ViperGPT: Visual Interface via Python Execution for Reasoning

Dídac Surís, Sachit Menon, Carl Vondrick

Presented by Dohhyun Kim, Andy Lauer

Motivation

Visual Queries

- Textual question about an image
- Requires both visual understanding and reasoning
- Example: "How many muffins can each kid have for it to be fair?"
  1. Find the muffins and children in image
  2. Count how many there are
  3. Determine the muffins should be divided
- Inherently compositional problems

Current Approaches

- End-to-end models
    - Black box results
    - Must perform all tasks in a single pass
    - Loses advantages of specialized models
    - Computers can do math without machine learning
    - Requires training of entire model

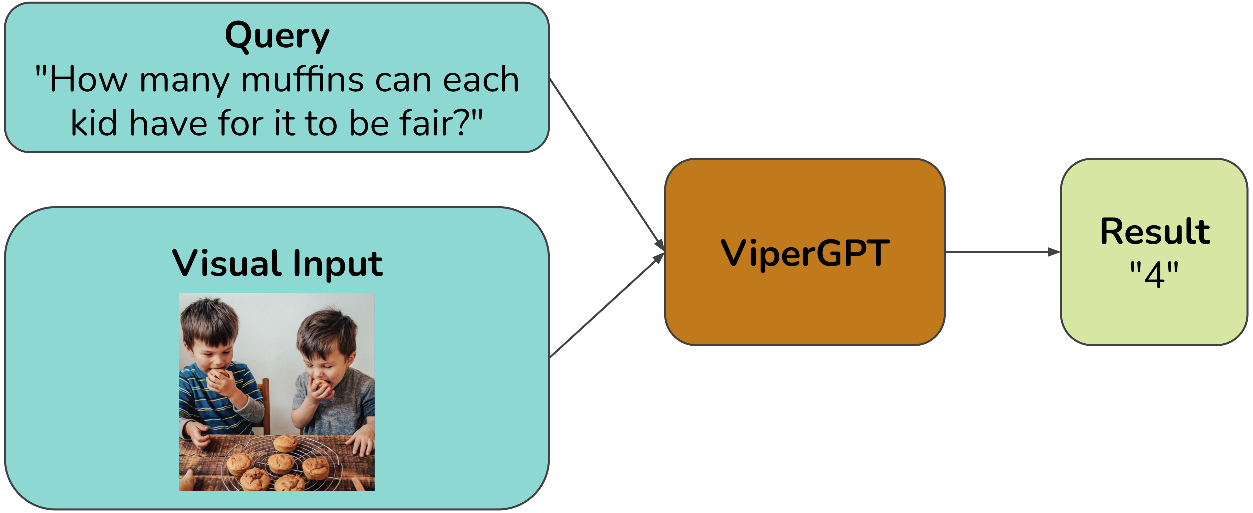
- Can we take advantage of composing individual models to solve a larger task?

