ViperGPT: Visual Inference via Python Execution for Reasoning

Dídac Surís*, Sachit Menon*, Carl Vondrick
Columbia University, ICCV, 2023

Presented by Chengyue Huang, Aditya Chandaliya

Outline

- Problem Statement
- Related Works
- Approach
- Experiments & Results
- Limitations, Societal Implications
- Summary of Strengths, Weaknesses, Relationship to Other Papers

Problem Statement: VLM Reasoning Tasks

- Visual Grounding
  - Identifying the bounding box in an image that corresponds best to a given query.

Query: pizza front

- Compositional Image Question Answering
  - Decomposing complex questions into simpler tasks.

Query: Does that pancake look brown and round?

- External Knowledge-dependent Image Question Answering
  - Many questions about images can only be answered correctly by integrating outside knowledge about the world.

Query: The real live version of this toy does what in the winter?

Georgia Tech

Problem Statement

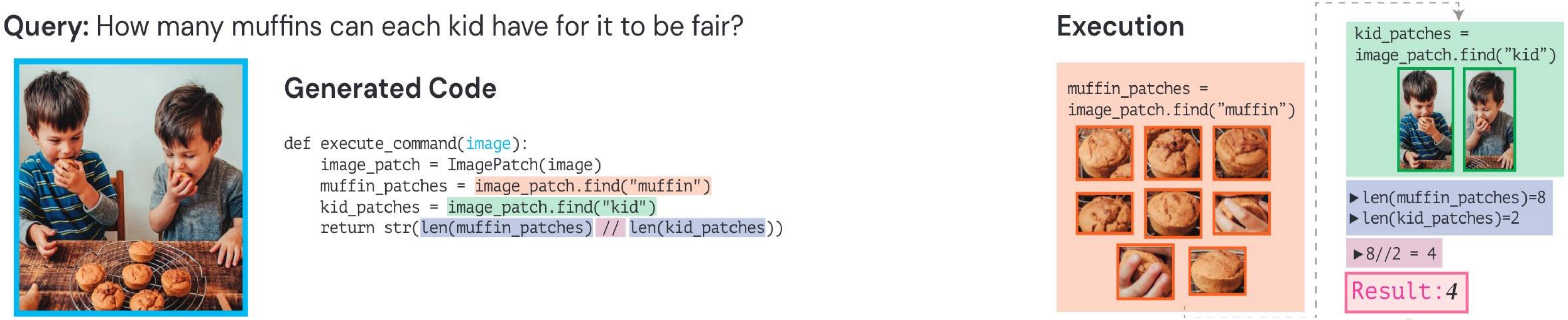
- Query: How many muffins can each kid have for it to be fair?
    1) Find the children and the muffins in the image
    2) Count how many there are of each
    3) Reason that 'fair' implies an even split, hence divide.

End-to-end models do not inherently leverage compositional reasoning.

- They fail to make use of
    - Advances in fundamental vision tasks at different steps
    - Computers can perform mathematical operations (e.g., division) easily without machine learning
- Uninterpretable decisions
    - No way to audit the result of each step to diagnose failure
    - Model becomes increasingly untenable as the data and computation grow

Problem Statement: ViperGPT

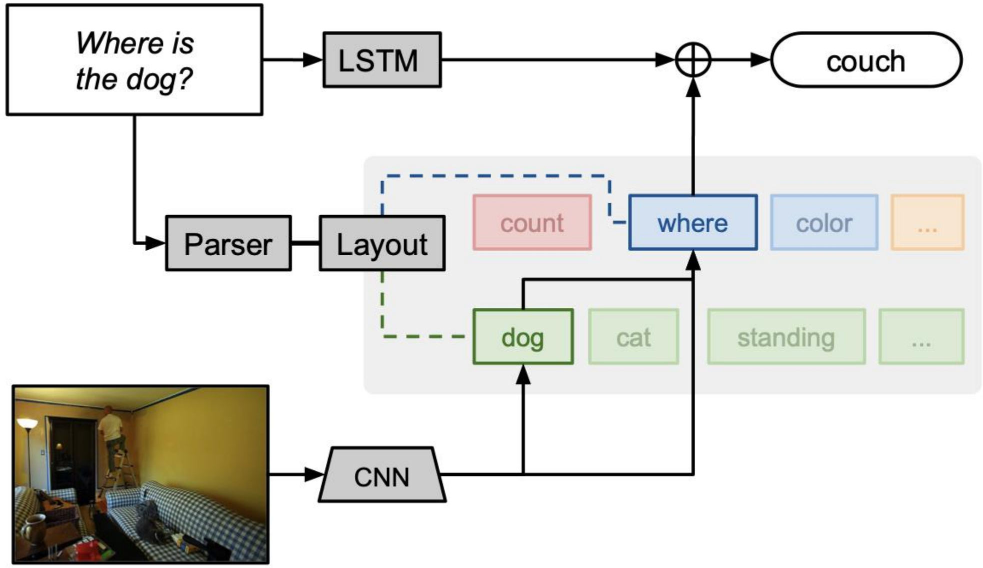
- Interpretability
  - Explicit code function calls for each step
  - Intermediate values that can be inspected
- Flexibility: Easily incorporate any vision or language module
- Composability: Decompose tasks into smaller sub-tasks performed step-by-step
- Training-free: Recombine existing models in new ways without additional training
- Generalizability: Unify all tasks into one system

