

ViperGPT: Visual Inference via Python Execution for Reasoning

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

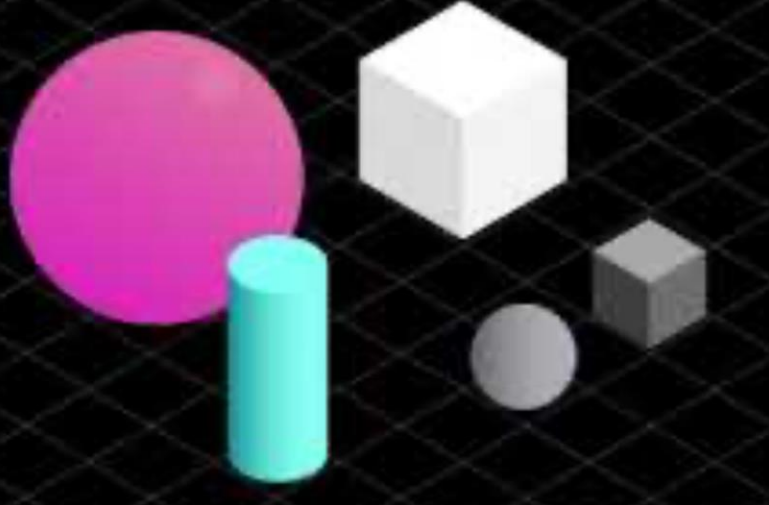
Presented by: Seulgi Kim, Hyunjeong Kim

Outline

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

Background: Neuro-symbolic reasoning

then rely on symbolic AI programs that apply logic and semantic reasoning to identify relationships among these objects.

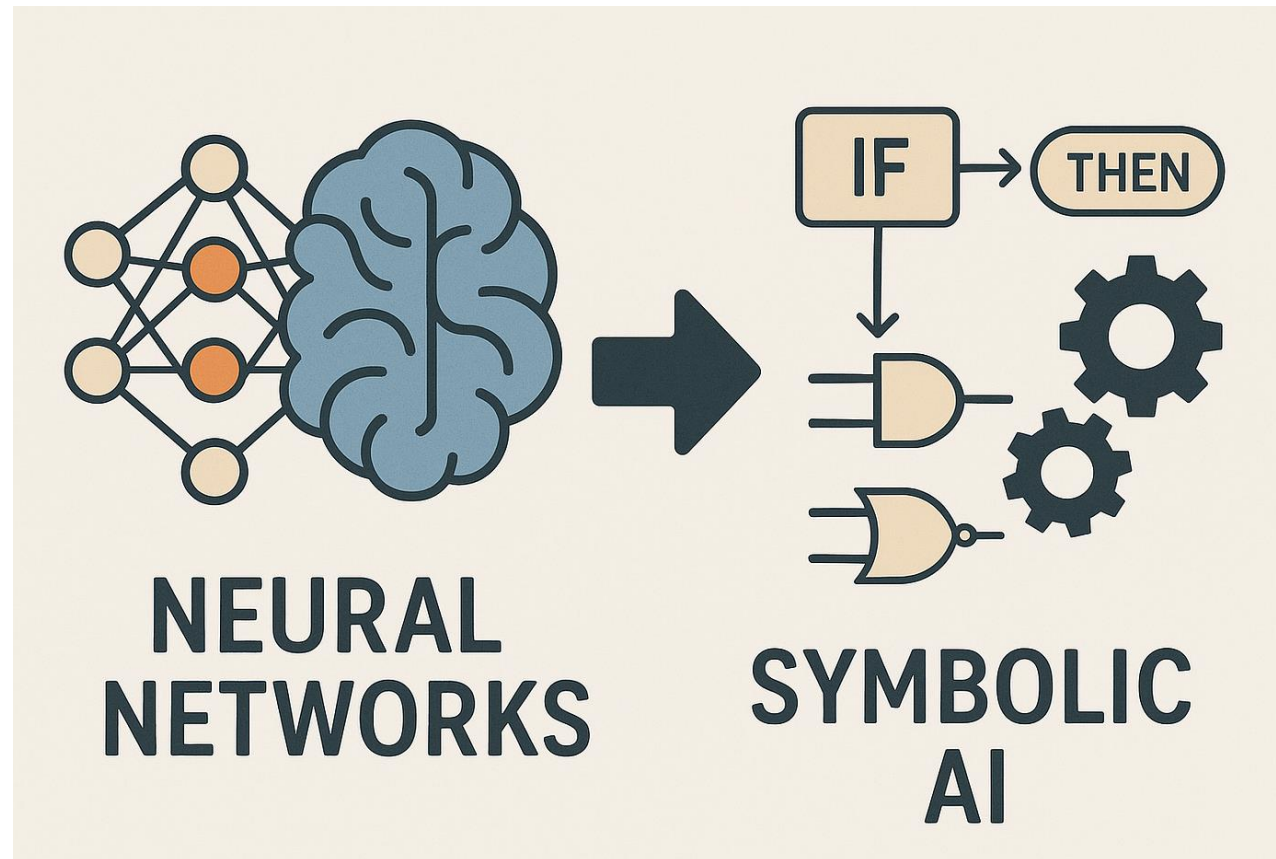


Q: Is the cylinder in front of the pink sphere?

Color	Cyan	Pink
Shape	Cylinder	Sphere
Position	5	2

Background: Neuro-symbolic reasoning

Neuro-symbolic reasoning is a composition of pattern recognition (neural networks) + analytic thinking (symbolic AI).