Method

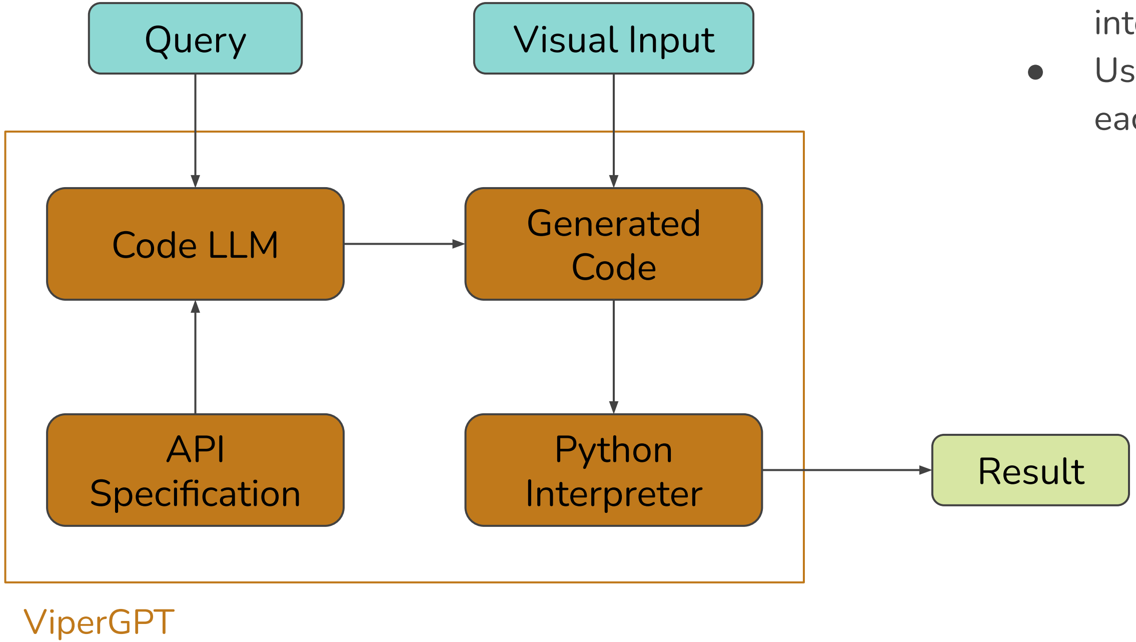
Architecture

ViperGPT: "A framework that leverages code-generation models to compose vision-and-language models into subroutines"

- Main idea: break complex task into modular sub-tasks
- Use specialized models to solve each sub-task via API calls



Architecture



- Main idea: break complex task into modular sub-tasks
- Use specialized models to solve each sub-task via API calls

API

- Defines functions available to the code generation model
- Each function performs a specific task
  - Typically implemented with a specialized model
- Examples: `find()`, `exists()`, `verify_property()`, `simple_query()`
- Only definition and docstring given to code LLM
  - Enables abstraction from implementation details
  - Improved modularity

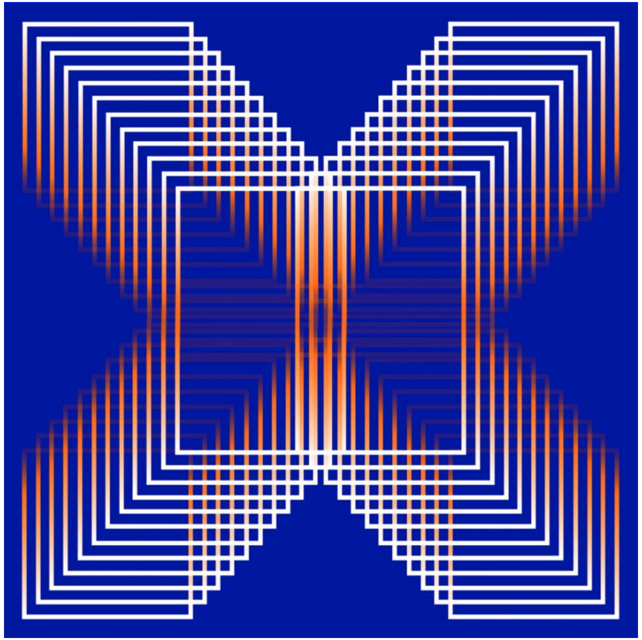
```python
def find(self, object_name: str) -> List[ImagePatch]:
    """Returns a list of ImagePatch objects matching object_name contained in the crop if any are found.
    Otherwise, returns an empty list.
    Parameters
    ----------
    object_name : str
        the name of the object to be found

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

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

def exists(self, object_name: str) -> bool:
    """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 cakes and gummy bears in the photo?
    >>> def execute_command(image)->str:
    >>>     image_patch = ImagePatch(image)
    >>>     is_cake = image_patch.exists("cake")
    >>>     is_gummy_bear = image_patch.exists("gummy bear")
    >>>     return bool_to_yesno(is_cake and is_gummy_bear)
    """
    return len(self.find(object_name)) > 0
```

Pretrained Models Used

Model	Task	Example API call
GLIP	Object detection	`find("drink")`, `exists("boy")`
X-VLM	Text-image similarity	`verify_property("bookcase", "wood")`
MiDaS	Depth estimation	`pizza.compute_depth()`
GPT-3	External knowledge	`llm_query("Who is the founder of {car_brand}?")`
BLIP-2	Simple visual queries	`simple_query("What toy is this?")`

Code Generation

- Given input query and API, create Python program to complete task
- Codex from OpenAI
- Trained on natural language and internet Python code



Generated Code Sample

Query: Are there water bottles to the right of the bookcase that is made of wood?