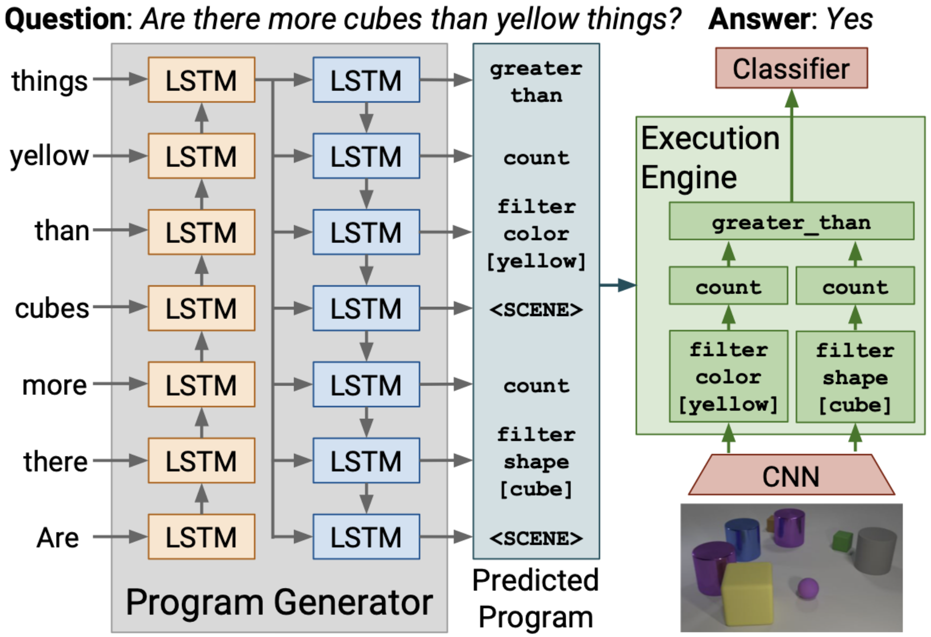


Related Works

Related Works: Neural Module Networks

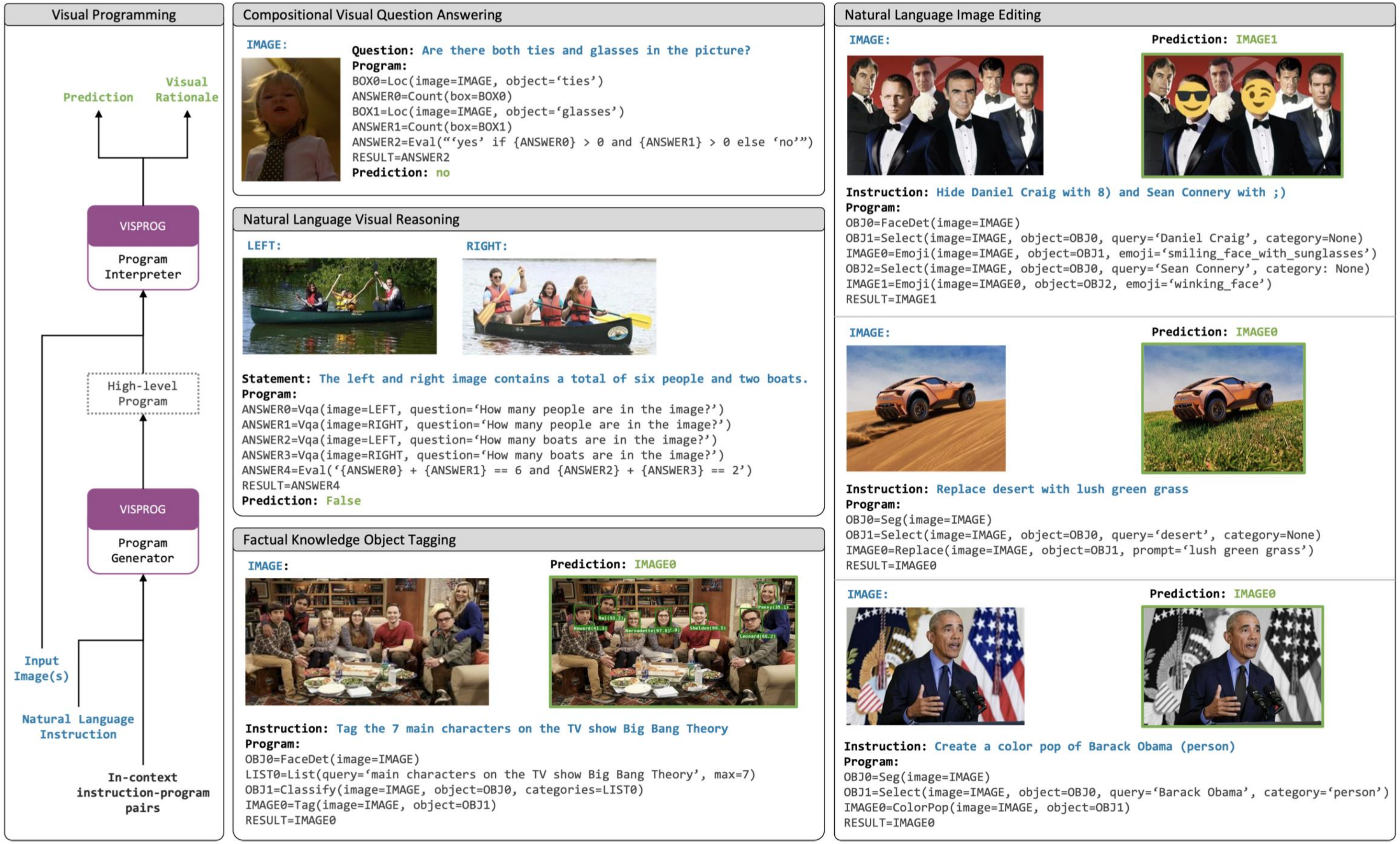
- Intuition: decompose tasks into simpler modules
  - Training end-to-end with modules rearranged in different ways for different problems
  - Each module would learn their appropriate function
- Cons
  - Expensive supervision in the form of programs → domain-limited
  - End-to-end training: learn the perceptual models jointly with the program generator → fail to produce the intended modular structure





