        return animal_patch.simple_query("What is this animal?")
return "Animal not found on gray street."
```

result: "Animal not found on gray street" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
gray_street_patches = image_patch.find("gray street")
gray_street_patch = gray_street_patches[0]
left_of_gray_street_patch = gray_street_patch.left
right_of_gray_street_patch = gray_street_patch.right
for animal_patch in image_patch.find("animal"):
  if (
    animal_patch.left < right_of_gray_street_patch
    and animal_patch.right > left_of_gray_street_patch
  ):
  return animal_patch.simple_query("What is the animal?")
return image_patch.simple_query("What is the animal?")
```

result: "horse" correct

Query: Is metal used to make the fence on the sidewalk?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
fence_patches = image_patch.find("fence")
sidewalk_patches = image_patch.find("sidewalk")
fence_patch = fence_patches[0]
sidewalk_patch = sidewalk_patches[0]
fence_on_sidewalk = fence_patch.left >= sidewalk_patch.left and fe
nce_patch.right <= sidewalk_patch.right
return bool_to_yesno(fence_on_sidewalk and fence_patch.verify_prop
erty("fence", "metal"))
```

result: None wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
fences = image_patch.find("fence")
for fence_patch in fences:
  if fence_patch.verify_property("fence", "metal"):
    return bool_to_yesno(True)
return bool_to_yesno(False)
```

result: "yes" correct

Query: Is it outdoors?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
return image_patch.exist("outdoors")
```

result: False wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
return bool_to_yesno(
  image_patch.verify_property("scene", "outdoors")
)
```

result: "yes" correct

Query: Where is it, a beach or a field?

gpt-3.5-turbo

```
image_patch = ImagePatch(image)
locations = ["beach", "field"]
for location in locations:
  if image_patch.exists(location):
    return location
return "unknown"
```

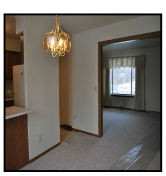
result: "beach" wrong

CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
places = ["beach", "field"]
for place in places:
  if image_patch.simple_query(f"Is this a {place}?") == "yes":
    return place
return image_patch.simple_query("Is this a beach or a field?")
```

result: "field" correct

Query: Is the carpet made of the same material as the frame?



gpt-3.5-turbo

```
image_patch = ImagePatch(image)
carpet_patches = image_patch.find("carpet")
frame_patches = image_patch.find("frame")
# Assume there's only one carpet and one frame patch.
carpet_patch = carpet_patches[0]
frame_patch = frame_patches[0]
same_material = carpet_patch.verify_property("carpet", fr
ame_patch.category)
return bool_to_yesno(same_material)
```

result: error wrong

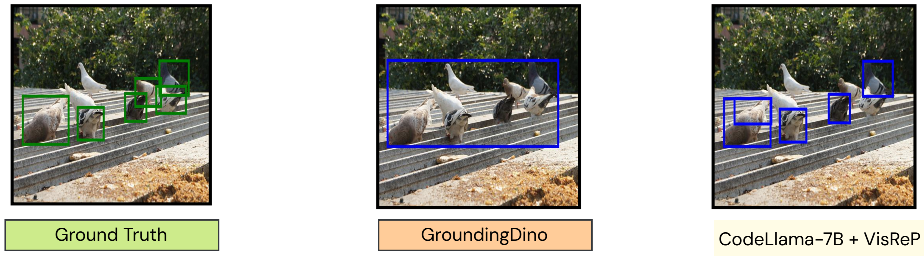
CodeLlama-7B + VisReP