Problem Statement

Task: Answering visual queries

Query: How many muffins can each kid have for it to be fair?



1. Recognize muffins



2. Recognize Kids



3. Count visible objects

```
► len(muffin_patches)=8  
► len(kid_patches)=2
```

4. Reasoning

► What does “*fair*” mean?

5. Mathematical Operation

```
►  $8 // 2 = 4$ 
```

Result: 4

Problem Statement

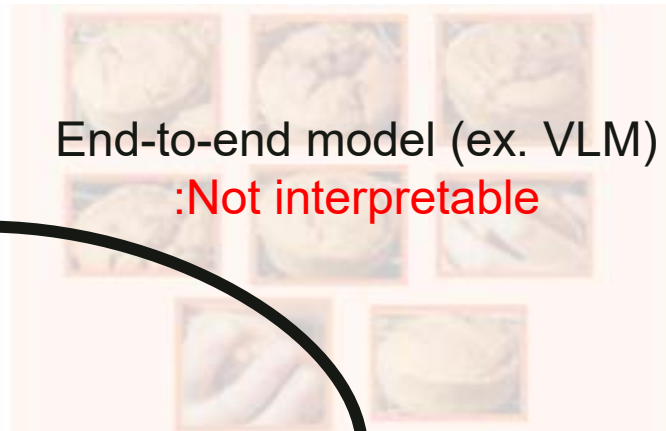
Existing methods approach this task with end-to-end training.

Query: How many muffins can each kid have for it to be fair?



Tons of Dataset

1. Recognize muffins



End-to-end model (ex. VLM)
:Not interpretable

2. Recognize kids



3. Count visible objects

```
► len(muffin_patches)=8  
► len(kid_patches)=2
```

4. Reasoning

► What does “fair” mean?

5. Mathematical Operation

```
► 8 // 2 = 4
```

Result: 4

Ground Truth

Problem Statement

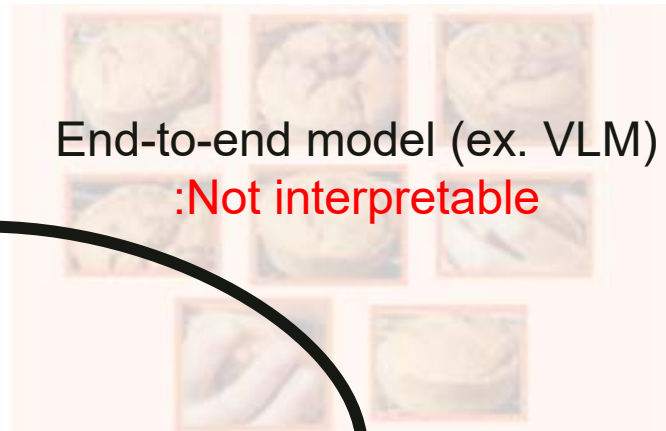
In existing methods, language models need to solve mathematical operations.

Query: How many muffins can each kid have for it to be fair?



Tons of Dataset

1. Recognize muffins



End-to-end model (ex. VLM)
:Not interpretable

2. Recognize kids



3. Count visible objects

```
► len(muffin_patches)=8  
► len(kid_patches)=2
```

4. Reasoning

► What does "fair" mean?

5. Mathematical Operation

```
► 8//2 = 4
```

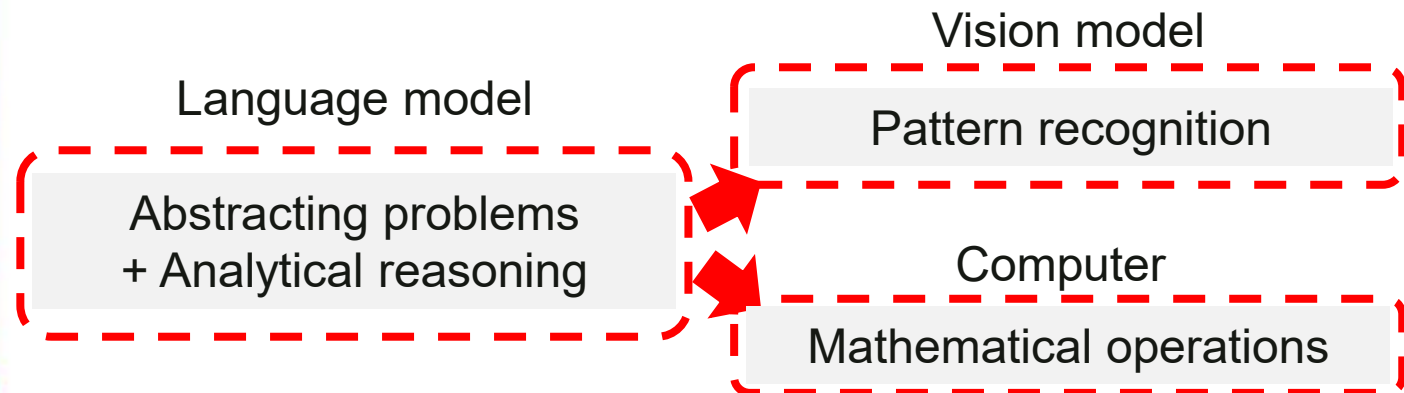
Result: 4

Ground Truth

Computers
can do this
easily!