Andreas, Jacob, et al. "Neural module networks." *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016.
Johnson, Justin, et al. "Inferring and executing programs for visual reasoning." *Proceedings of the IEEE international conference on computer vision*. 2017.

Related Works: Automatic Module Integration Using LLMs



ViperGPT: directly generate unrestricted Python code

VISPROG

- Generates a list of pseudocode instructions which needs further interpretation

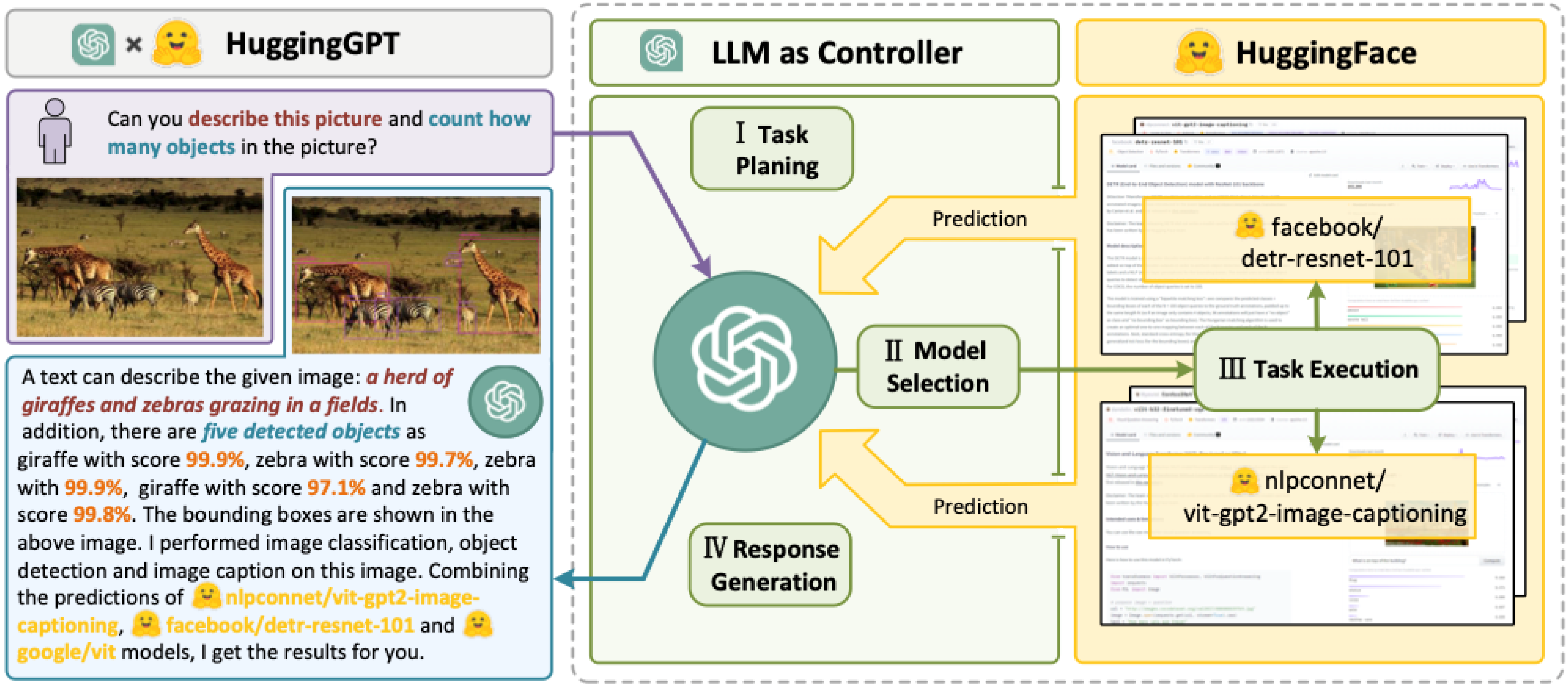
Very similar ideas. VISPROG was ~4 months earlier but went unnoticed due to CVPR publicity restrictions.

Gupta, Tanmay, and Aniruddha Kembhavi. "Visual programming: Compositional visual reasoning without training." *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 2023.