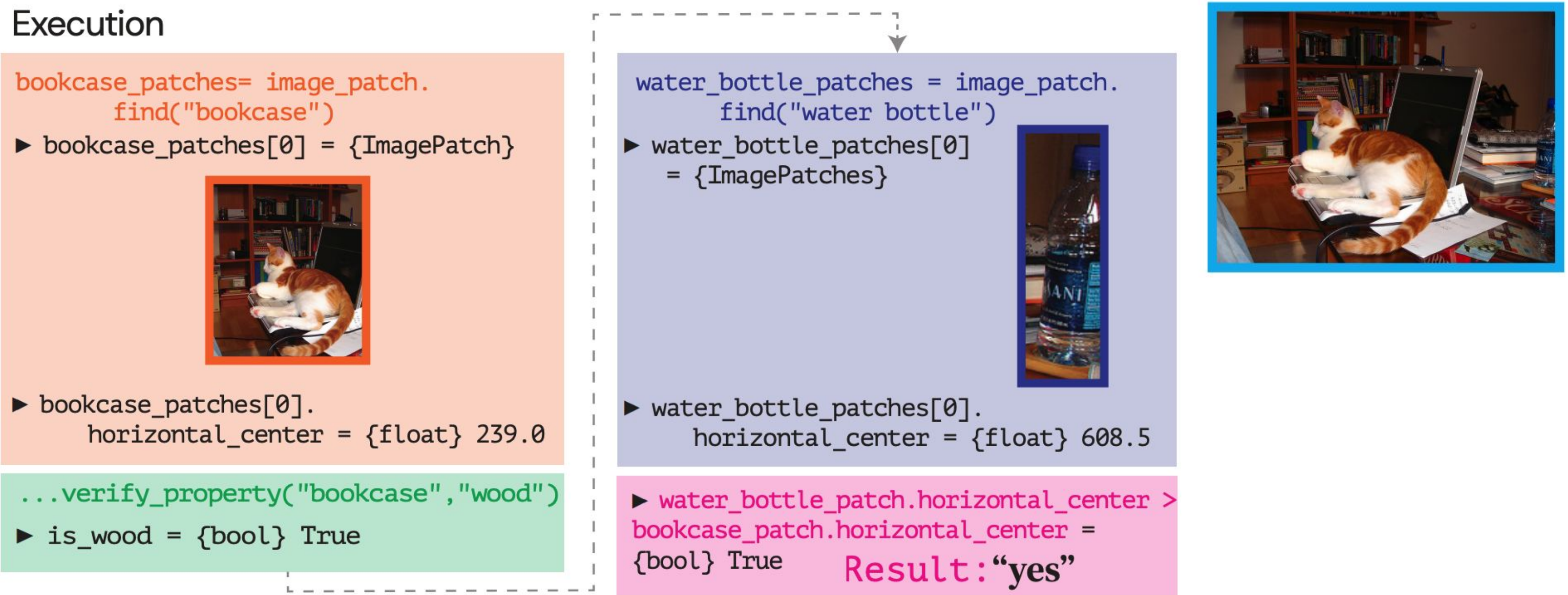
In:

Generated code

```
def execute_command(image):
    image_patch = ImagePatch(image)
    bookcase_patches = image_patch.find("bookcase")
    for bookcase_patch in bookcase_patches:
        is_wood = bookcase_patch.verify_property("bookcase", "wood")
        if is_wood:
            water_bottle_patches = image_patch.find("water bottle")
            for water_bottle_patch in water_bottle_patches:
                if water_bottle_patch.horizontal_center > \
                bookcase_patch.horizontal_center:
                    return "yes"
            return "no"
    return "no"
```

Generated Code Sample

Query: Are there water bottles to the right of the bookcase that is made of wood?



Experiments

Experimental Setup

- Select four tasks for evaluation
  - Visual grounding
  - Compositional image question answering
  - External knowledge-dependent image question answering
  - Video causal and temporal reasoning
- Datasets
  - GQA - Composition
  - OK-VQA - External Knowledge
  - NExT-QA - Visual Reasoning for videos

“We believe that the evident strength of this approach may not be adequately explored by existing benchmarks”

GQA

Table 2. **GQA Results**. We report accuracy on the test-dev set.