Problem Statement

- Language model excels at analytical reasoning
- Computer excels at mathematical operations



Problem Statement

Proposed method: compositional approach



Language model

Abstracting problems
+ Analytical reasoning

Step-by-step thinking

1. Recognize muffins

2. Recognize Kids

3. Count visible objects

4. Reasoning

5. Mathematical Operation

Vision model

Pattern recognition



Computer

Mathematical operations

```
► len(muffin_patches)=8
```

```
► len(kid_patches)=2
```

```
► 8//2 = 4
```

Problem Statement

Proposed method: compositional approach

Language model

Abstracting problems
+ implement pseudo code

Generated Code

```
def execute_command(image):  
    image_patch = ImagePatch(image)  
    muffin_patches = image_patch.find("muffin")  
    kid_patches = image_patch.find("kid")  
    return str(len(muffin_patches) // len(kid_patches))
```

Vision model

Pattern recognition



Computer

Mathematical operations

► `len(muffin_patches)=8`
► `len(kid_patches)=2`

► $8 // 2 = 4$

Contribution

- **Compositional**
 - Process Visual tasks, Reasoning tasks, and mathematical operation tasks separately.
- **Interpretability**
 - All the steps are explicit as a code, with intermediate values.
- **Flexibility**
 - Incorporate any vision or language module.
 - Only require changing API specification as a prompt.
- **Training—free**
- **Generalizability**
 - Existing methods are highly domain-limited.
 - ViperGPT can process any tasks (domain-free).

Related Works

PoT: Program of Thoughts Prompting¹

- In PoT, LLM is used to **translate natural language problems into python programs** as intermediate reasoning steps.
- ViperGPT extends this ability to solve **complex, multimodal problems** (object recognition, depth estimation, ..).

Question: Ketty saves 20000 dollars to the bank. After three years, the sum with compound interest rate is 1000 dollars more than the sum with simple interest rate. What is the interest rate of the bank?

Assuming the interest rate is x . The sum after two years with simple interest rate is $20000 + x * 20000 * 3 = 20000 + 60000x$. The sum after two years with compound interest rate is $20000 * (1 + x)^3 = 200000 + 60000 * x + 60000x^2 + 20000x^3$. The difference can be written as $60000x^2 + 20000x^3 = 1000$. In order to solve x , we can use the quadratic formula. $x = (-b \pm \sqrt{b^2 - 4ac}) / 2a$, ..., $x = (-20000 \pm 6160) / 120000$, $x = -0.051333$.

CoT

↓
-0.051333



```
interest_rate = Symbol('x')
sum_in_two_years_with_simple_interest= 20000 +
interest_rate * 20000 * 3
sum_in_two_years_with_compound_interest = 20000 * (1 +
interest_rate)**3
# Since compound interest is 1000 more than simple interest.
ans = solve(sum_after_in_yeras_with_compound_interest -
sum_after_two_years_in_compound_interest - 1000,
interest_rate)
```

PoT

python SymPy ↓
 $x = 0.24814$



Related Works

PAL: Program-aided language models²

- PAL and PoT were simultaneously published (3 days difference) with almost same ideas.
- PAL generalizes to more tasks (13 benchmarks) with prompt-agnostic way.
- ViperGPT extends this ability to solve **complex, multimodal problems** (object recognition, depth estimation, ..).

Chain-of-Thought (Wei et al., 2022)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls. 2 cans of 3 tennis balls each is 6 tennis balls. $5 + 6 = 11$. The answer is 11.

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves. They sold 93 in the morning and 39 in the afternoon. So they sold $93 + 39 = 132$ loaves. The grocery store returned 6 loaves. So they had $200 - 132 - 6 = 62$ loaves left. The answer is 62.



Program-aided Language models (this work)

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 tennis balls.
`tennis_balls = 5`
2 cans of 3 tennis balls each is
`bought_balls = 2 * 3`
tennis balls. The answer is
`answer = tennis_balls + bought_balls`

Q: The bakers at the Beverly Hills Bakery baked 200 loaves of bread on Monday morning. They sold 93 loaves in the morning and 39 loaves in the afternoon. A grocery store returned 6 unsold loaves. How many loaves of bread did they have left?

Model Output

A: The bakers started with 200 loaves
`loaves_baked = 200`
They sold 93 in the morning and 39 in the afternoon
`loaves_sold_morning = 93`
`loaves_sold_afternoon = 39`
The grocery store returned 6 loaves.
`loaves_returned = 6`
The answer is
`answer = loaves_baked - loaves_sold_morning`
`- loaves_sold_afternoon + loaves_returned`

`>>> print(answer)`
74



Related Works

Faithful Chain of thought Reasoning³

- Faithful CoT provides rationale for each line of code.
- Similar approach with PAL/Pot, but Faithful CoT focuses more on **interpretability** by decomposing the problems in more structured way.
- Similarly, ViperGPT also focuses on interpretability by presenting intermediate values.

Would an apple sink in water?