Georgia Tech

Related Works: HuggingGPT

- 4 step process:
  - Decompose request into tasks with dependencies
  - Assign models for each task
  - Execute tasks via HuggingFace
  - Synthesize information into one response
- Similar reasoning approach to ViperGPT, but with more tools
- Structured natural language instead of code

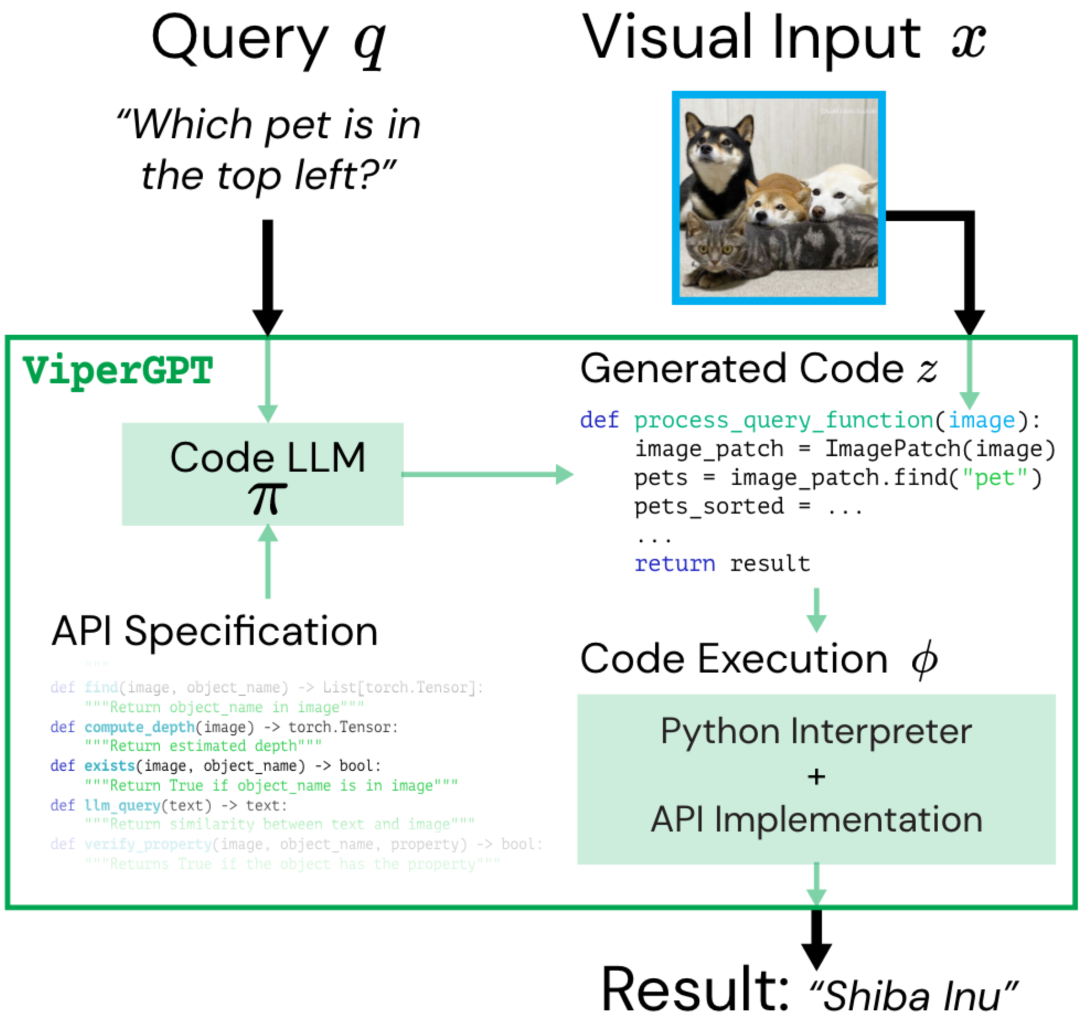


[{"task": "pose-detection", "id": 0, "dep": [-1], "args": {"image": "e3.jpg" }}, {"task": "pose-text-to-image", "id": 1, "dep": [0], "args": {"text": "a girl reading a book", "image": "<resource>-0" }}]

Yongliang Shen, Kaitao Song, Xu Tan, Dongsheng Li, Weiming Lu, and Yueting Zhuang. HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face. 2023. Available at: https://arxiv.org/abs/2303.17580.

Approach

Approach: Overview

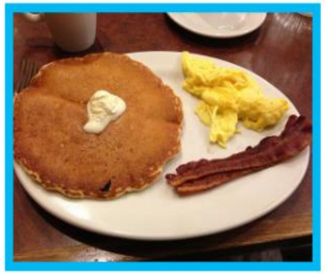


- ViperGPT is a framework for solving complex visual queries programmatically.
- Inputs
  - Visual input x: image / videos
  - Textual query q: questions or descriptions
- Output r: any type (e.g., text / image crops)
- Program generator π: $z = \pi(q)$
  - π: LLMs
  - z : Python code
- Execution engine ϕ: $r = \phi(x, z)$
  - Python Interpreter
  - API Implementation

Approach: Program Generation

Query: Does that pancake look brown and round?

In:



Generated code

```
def execute_command(image):
    image_patch = ImagePatch(image)
    pancake_patches = image_patch.find("pancake")
    is_brown = pancake_patches[0].verify_property("pancake", "brown")
    is_round = pancake_patches[0].verify_property("pancake", "round")
    return bool_to_yesno(is_brown and is_round)
```

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"
```

- Program Generator: GPT-3 Codex
  - Obviates the need for task-specific training for program generation.
- Input: a sequence of code text
  - Prompt: API specification
  - Query for the sample under consideration
- Output: Python function definition as a string.

Georgia Tech

Approach: Modules - ImagePatch

```
class ImagePatch:
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.
    Attributes
    ----------
    cropped_image : array_like
        An array-like of the cropped image taken from the original image.
    left : int
        An int describing the position of the left border of the crop's bounding box in the original image.
    lower : int
        An int describing the position of the bottom border of the crop's bounding box in the original image.
    right : int
        An int describing the position of the right border of the crop's bounding box in the original image.
    upper : int
        An int describing the position of the top border of the crop's bounding box in the original image.

