Visual Programming: Compositional visual reasoning without training

Tanmay Gupta, Aniruddha Kembhavi
PRIOR @ Allen Institute for AI
https://prior.allenai.org/projects/visprog

1. Overview

This supplementary material includes

- Module implementation details (Sec. 2)
- Additional instruction tuning examples (Sec. 3)
- Task prompts for VisProg(Sec. 4)
- qualitative_results.pdf (attached) with additional visual rationales for successful and failure cases.

2. Module Details

To help understand the generated programs better, we now provide a few implementation details about some of the modules.

Select. The module takes a `query` and a `category` argument. When the `category` is provided, the selection is only performed over the regions that have been identified as belonging to that category by a previous module in the program (typically the `Seg` module). If `category` is `None`, the selection is performed over all regions. The query is the text to be used for region-text scoring to perform the selection. We use CLIP-ViT [2] to select the region with the maximum score for the query. When the query contains multiple phrases separated by commas, the highest-scoring region is selected for each phrase.

Classify. The `Classify` module takes lists of object regions and categories and tries to assign one of the categories to each region. For simplicity, we assume the images in the tagging task has at most 1 instance of each category. The `Classify` module operates differently based on whether the category list has 1 or more elements. If the category list has only 1 element, the category is assigned to the region with the highest CLIP score, similar to the `Select` module. When more than one category is provided, first, each region is assigned the category with the best score. Due to classification errors, this can lead to multiple regions being assigned the same category. Therefore, for each of the assigned categories (excluding the ones that were not assigned to any region), we perform a de-duplication step that retains only the maximum scoring region for each category.

List. The `List` module uses GPT3 to create a flexible and powerful knowledge retriever. Fig. 1 shows the prompt provided to GPT3 to retrieve factual knowledge.

```
Create comma separated lists based on the query.

Query: List at most 3 primary colors separated by commas
List:
red, blue, green

Query: List at most 2 north american states separated by commas
List:
California, Washington

Query: List at most {list_max} {new_query} separated by commas
List:
```

Figure 1. **Prompt for the `List` module.** `list_max` denotes the default maximum list length and `new_query` is the placeholder for the new retrieval query

3. Instruction Tuning

In Fig. 2, we provide additional examples of instruction tuning. These examples show various ways in which the original instructions may be modified to overcome failures. Common approaches include providing category names (*e.g*. "table-merged") to restrict search space for the `Select` module, improving the query for the localization module by providing more information (*e.g*. "most recent CEO of IBM" instead of just "CEO of IBM"), and adjusting the length of item list returned by the `List` module used for knowledge retrieval.