Translator
(LM)

```
// 1. What is the density of an apple?  
// The density of an apple is about 0.75 g/cm^3.  
// 2. What is the density of water?  
// Water has a density of 1 g/cm^3.  
// Then, we represent these answers in Datalog:  
// 1. The density of an apple is about 0.75 g/cm^3.  
.decl Has_density(Object:symbol, Density:float)  
Has_density("apple", 0.75).  
// 2. Water has a density of 1 g/cm^3.  
Has_density("water", 1).  
// Now, we derive the final answer: Would an apple sink in water?  
// The answer is Yes only if an apple is more dense than water.  
.decl Answer()  
Answer() :- Has_density("apple", density1), Has_density("water",  
density2), density1 > density2.  
.output Answer
```

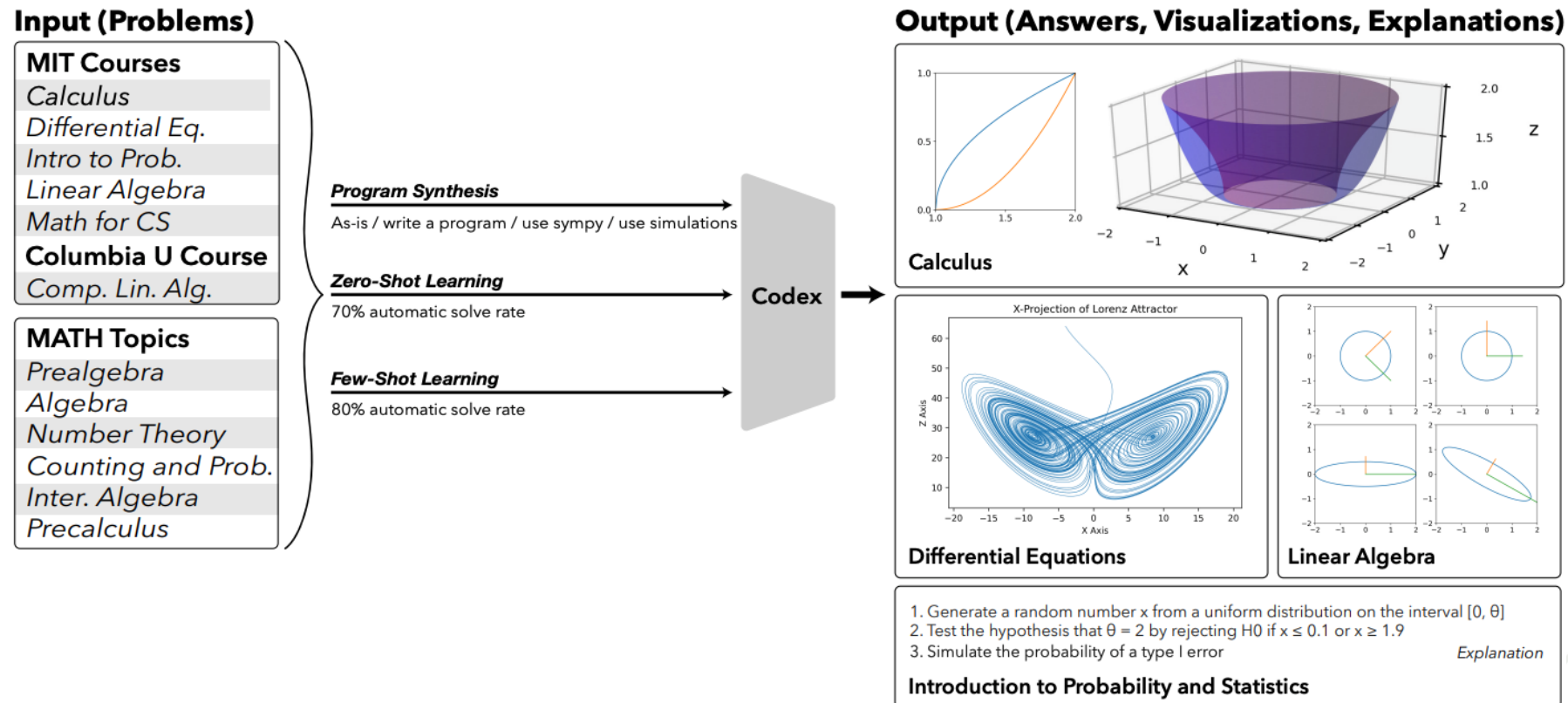
Deterministic
Solver

No

Related Works

Neural network solves, explains, and generates university math problems⁴

- First shows that **OpenAI's Codex** can solve complicated tasks such as MIT course problems.
- ViperGPT also utilizes Codex to implement actual code.



Related Works

VISPROG: Visual Programming – Compositional visual reasoning without training⁵

- VISPROG and ViperGPT were 4 months difference with almost similar ideas.
- Unlike VISPROG that generates pseudocode that needs further interpretation, ViperGPT **directly generates executable python code**.

Natural Language Visual Reasoning

LEFT:



RIGHT:



Statement: The left and right image contains a total of six people and two boats.

Program:

```
ANSWER0=Vqa(image=LEFT, question='How many people are in the image?')
ANSWER1=Vqa(image=RIGHT, question='How many people are in the image?')
ANSWER2=Vqa(image=LEFT, question='How many boats are in the image?')
ANSWER3=Vqa(image=RIGHT, question='How many boats are in the image?')
ANSWER4=Eval('{ANSWER0} + {ANSWER1} == 6 and {ANSWER2} + {ANSWER3} == 2')
RESULT=ANSWER4
```

Prediction: False

Compositional Visual Question Answering

IMAGE:



Question: Are there both ties and glasses in the picture?