    Methods
    -------
    find(object_name: str)->List[ImagePatch]
        Returns a list of new ImagePatch objects containing crops of the image centered around any objects found in the
        image matching the object_name.
    exists(object_name: str)->bool
        Returns True if the object specified by object_name is found in the image, and False otherwise.
    verify_property(property: str)->bool
        Returns True if the property is met, and False otherwise.
    best_text_match(option_list: List[str], prefix: str)->str
        Returns the string that best matches the image.
    simple_query(question: str=None)->str
        Returns the answer to a basic question asked about the image. If no question is provided, returns the answer
        to "What is this?".
    compute_depth()->float
        Returns the median depth of the image crop.
    crop(left: int, lower: int, right: int, upper: int)->ImagePatch
        Returns a new ImagePatch object containing a crop of the image at the given coordinates.
    """
```

- Each module is implemented as a class method.

Approach: Modules - VideoSegment

```
class VideoSegment:
    """A Python class containing a set of frames represented as ImagePatch objects, as well as relevant information.
    Attributes
    ----------
    video : torch.Tensor
        A tensor of the original video.
    start : int
        An int describing the starting frame in this video segment with respect to the original video.
    end : int
        An int describing the ending frame in this video segment with respect to the original video.
    num_frames->int
        An int containing the number of frames in the video segment.

    Methods
    -------
    frame_iterator->Iterator[ImagePatch]
    trim(start, end)->VideoSegment
        Returns a new VideoSegment containing a trimmed version of the original video at the [start, end] segment.
    select_answer(info, question, options)->str
        Returns the answer to the question given the options and additional information.
    """
```

- Each module is implemented as a class method.

Approach: API

```python
    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
```

```python
def llm_query(question: str) -> str:
    '''Answers a text question using GPT-3. The input question is always a formatted string with a variable in it.