		Accuracy (%) ↑
Sup.	LGCN [20]	55.8
	LXMERT [51]	60.0
	NSM [24]	63.0
	CRF [39]	72.1
ZS	BLIP-2 [30]	44.7
	ViperGPT (ours)	**48.1**

GQA

- **GQA**. The GQA API contains all the contents in the API from Listing 1 up until the `llm_query` function, which is not used. The `ImagePatch` usage examples look like the following:

```python
# Is there a backpack to the right of the man?
def execute_command(image)->str:
    image_patch = ImagePatch(image)
    man_patches = image_patch.find("man")
    # Question assumes one man patch
    if len(man_patches) == 0:
        # If no man is found, query the image directly
        return image_patch.simple_query("Is there a backpack to the right of the man?")
    man_patch = man_patches[0]
    backpack_patches = image_patch.find("backpack")
    # Question assumes one backpack patch
    if len(backpack_patches) == 0:
        return "no"
    for backpack_patch in backpack_patches:
        if backpack_patch.horizontal_center > man_patch.horizontal_center:
            return "yes"
    return "no"
```

Listing 3. **GQA example.**

OK-VQA

Table 3. **OK-VQA Results.**

		Accuracy (%) ↑
Sup.	TRiG [13]	50.5
	KAT [16]	54.4
	RA-VQA [32]	54.5
	REVIVE [33]	58.0
	PromptCap [21]	58.8
ZS	PNP-VQA [52]	35.9
	PICa [60]	43.3
	BLIP-2 [30]	45.9
	Flamingo [1]	50.6
	ViperGPT (ours)	**51.9**

Model	LLM Backbone	OKVQA
Flamingo	Chinchilla-7B	44.7
BLIP-2	Flan-T5$_{XXL}$(13B)	45.9
LLaVA	Vicuna-13B	54.4
MiniGPT-4	Vicuna-13B	37.5
InstructBLIP	Vicuna-7B	–
InstructBLIP	Vicuna-13B	–
Shikra	Vicuna-13B	47.2
IDEFICS-9B	LLaMA-7B	–
IDEFICS-80B	LLaMA-65B	–
Qwen-VL	Qwen-7B	–
Qwen-VL-Chat	Qwen-7B	–
LLaVA-1.5	Vicuna-1.5-7B	–
+ShareGPT4V	Vicuna-1.5-7B	–
LLaVA-1.5	Vicuna-1.5-13B	–
MiniGPT-v2	LLaMA-2-Chat-7B	56.9
MiniGPT-v2-Chat	LLaMA-2-Chat-7B	55.9
VILA-7B	LLaMA-2-7B	–
VILA-13B	LLaMA-2-13B	–
+ShareGPT4V	LLaMA-2-13B	–

OK-VQA

Table 3. **OK-VQA Results.**

- **OK-VQA**. The API only uses the `simple_query` method from `ImagePatch`. It additionally uses the `llm_query` function. The `ImagePatch` usage examples look like the following:

```

# Who is famous for allegedly doing this in a lightning storm?
def execute_command(image)->str:
    # The question is not direct perception, so we need to ask the image for more information
    # Salient information: what is being done?
    image = ImagePatch(image)
    guesses = []
    action = image.simple_query("What is being done?")
    external_knowledge_query = "Who is famous for allegedly {} in a lightning storm?".format(action)
    step_by_step_guess = llm_query(external_knowledge_query)
    guesses.append("what is being done is {}".format(action) + ", so " + step_by_step_guess)
    direct_guess = image.simple_query("Who is famous for allegedly doing this in a lightning storm?")
    guesses.append(direct_guess)
    return process_guesses("Who is famous for allegedly doing this in a lightning storm?", guesses)
```

Listing 4. **OK-VQA example.**

NExT-QA

Table 4. **NExT-QA Results**. Our method gets overall state-of-the-art results (including *supervised* models) on the hard split. "T" and "C" stand for "temporal" and "causal" questions, respectively.