Program:

```
BOX0=Loc(image=IMAGE, object='ties')
ANSWER0=Count(box=BOX0)
BOX1=Loc(image=IMAGE, object='glasses')
ANSWER1=Count(box=BOX1)
ANSWER2=Eval("'yes' if {ANSWER0} > 0 and {ANSWER1} > 0 else 'no'")
RESULT=ANSWER2
```

Prediction: no

Approach

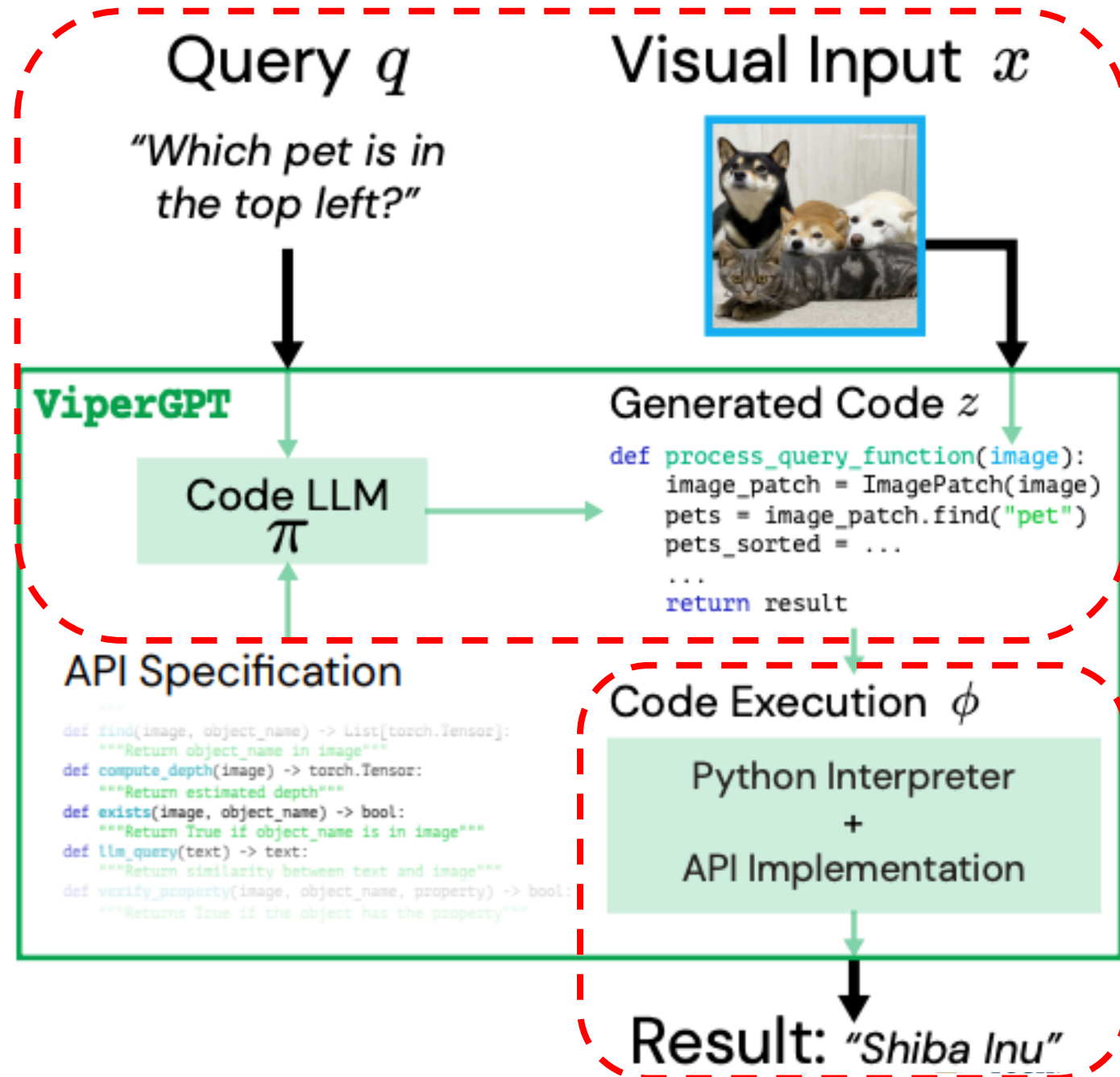
Flowchart of ViperGPT

- Input: Textual query q , Visual x
- Code LLM: $\pi(q, API) = z$
 - π = LLM
 - API = API specification
 - z = Generated code
- Code Execution: $\phi(x, z) = r$
 - ϕ = Execution engine
 - r = Result

where,

- q = question, description
- x = RGB, video, depth
- r = text, image crops,... (any type)