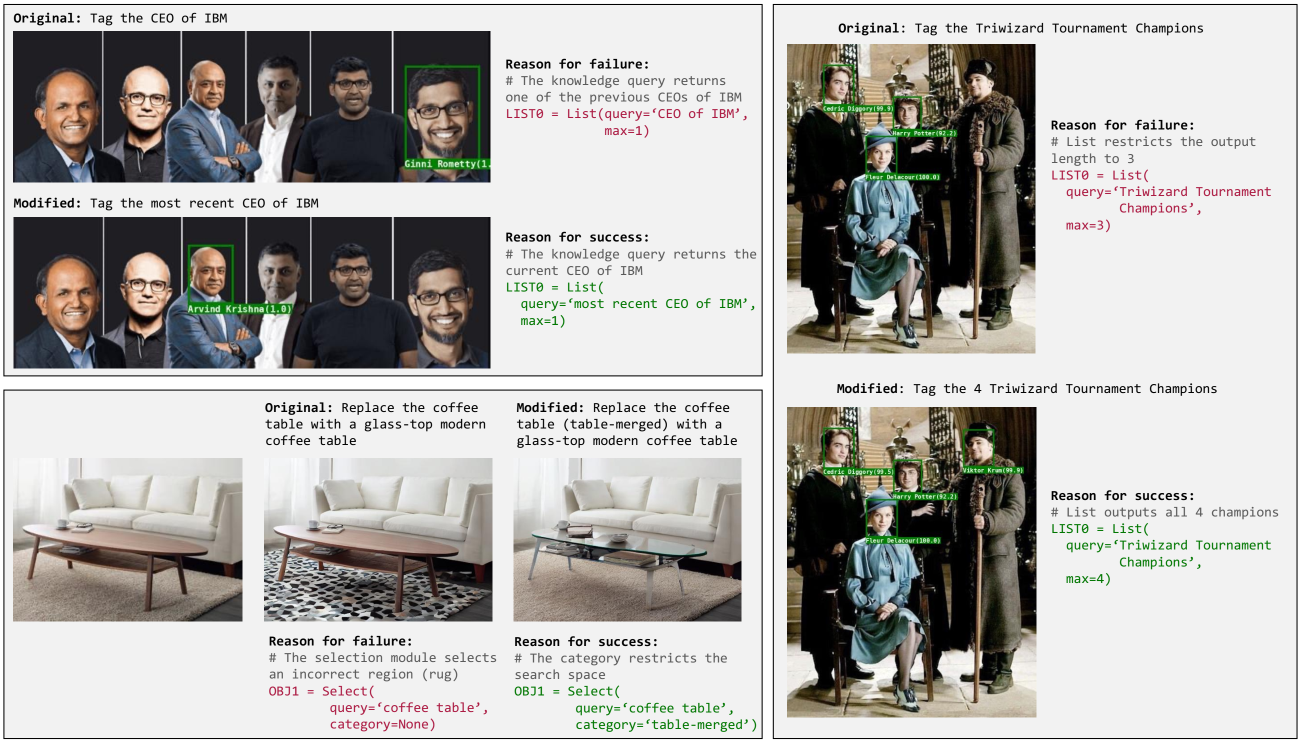
4. Task Prompts

We show the prompt structures for GQA (Figure 6), NLVR (Figure 5), knowledge tagging (Figure 3), and language-guided image editing (Figure 4) tasks with 3 in-context examples each.

References

[1] Zoe Papakipos and Joanna Bitton. Augly: Data augmentations for robustness. *ArXiv*, abs/2201.06494, 2022. 2

[2] 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



Original: Tag the CEO of IBM

Reason for failure:
```
# The knowledge query returns
one of the previous CEOs of IBM
LIST0 = List(query=‘CEO of IBM’,
             max=1)
```

Modified: Tag the most recent CEO of IBM

Reason for success:
```
# The knowledge query returns the
current CEO of IBM
LIST0 = List(
  query=‘most recent CEO of IBM’,
  max=1)
```

Original: Replace the coffee table with a glass-top modern coffee table

Modified: Replace the coffee table (table-merged) with a glass-top modern coffee table

Reason for failure:
```
# The selection module selects
an incorrect region (rug)
OBJ1 = Select(
        query=‘coffee table’,
        category=None)
```

Reason for success:
```
# The category restricts the
search space
OBJ1 = Select(
        query=‘coffee table’,
        category=‘table-merged’)
```

Original: Tag the Triwizard Tournament Champions

Reason for failure:
```
# List restricts the output
length to 3
LIST0 = List(
  query=‘Triwizard Tournament
         Champions’,
  max=3)
```

Modified: Tag the 4 Triwizard Tournament Champions

Reason for success:
```
# List outputs all 4 champions
LIST0 = List(
  query=‘Triwizard Tournament
         Champions’,
  max=4)
```

Figure 2. **Instruction tuning examples.** For failure cases, the original instruction can often be modified to improve performance.

```
Think step by step to carry out the instruction.

Instruction: Tag the 7 dwarfs in Snow White
Program:
OBJ0=FaceDet(image=IMAGE)
LIST0=List(query=‘dwarfs in snow white’, max=7)
OBJ1=Classify(image=IMAGE, object=OBJ0, categories=LIST0)
IMAGE0=Tag(image=IMAGE, object=OBJ1)

Instruction: Tag the presidents of US
Program:
OBJ0=FaceDet(image=IMAGE)
LIST0=List(query=‘presidents of US’, max={list_max})
OBJ1=Classify(image=IMAGE, object=OBJ0, categories=LIST0)
IMAGE0=Tag(image=IMAGE, object=OBJ1)

Instruction: Tag the shoes (4) by their type
Program:
OBJ0=Loc(image=IMAGE,object=‘shoe’)
LIST0=List(query=‘type of shoes’, max=4)
OBJ1=Classify(image=IMAGE, object=OBJ0, categories=LIST0)
IMAGE0=Tag(image=IMAGE, object=OBJ1)

Instruction: {new_instruction}
Program:
```