		Accuracy (%) ↑		
		Hard Split - T	Hard Split - C	Full Set
Sup.	ATP [7]	45.3	43.3	54.3
	VGT [58]	-	-	56.9
	HiTeA [61]	48.6	47.8	63.1
ZS	ViperGPT (ours)	**49.8**	**56.4**	60.0

Methods		Val				ATP-hard subset		
		Acc@C	Acc@T	Acc@D	Acc@All	Acc@C	Acc@T	Acc@All
Supervised								
VFC [57]	[ICCV2021]	49.6	51.5	63.2	52.3	-	-	-
ATP [4]	[CVPR2022]	53.1	50.2	66.8	54.3	38.4	36.5	38.8
MIST [7]	[CVPR2023]	54.6	56.6	66.9	57.2	-	-	-
GF [1]	[NeurIPS2023]	56.9	57.1	70.5	58.8	48.7	50.3	49.3
CoVGT [54]	[TPAMI2023]	59.7	58.0	69.9	60.7	-	-	-
SeViT [15]	[arXiv2023.1]	54.0	54.1	71.3	56.7	43.3	46.5	-
HiTeA [64]	[ICCV2023]	62.4	58.3	75.6	63.1	47.8	48.6	-
Zero-shot								
VFC [29]	[ICCV2023]	51.6	45.4	64.1	51.5	32.2	30.0	31.4
InternVideo [51]	[arXiv2022.12]	43.4	48.0	65.1	49.1	-	-	-
AssistGPT [6]	[arXiv2023.6]	60.0	51.4	67.3	58.4	-	-	-
ViperGPT [45]	[ICCV2023]	-	-	-	60.0	-	-	-
SeViLA [66]	[NeurIPS2023]	61.3	61.5	75.6	63.6	-	-	-
LLoVi [67]	[arXiv2024.2]	69.5	61.0	75.6	67.7	-	-	-
VideoAgent	(ours)	**72.7**	**64.5**	**81.1**	**71.3**	**57.8**	**58.8**	**58.4**

Table 3: *Results on NExT-QA compared to the state of the art.* C, T, and D are causal, temporal, and descriptive subsets, respectively.