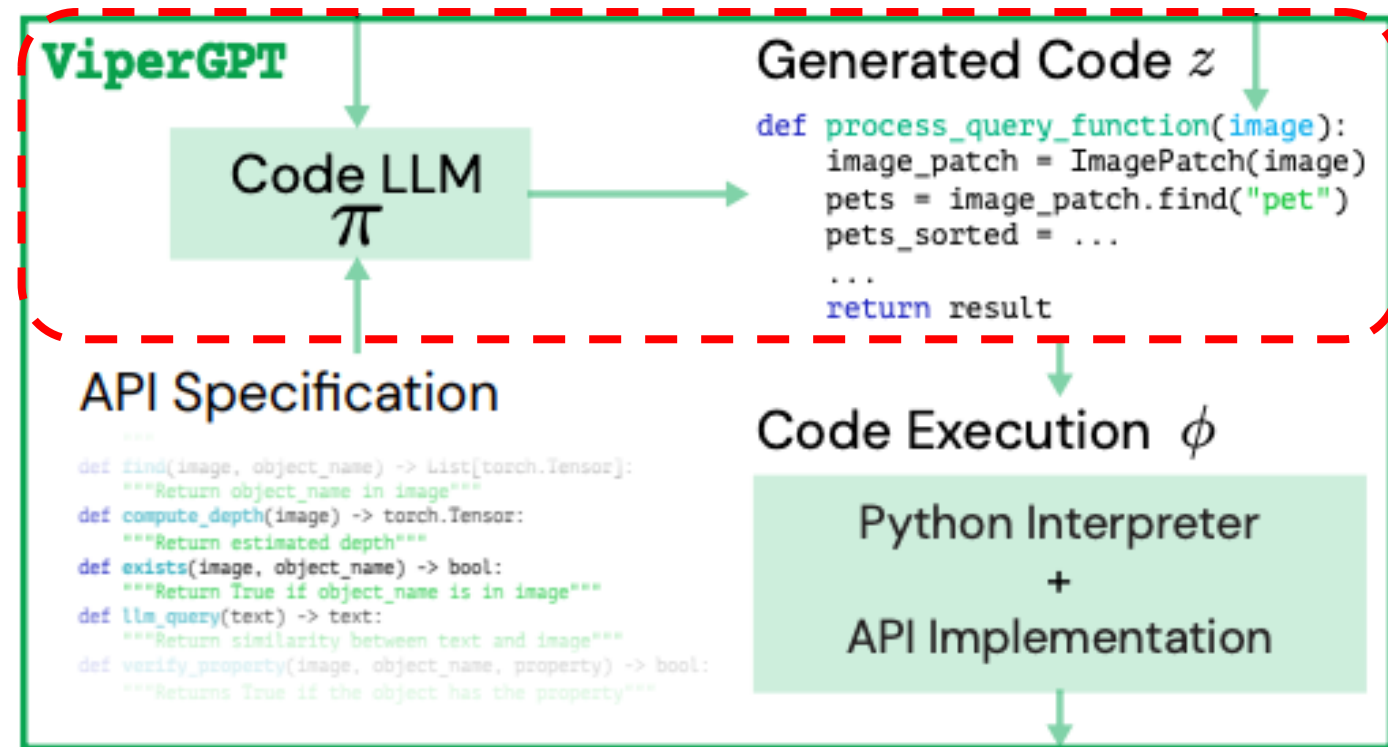
17/68



Approach

Program Generator π

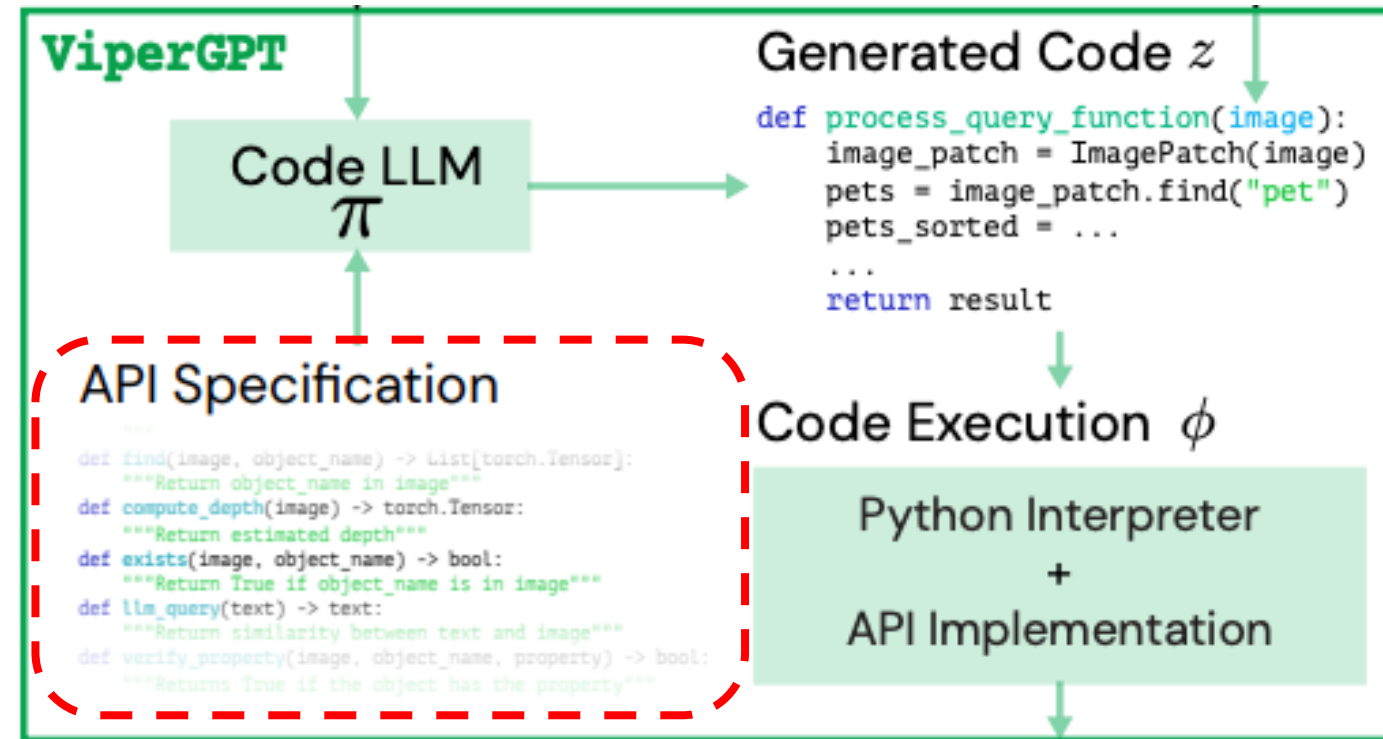
- Used GPT-3 Codex for program generator π .
- Why Codex?
 - The generated code should work.
 - Codex is specifically trained on code data.