Figure 3. **Knowledge tagging prompt.** Note that the prompt has an additional placeholder to configure the default `max` value for `List` module. While the first example infers `max` from a natural instruction, the third example demonstrates how a user might minimally augment a natural instruction to provide argument values.

```
Think step by step to carry out the instruction.

Emoji Options:
:p = face_with_tongue
8) = smiling_face_with_sunglasses
:) = smiling_face
;) = winking_face

Instruction: Hide the face of Nicole Kidman with :p
Program:
OBJ0=Facedet(image=IMAGE)
OBJ1=Select(image=IMAGE, object=OBJ0, query=‘Nicole Kidman’, category=None)
IMAGE0=Emoji(image=IMAGE, object=OBJ1, emoji=‘face_with_tongue’, category=None)
RESULT=IMAGE0

Instruction: Create a color pop of the girl and umbrella
Program:
OBJ0=Seg(image=IMAGE)
OBJ1=Select(image=IMAGE, object=OBJ0, query=‘girl,umbrella’)
IMAGE0=ColorPop(image=IMAGE, object=OBJ1)
RESULT=IMAGE0

Instruction: Replace the red bus (bus) with a blue bus
Program:
OBJ0=Seg(image=IMAGE)
OBJ1=Select(image=IMAGE, object=OBJ0, query=‘red bus’, category=‘bus’)
IMAGE0=Replace(image=IMAGE, object=OBJ1, prompt=‘blue bus’)
RESULT=IMAGE0

Instruction: {new_instruction}
Program:
```

Figure 4. **Image editing prompt**. Note that the prompt includes a mapping of emojis to their names in the AugLy [1] library that is used to implement `Emoji` module. The third example shows how to provide the `category` value for the `Select` module.

```
Think step by step if the statement is True or False.

Statement: There are at least seven wine bottles in the image on the left
Program:
BOX0=Loc(image=LEFT, object=‘wine bottle’)
ANSWER0=Count(bbox=BOX0)
ANSWER1=Eval(‘{ANSWER0} >= 7’)
RESULT=ANSWER1

Statement: One dog is laying down.
Program:
BOX0=Loc(image=LEFT, object=‘dog laying down’)
ANSWER0=Count(bbox=BOX0)
BOX1=Loc(image=RIGHT, object=‘dog laying down’)
ANSWER1=Count(bbox=BOX1)
ANSWER2=Eval(‘{ANSWER0} + {ANSWER1} == 1’)
RESULT=ANSWER2

Statement: The flowers in the clear glass vase are white with green stems.
Program:
BOX0=Loc(image=LEFT, object=‘flower in clear glass vase’)
ANSWER0=Count(bbox=BOX0)
IMAGE0=Crop(image=LEFT, bbox=BOX0)
ANSWER1=Vqa(image=IMAGE0, question=‘Are the flowers in the vase white?’)
ANSWER2=Vqa(image=IMAGE0, question=‘Are the stems of the flowers green?’)
BOX1=Loc(image=RIGHT, object=‘flower in clear glass vase’)
ANSWER3=Count(bbox=BOX1)
IMAGE1=Crop(image=RIGHT, bbox=BOX1)
ANSWER4=Vqa(image=IMAGE1, question=‘Are the flowers in the vase white?’)
ANSWER5=Vqa(image=IMAGE1, question=‘Are the stems of the flowers green?’)
ANSWER6=Eval(‘{ANSWER0} == 1 and {ANSWER1} and {ANSWER2}’)
ANSWER7=Eval(‘{ANSWER3} == 1 and {ANSWER4} and {ANSWER5}’)
ANSWER8=Eval(‘{ANSWER6} xor {ANSWER7}’)
RESULT=ANSWER8

Statement: {new_statement}
Program:
```

Figure 5. **NLVR prompt**

```
Think step by step to answer the question.

Question: Is the vehicle in the top of the image?
Program:
BOX0=Loc(image=IMAGE, object=‘TOP’)
IMAGE0=Crop(image=IMAGE, bbox: BOX0)
BOX1=Loc(image=IMAGE0, object=‘vehicle’)
ANSWER0=Count(bbox=BOX1)
ANSWER1=Eval(“'yes' if {ANSWER0} > 0 else 'no'”)
RESULT=ANSWER1

Question: What color is the curtain that is to the right of the mirror?
Program:
BOX0=Loc(image=IMAGE, object=‘mirror’)
IMAGE0=CropRight(image=IMAGE, bbox=BOX0)
ANSWER0=Vqa(image=IMAGE0, question=‘What color is the curtain?’)
RESULT=ANSWER0

Question: Does the traffic cone have white color?
Program:
BOX0=Loc(image=IMAGE, object=‘traffic cone’)
IMAGE0=Crop(image=IMAGE, bbox=BOX0)
ANSWER0=Vqa(image=IMAGE0, question=‘What color is the traffic cone?’)
ANSWER1=Eval(“'yes' if {ANSWER0} == 'white' else 'no'”)
RESULT=ANSWER1

Question: {new_question}
Program:
```

Figure 6. **GQA prompt**