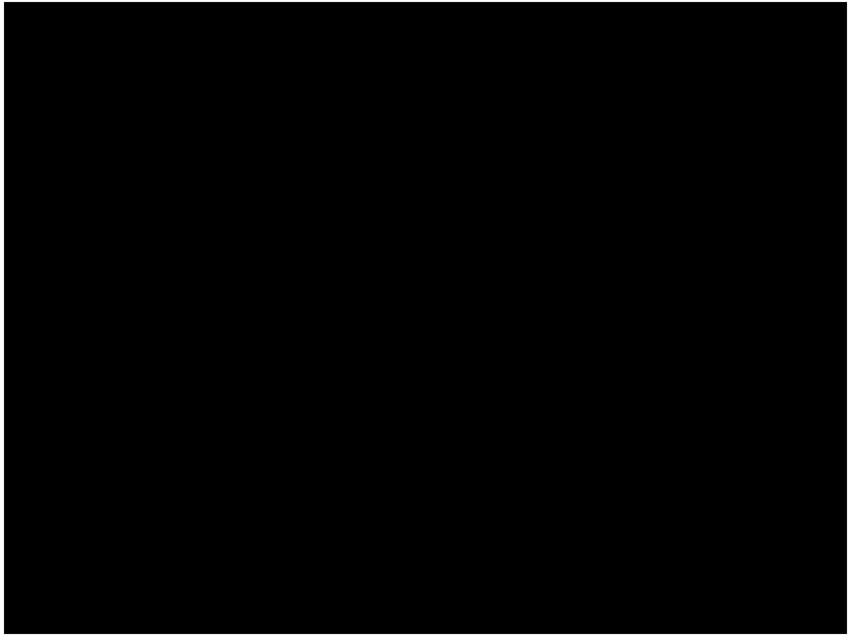
NExT-QA

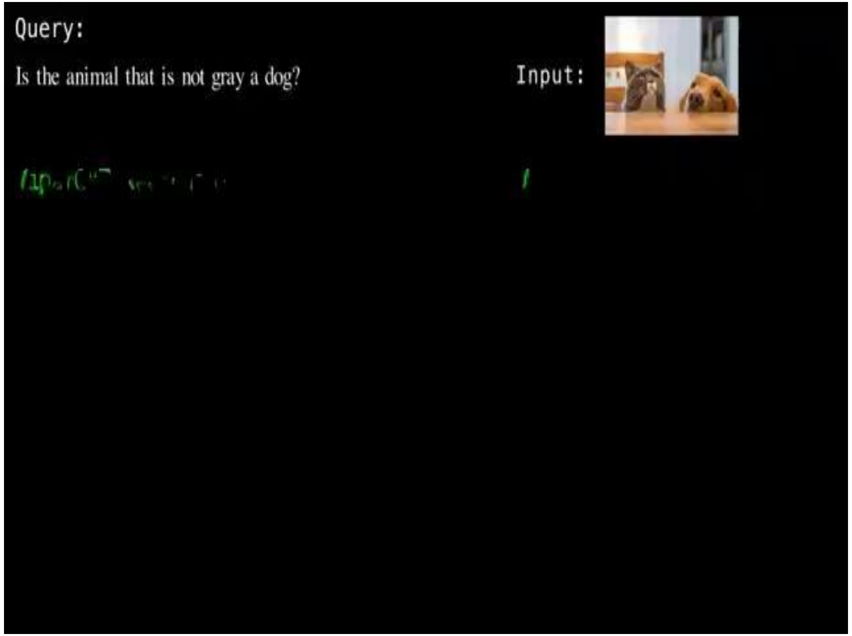
- **NeXT-QA.** The `VideoSegment` class is added to the API definition, and the available `ImagePatch` methods are `find`, `exists`, `best_text_match` and `simple_query`. The function `best_image_match` is also used. The `ImagePatch` usage examples look like:

```python
# why does the man with a red hat put his arm down at the end of the video
# possible answers: ['watching television', 'searching for food', 'move its head', 'looking over cardboard box', 'looks at the camera']
def execute_command(video, possible_answers, question)->[str, dict]:
    # Reason every step
    video_segment = VideoSegment(video)
    # Caption last frame of the video (end of video)
    last_frame = ImagePatch(video_segment, -1)
    last_caption = last_frame.simple_query("What is this?")
    men = last_frame.find("man")
    if len(men) == 0:
        men = [last_frame]
    man = men[0]
    man_action = man.simple_query("What is the man doing?")
    # Answer the question. Remember to create the info dictionary
    info = {
        "Caption of last frame": last_caption,
        "Man looks like he is doing": man_action
    }
    answer = video_segment.select_answer(info, question, possible_answers)
    return answer, info
```