    Parameters
    ----------
    question: str
        the text question to ask. Must not contain any reference to 'the image' or 'the photo', etc.
    '''
    return llm_query(question)
```

API specifies

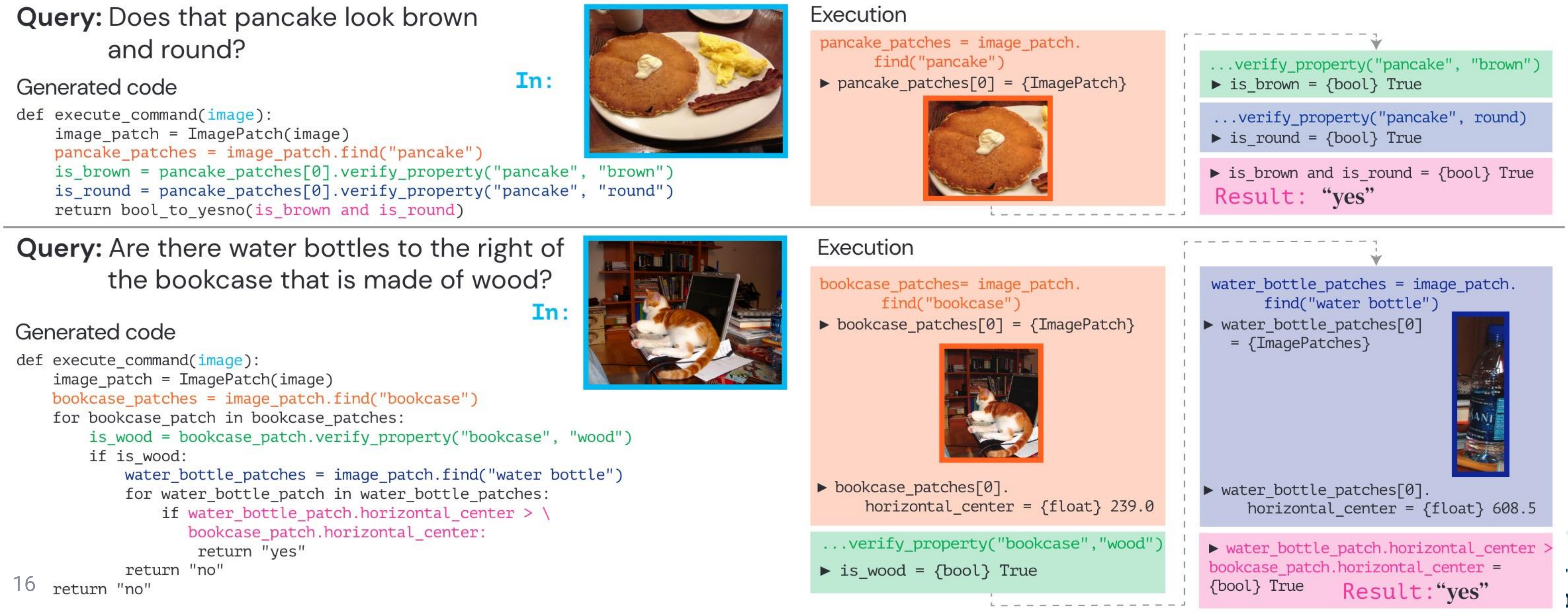
- Input and output types
- Docstrings to explain the purpose of these functions in natural language
- Examples that show how to use these classes and their functions (query-code pairs)

Only specifications, no full implement

- LLM context windows are limited
- Code generation is independent of changes made to the module implementation

Approach: Program Execution

- Python interpreter: logical operations
- Pretrained model APIs: perceptual operations



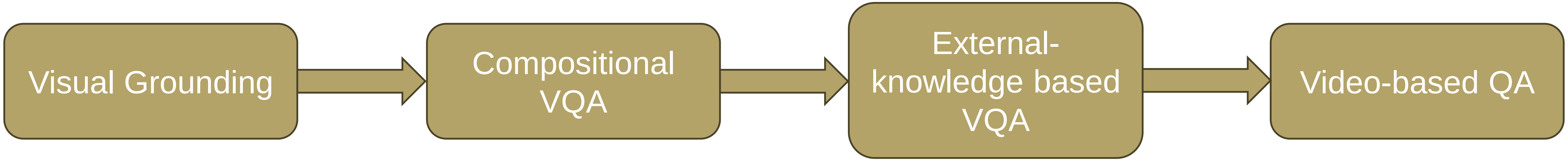
Approach: Program Execution

- Pretrained Models
  - GLIP: find, exists
  - MiDaS: compute_depth
  - BLIP-2: simple_query
  - X-VLM: verify_property, best_image_match, best_text_match
  - GPT-3: llm_query, select_answer
  - Codex: code generation

Evaluations and Results

Evaluations and Results

- Defines 4 main tasks ranging from basic understanding to complex synthesis
- Each task is a "prerequisite" for following task



Overview of Modules

- **find**: image and noun --> identifies patches containing noun
- **exists**: image and noun --> identifies if noun exists in image
- **verify_property**: image, noun, and property --> identifies if noun has property in image
- **best_image_match**: image patches and noun --> returns image patch matching noun
- **best_text_match**: list of nouns and image --> returns noun that matches image
- **compute_depth**: image patch --> median depth of patch
- **distance**: image patches --> distance between patches
- **simple_query**: short image/text questions that cannot be decomposed
- **llm_query**: queries requiring external knowledge
- **select_answer**: textual information about scene and possible answers --> best answer

Visual Grounding

- Requires spatial reasoning and object identification
- Modules provided:
  - **Find, exists, verify_property, best_image_match, compute_depth, distance**
- Evaluated on RefCOCO and RefCOCO+
- Takeaways:
  - Clearly outperforms zero-shot methods
  - Still far behind fine-tuned models
  - Expected result since this task focuses on visual understanding instead of reasoning

		IoU (%) ↑	
		RefCOCO	RefCOCO+
Sup.	MDETR [53]	90.4	85.5
	OFA [53]	94.0	91.7
ZS	OWL-ViT [38]	30.3	29.4
	GLIP [31]	55.0	52.2
	ReCLIP [49]	58.6	60.5
	ViperGPT (ours)	**72.0**	**67.0**

Compositional VQA

- Requires breaking down complex questions into simpler components
- Modules added:
  - **simple_query, best_text_match**
- Evaluated on GQA dataset
- Takeaways:
  - Slightly better than BLIP-2
  - Decently far behind all fine-tuned methods
  - Some emphasis on reasoning, but still largely focusing on spatial understanding

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**

Q: Are there any cups to the left of the tray on top of the table?
A: No

Drew A. Hudson and Christopher D. Manning. GQA: A New Dataset for Real-World Visua Reasoning and Compositional Question Answering. 2019. Available at: https://arxiv.org/abs/1902.09506.

External-Knowledge Based VQA

- Requires querying external knowledge to reason about the image
- Modules added:
  - **llm_query**
- Evaluated on OK-VQA dataset
- Takeaways:
  - Better than zero-shot and on-par with some fine-tuned models
  - Likely due to emphasis on reasoning & CoT
- **Query:** The real live version of this toy does what in the winter?

Generated code

```
def execute_command(image):
    image = ImagePatch(image)
    toy = image.simple_query("What is this toy?")
    result = llm_query("The real live version of
                {} does what in the winter?", toy)
    return result
```

In:

Execution

```
▶ toy = {str} "bear"