```
image_patch = ImagePatch(image)
carpet_patches = image_patch.find("carpet")
carpet_patch = carpet_patches[0]
carpet_material = carpet_patch.simple_query("What is this material?")
frames = image_patch.find("frame")
for frame in frames:
  if frame.verify_property("frame", "carpet"):
    frame_material = frame.simple_query("What is this material?")
    if frame_material == carpet_material:
      return "yes"
  else:
    return "no"
return "no"
```

result: "no" correct

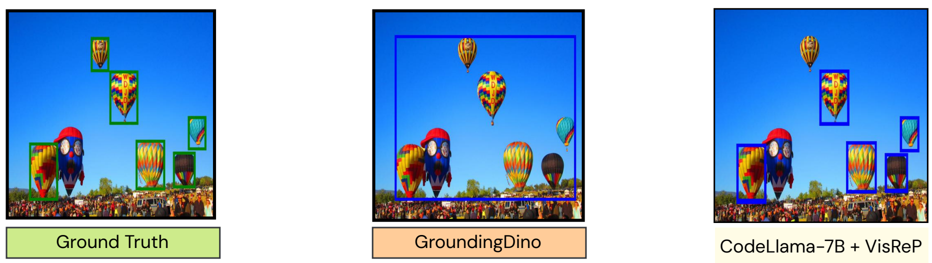
Figure 5. Qualitative examples on VQA (GQA) showing errors made by gpt-3.5-turbo (ViperGPT) that are fixed by .

"All the pigeons that aren't fully white"



Program
```
image_patch = ImagePatch(image)
pigeon_patches = image_patch.find("pigeon")
# We sort the pigeons by their color, so that the ones that are
# not fully white are at the end.
pigeon_patches.sort(key=lambda x: x.simple_query("What is the color?"))
# We then filter the pigeons to only keep the ones that are not
# fully white.
non_white_pigeon_patches = [_ for _ in pigeon_patches if _.simple_query("What is the color?") != "white"]
return non_white_pigeon_patches
```

"The balloons without glasses"



Program
```
image_patch = ImagePatch(image)
# Think of things that could be balloons.
search_terms = ["balloon", "air balloon"]
candidate_patches = []
# Then search for all of them.
for search_term in search_terms:
  candidate_patches.extend(image_patch.find(search_term))
verification_question = "Does this balloon have glasses?"
verified_patches = [_ for _ in candidate_patches if _.simple_query(verification_question) == "no"]
# There could be multiple items that fit this description
# so we return all of them.
return verified_patches
```

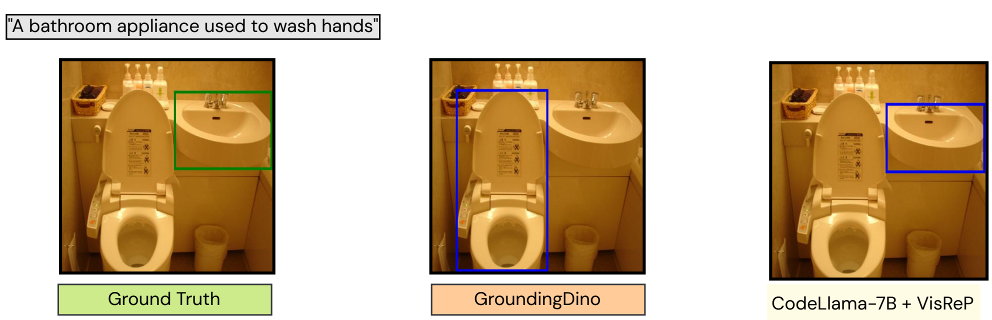
"The closer cookie"

Program
```
image_patch = ImagePatch(image)
# Think of things that could be cookies.
search_terms = ["cookie"]
candidate_patches = []
# Then search for all of them.
for search_term in search_terms:
  candidate_patches.extend(image_patch.find(search_term))
# We can't really define a question that would be applicable to all
# cookies, though, so we pick the one that is closest to the upper
# left corner of the image, since that will be the closest to the
# cookie that is closest to the edge of the image.
closest_cookie = min(candidate_patches, key=lambda x: x.vertical_center)
return closest_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+.

Program
```python
image_patch = ImagePatch(image)
# Think of things that could be stools.
search_terms = ["stool"]
candidate_patches = []
# Then search for all of them.
for search_term in search_terms:
  candidate_patches.extend(image_patch.find(search_term))
# The question is only asking for stools that aren't being used,
# so we filter the list to only include those.
verified_patches = [_ for _ in candidate_patches if _.simple_query("Is this stool being used?") == "no"]
# There could be multiple items that fit this description
# so we return all of them.
return verified_patches
```

Program
```python
image_patch = ImagePatch(image)
# Think of things that could be steaks.
search_terms = ["steak", "meat"]
candidate_patches = []
# Then search for all of them.
for search_term in search_terms:
candidate_patches.extend(image_patch.find(search_term))
# We need to filter out any steaks that are on a utensil, since
# the question is only asking for steaks not on a utensil.
verified_patches = [_ for _ in candidate_patches if _.verify_property("steak", "not on a utensil")]
# There could be multiple steaks that are not on a utensil here,
# though the question is only asking for one. We pick a random
# one and return it.
return verified_patches[0]
```



Program
```python
image_patch = ImagePatch(image)
# Think of things that could be bathroom appliances.
search_terms = ["bathroom appliance", "sink", "toilet"]
candidate_patches = []
# Then search for all of them.
for search_term in search_terms:
candidate_patches.extend(image_patch.find(search_term))
verification_question = "Is this a bathroom appliance used to wash the hands?"
verified_patches = [_ for _ in candidate_patches if _.simple_query(verification_question) == "yes"]
# There could be multiple items that fit this description
# so we return all of them.
return verified_patches
```

Figure 7. Qualitative examples on object detection (Omnilabel) showing errors made by a state-of-the-art detector (GroundingDino) that are fixed by CodeLlama+.

4. ImagePatch API

```
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
      # 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.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)
```

```
        >>> 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 s
        pecified 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
```

```
        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 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
```