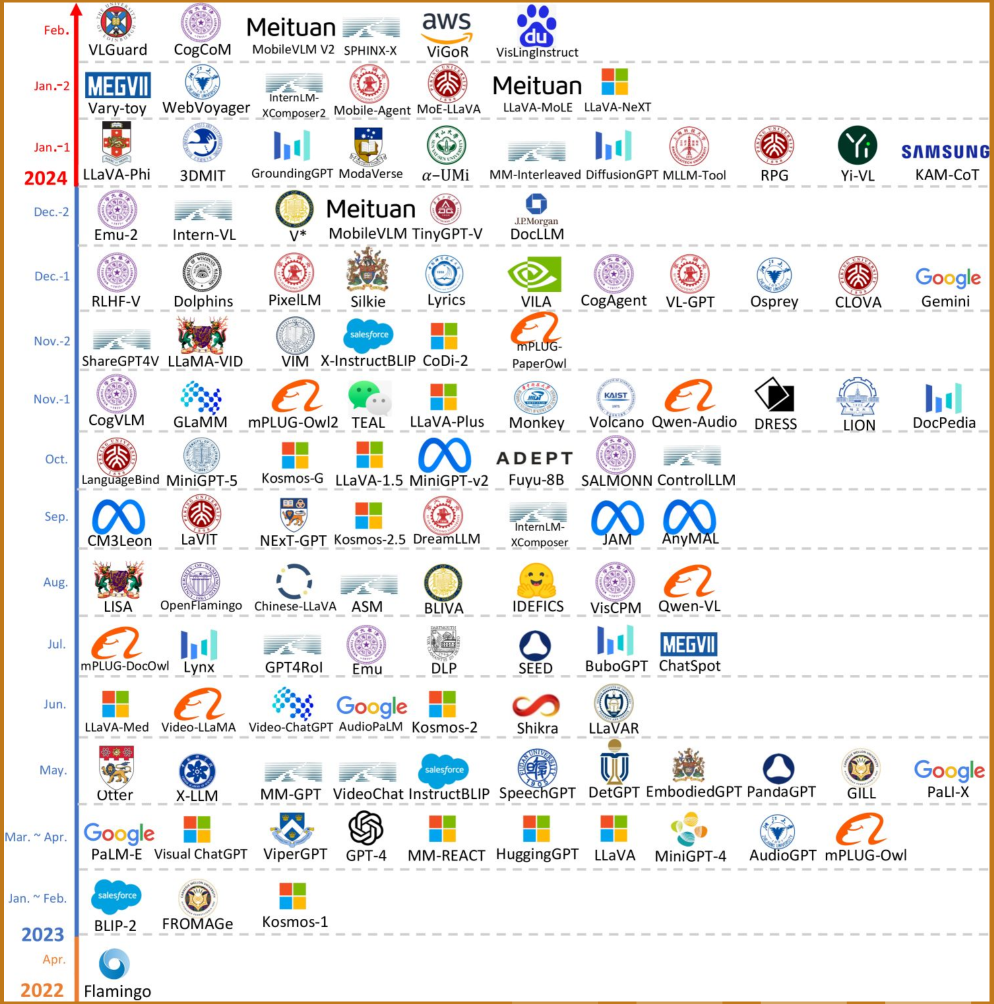
Listing 5. **NeXT-QA example.**

Examples of Logic and Math



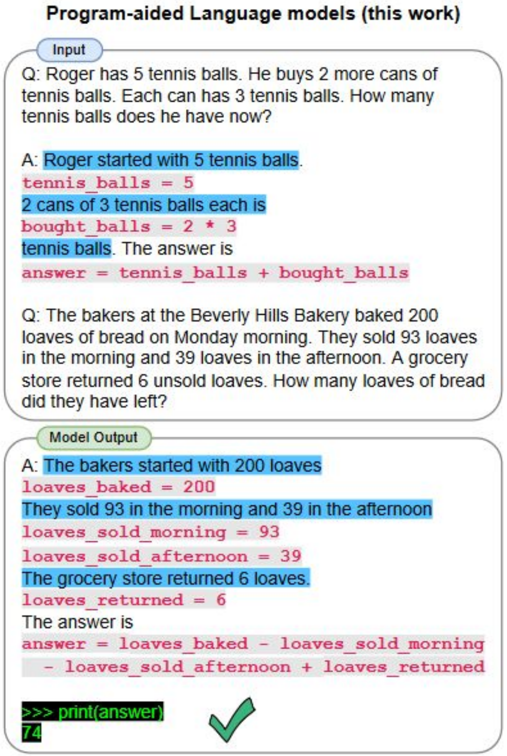


Related Works



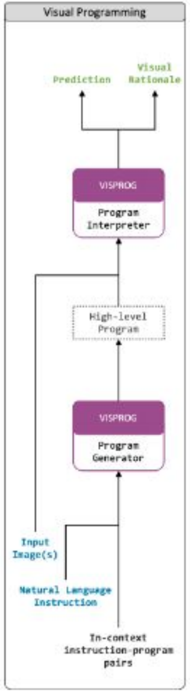
PAL: Program-aided Language Models

- Gao, Madaan, Zhou et al.
- Published Nov. 2022(4 months before)
- Proposed code generation for logical reasoning
- Excludes vision component



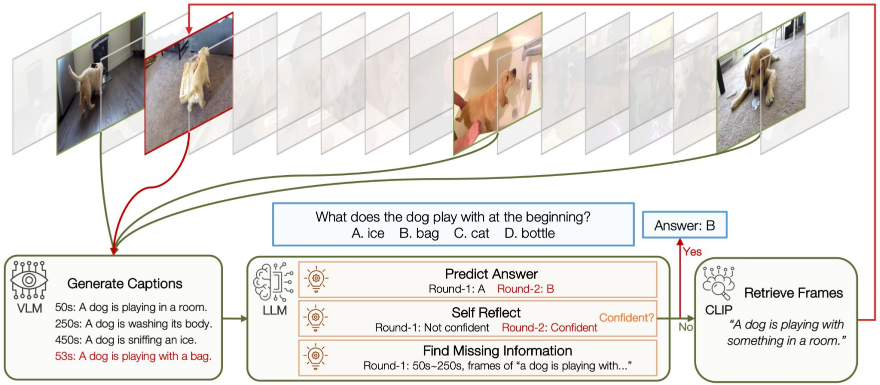
VisProg: Compositional visual reasoning without training

- Gupta et al.
- Published Nov. 2022(4 months before)
- Covers a wide range of image tasks, not just QA
- Does not generate actual code, just pseudocode