▶ guess = {str} "hibernate"
```

Result: "hibernate"

BLIP-2 result: "ski"

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**

Georgia Tech

Video-Based QA

- Requires causal and temporal reasoning about the video
- Modules added for this task:
  - **select_answer, VideoSegement**
- Evaluated on multiple choice NExT-QA dataset
- Takeaways:
  - Achieves SoTA results, surpassing fine-tuned methods
  - Underscores importance of reasoning in video understanding
  - Only has perception for images, but extrapolates to temporal and causal contexts using frames relations

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
Sup.	VGT [58]	-	-	56.9
Sup.	HiTeA [61]	48.6	47.8	63.1
ZS	ViperGPT (ours)	**49.8**	**56.4**	60.0

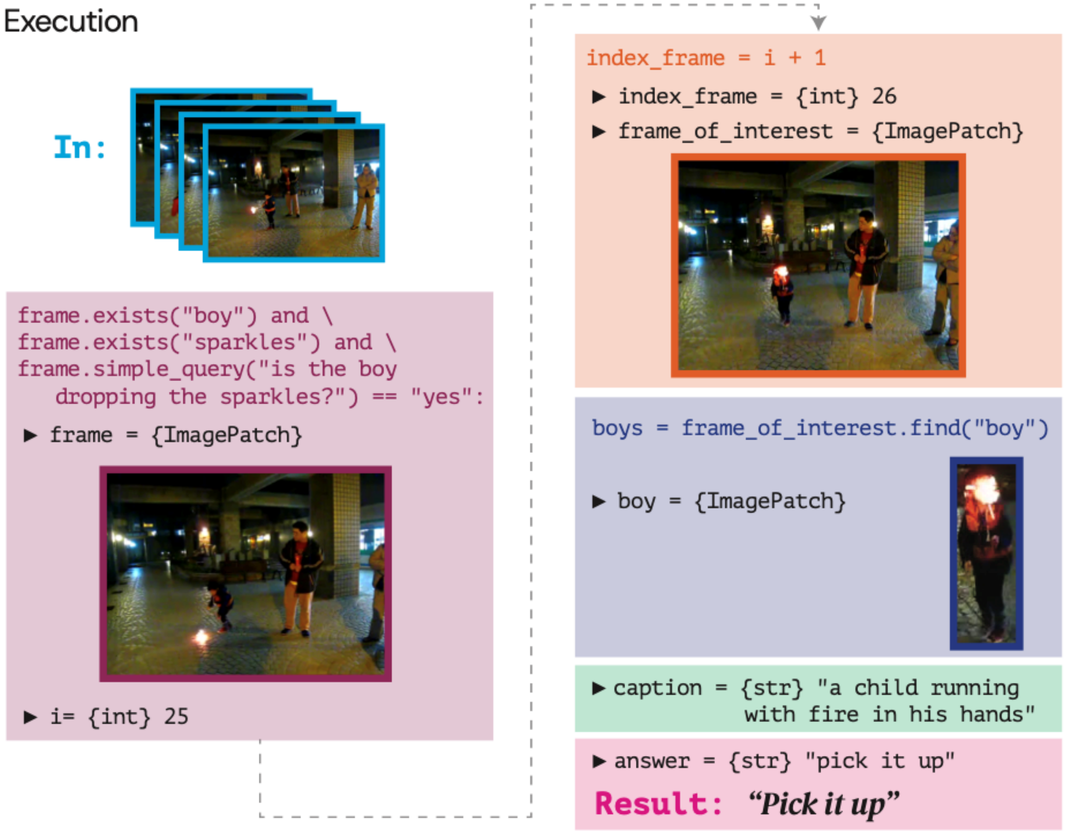
Video-Based QA

Query: What did the boy do after he dropped the sparkles on the floor?

Generated code

```
def execute_command(video, question, possible_answers):
    video_segment = VideoSegment(video)
    drop_detected = False
    for i, frame in enumerate(video_segment.frame_iterator()):
        if frame.exists("boy") and frame.exists("sparkles") and \
                frame.simple_query("is the boy dropping the sparkles?") == "yes":
            drop_detected = True
            break
    if drop_detected:
        index_frame = i + 1
    else:
        index_frame = video_segment.num_frames // 2
    frame_of_interest = ImagePatch(video_segment, index_frame)
    boys = frame_of_interest.find("boy")
    if len(boys) == 0:
        boys = [frame_of_interest]
    boy = boys[0]
    caption = boy.simple_query("What is this?")
    info = {
        "Caption of frame after dropping the sparkles": caption,
    }
    answer = select_answer(info, question, possible_answers)
    return answer
```

Execution

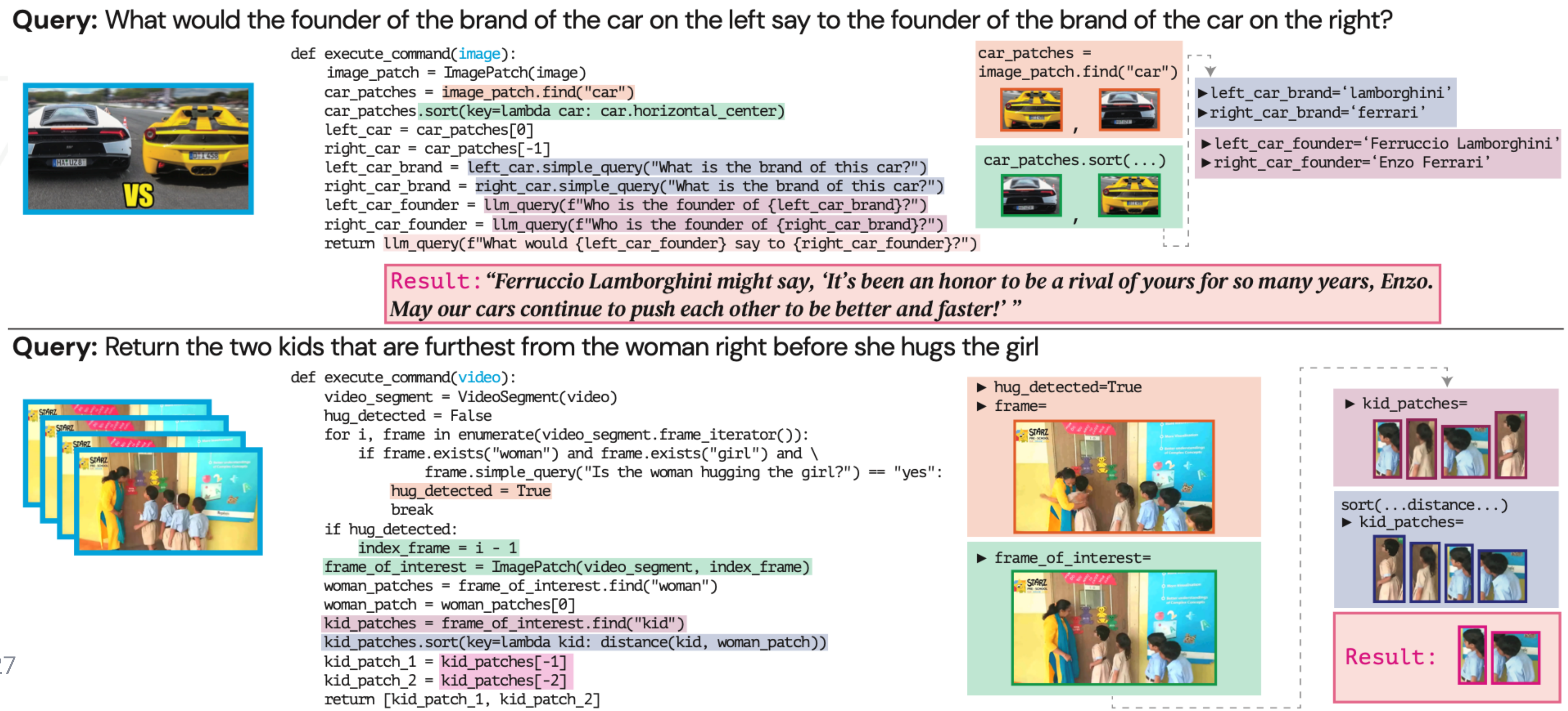


- Loop to identify when the sparkles are dropped
- Identify the image patch of the boy and determine the action he takes
- Use action to select correct answer

Discussion

Emergent Capabilities – Real-World Usage

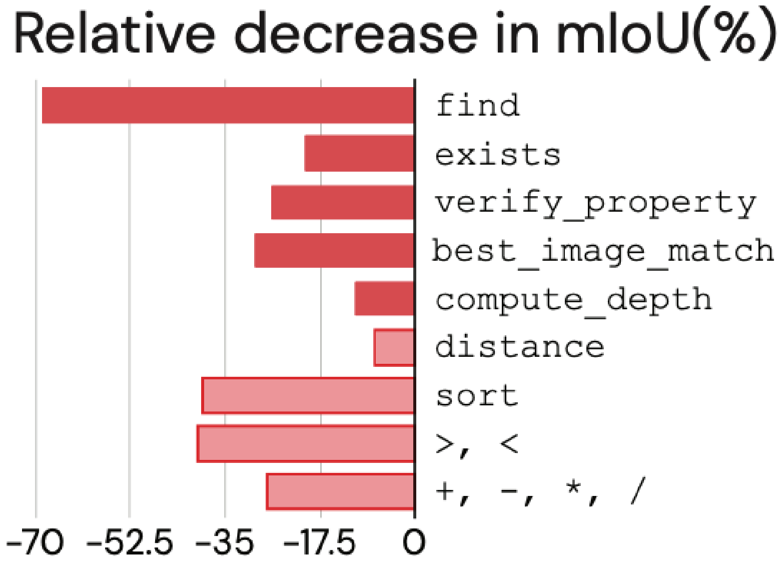
- Beyond benchmarks, Viper-GPT easily adapts to real-world queries
- Simple to add new modules with the provided framework



Emergent Capabilities - Intervention

- New method to evaluate importance of individual modules
  - Cannot evaluate intermediate output --> no ground truth data
  - Cannot compare accuracy between programs --> not all programs use the same modules
- Intervention: Substitute a module with a default value to measure performance drop with nonfunctional module
- Analysis performed on RefCOCO:

Figure 7. **Intervention.** We analyze the importance of various vision modules and Python functions in the generated programs as measured by the drop in mIoU when they are made nonfunctional.



Emergent Capabilities – Context-Aware Responses

- Program can be adapted based on context provided as comments
- Important when considering different cultures, norms, and expectations

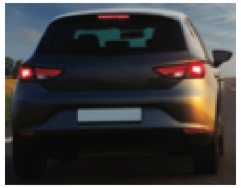
Query: Return the car that is on the correct lane

```
# Context: the picture was taken in the US
def execute_command(image):
    cars = image.find("car")
    for car in cars:
        if car.horizontal_center > image.horizontal_center:
            return car
    return None
```

Result: *None*

```
# Context: the picture was taken in the UK
def execute_command(image):
    cars = image.find("car")
    for car in cars:
        if car.horizontal_center < image.horizontal_center:
            return car
    return None
```

Result:



Georgia Tech

Limitations & Societal Implications

- Limitations:
  - Highly dependent on performance of pre-trained models—no ability to fine-tune for specific tasks
  - Produced code is interpretable, however not as simple as CoT which is easier for the public to understand
  - Programs generated can be overly complex or incorrect for complex tasks—difficult to find error without manual inspection

- Societal Implications
  - Enhances interpretability of VLM reasoning process, allowing for intermediate steps to be manually altered
  - Framework can be implemented at any scale with any models --> ease of access to more powerful VLM systems
  - Inherits biases of pre-trained models it uses (no inherent bias mitigation in the framework)

Georgia Tech

Summary of Strengths and Weaknesses

- Strengths:
  - Plug-and-Play system: can build modules with any models available
  - As better pre-trained models are created (GPT, BLIP, etc.) performance increases
  - Outputted programs are interpretable by humans for correction and general understanding
  - Achieves strong zero-shot results compared to other zero-shot models

- Weaknesses:
  - Performance is generally subpar compared to fine-tuned models
  - Not many zero-shot models available for comparison on some tasks (GQA and NExT-QA) and analysis is very shallow
  - Heavily dependent on capabilities of pre-trained models—areas that the pre-trained models struggle in will be reflected in the compositional model

Thank you!