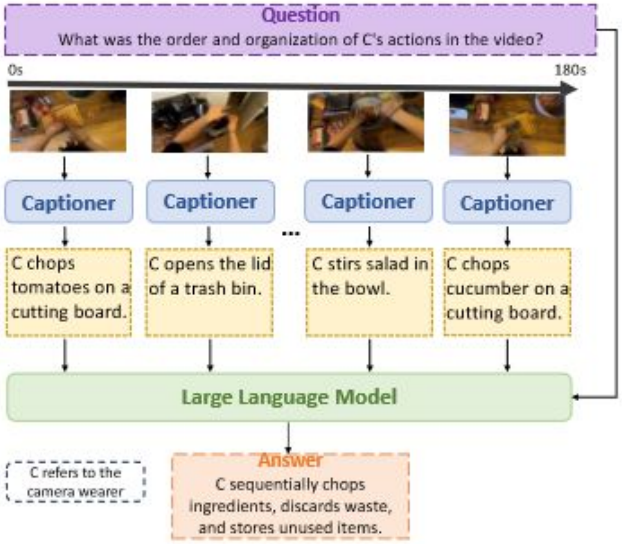


Task	Input	Output	Modules
Compositional Visual QA (GQA)	Image + Question	Text	Loc, Vqa, Eval, Count, Crop, CropLeft, CropRight, CropAbove, CropBelow
Reasoning on Image Pairs (NLVR)	Image Pair + Statement	True/False	Vqa, Eval
Factual Knowledge Object Tagging	Image + Instruction	Image	FaceDet, List, Classify, Loc, Tag
Image Editing with Natural Language	Image + Instruction	Image	FaceDet, Seg, Select, Replace, ColorPop, BgBlur, Emoji

VideoAgent/LLoVi



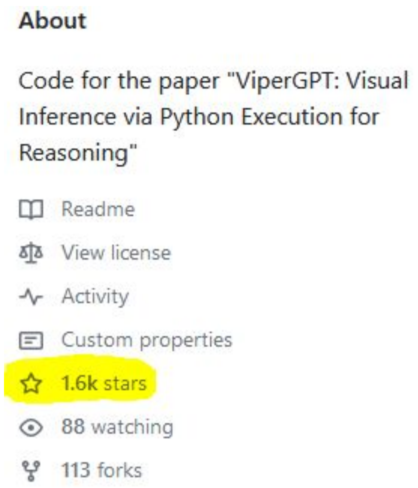
- Wang and Zhang et al. | Zhang and Lu et al.
- Published Feb., Mar. 2024(11,12 months after)
- No code generation
- Similar in terms of combining VLM + LLM
- Outperforms ViperGPT, current SOTAs
- Excels at long-range understanding



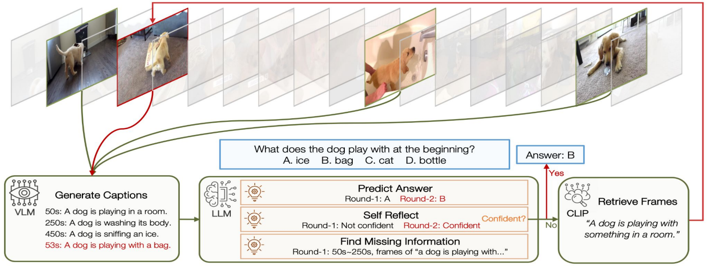
Our Thoughts

Strengths

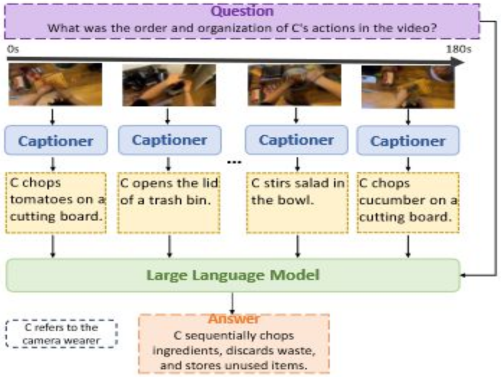
- Modularity, adoptable to tasks and future improvements
- No need to train
- Good at generalization
- Foundational model
- Transparency, easier to benchmark performance



Weaknesses



- Model
    - Re-uses existing models
    - Reliant on API access
    - Longer runtime due to larger modules?
    - Perhaps not as big of impact as expected?
- Paper
    - Experimental section did not measure computing costs/inference time
    - Lack of ablations



Questions