Approach

Prompt to the Program Generator π

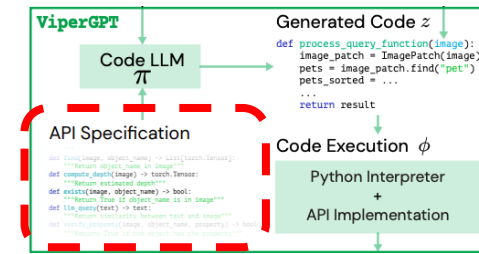
- Feeds API specification as a prompt.
- Composed with 2 global classes
 1. class ImagePatch
 2. class VideoSegment



Approach

Prompt to the Program Generator π : class ImagePatch

- ImagePatch.find() method find object in the given image.
- Internally calls GLIP pretrained model in .find() method.



Query: How many muffins can each kid have for it to be fair?



Generated Code

```
def execute_command(image):  
    image_patch = ImagePatch(image)  
    muffin_patches = image_patch.find("muffin")  
    kid_patches = image_patch.find("kid")  
    return str(len(muffin_patches) // len(kid_patches))
```

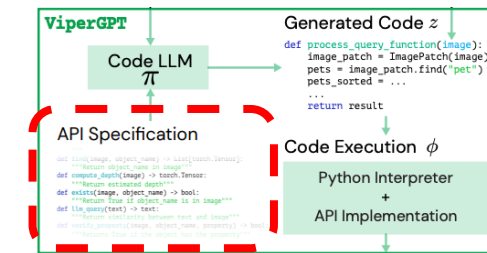


```
class ImagePatch:  
    """A Python class containing a crop of an image centered around a particular object, as well as relevant information.  
    Attributes  
    -----  
    cropped_image : array_like  
        An array-like of the cropped image taken from the original image.  
    left : int  
        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.  
    """  
  
    def __init__(self, image, left: int=None, lower: int=None, right: int=None, upper: int=None):  
        """Initializes an ImagePatch object by cropping the image at the given coordinates and stores the coordinates as attributes.
```

Approach

Prompt to the Program Generator π : class ImagePatch

- Specified Input and Output types
- Provided Docstring to explain the purpose of each method
- Specified Example code
- Exact code (x), Natural Language (o)



```
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
    >>> """
```

VipergPT .find() method specification

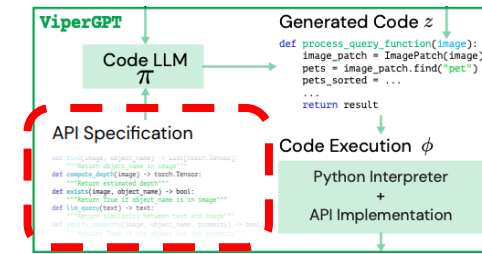
The screenshot shows a terminal window with the title 'seulgikim — less ◀ man find — 77x42'. The content is the 'find' command manual page, which includes sections for NAME, SYNOPSIS, and DESCRIPTION. The NAME section states 'find - walk a file hierarchy'. The SYNOPSIS section shows the command syntax: 'find [-H | -L | -P] [-EXdsx] [-f path] path ... [expression]' and 'find [-H | -L | -P] [-EXdsx] -f path [path ...] [expression]'. The DESCRIPTION section explains that the 'find' utility recursively descends the directory tree for each path listed, evaluating an expression (composed of the "primaries" and "operands" listed below) in terms of each file in the tree. It also mentions that the options are as follows.

Similar to how human search methods in linux man page

Approach

Prompt to the Program Generator π : class ImagePatch

- ImagePatch.compute_depth() method returns depth of the center of the given image patch.
- Internally calls MiDaS pretrained model in .compute_depth() method.

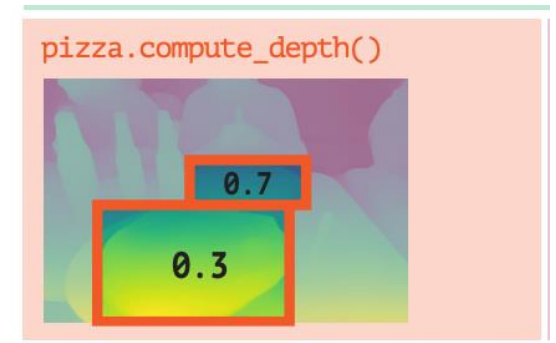


Query: pizza front

Generated code

```
def execute_command(image):  
    image_patch = ImagePatch(image)  
    pizza_patches = image_patch.find("pizza")  
    pizza_patches.sort(key=lambda pizza: pizza.compute_depth())  
    patch_return = pizza_patches[0]  
    return patch_return
```

In:



Approach

Prompt to the Program Generator π : class VideoSegment

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

In:

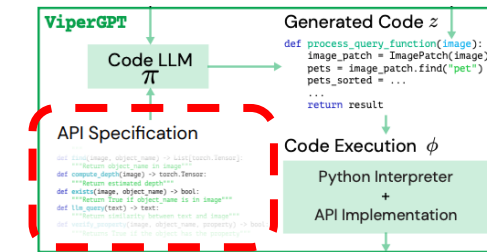


```
class VideoSegment:
    """A Python class containing a set of frames represented as ImagePatch objects, as well as relevant information.
    Attributes
    -----
    video : torch.Tensor
        A tensor of the original video.
    start : int
        An int describing the starting frame in this video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

    def __init__(self, video: torch.Tensor, start: int = None, end: int = None, parent_start=0, queues=None):
        """Initializes a VideoSegment object by trimming the video at the given [start, end] times and stores the
        start and end times as attributes. If no times are provided, the video is left unmodified, and the times are
        set to the beginning and end of the video.

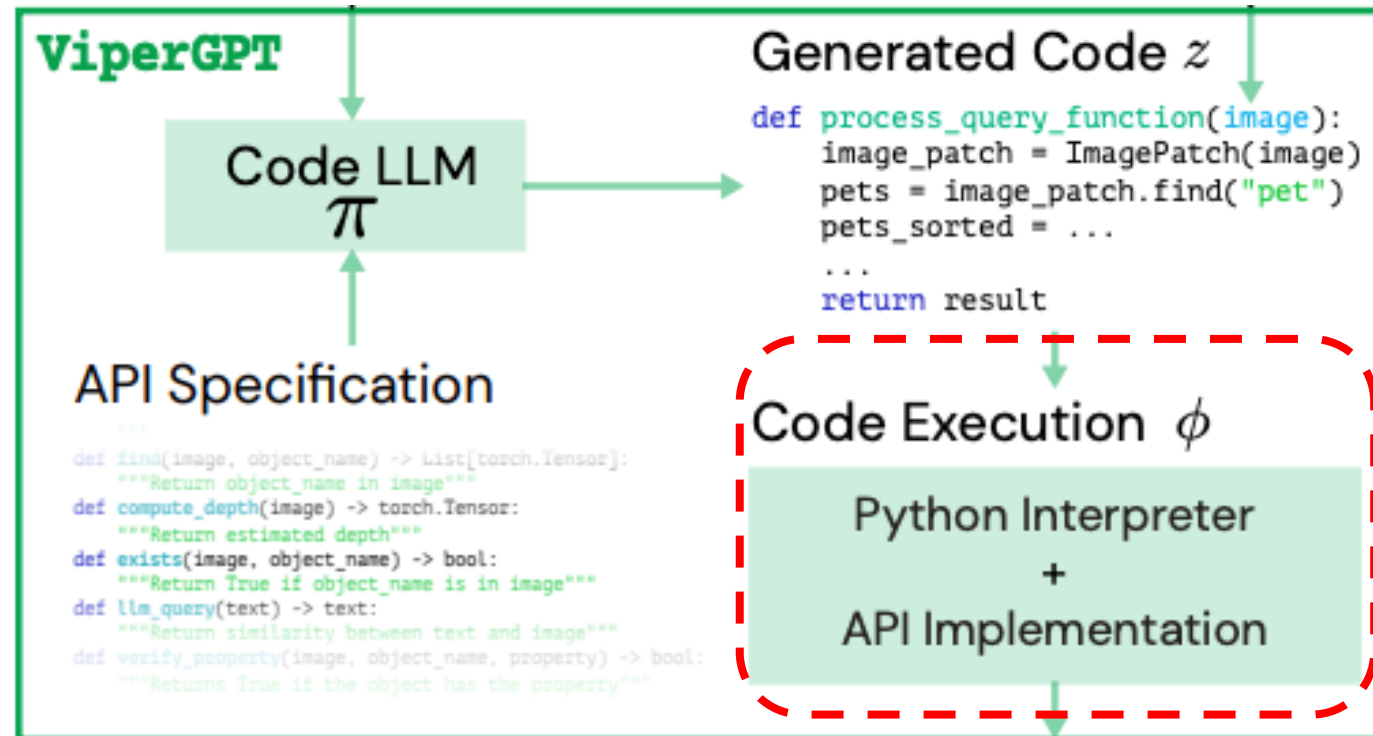
        Parameters
        -----
        video : torch.Tensor
            A tensor of the original video.
        start : int
```



Approach

Program Executor ϕ

- Python interpreter executes the generated code z .
- Certain perceptual function (ex. Depth estimation, object detection,..) is processed by pretrained models.



Experiments

Experiment design

ViperGPT evaluates its efficacy with 4 different setups:

1. Visual Grounding
2. Compositional Image-Question Answering
3. External knowledge dependent image-question answering
4. Video causal and temporal reasoning

Experiments and Results

1. Visual Grounding

- Evaluates spatial understanding and visual understanding.
- Outperforms zero-shot models, but way worse than supervised models.

Query: pizza front

Generated code

```
def execute_command(image):  
    image_patch = ImagePatch(image)  
    pizza_patches = image_patch.find("pizza")  
    pizza_patches.sort(key=lambda pizza: pizza.compute_depth())  
    patch_return = pizza_patches[0]  
    return patch_return
```

In:

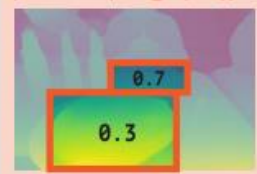


Execution

```
pizza_patches = image_patch.find("pizza")  
► pizza_patches= {List[ImagePatch]}
```



pizza.compute_depth()



pizza_patches.sort()



```
patch_return = pizza_patches[0]  
return patch_return
```

Result:



Table 1. **RefCOCO Results.** We report accuracy on the REC task and testA split. ZS=zero shot, Sup.=supervised.

		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

Figure 3. **Visual grounding on RefCOCO.**

Experiments and Results

2. Compositional Image Question Answering

- Evaluates how well the model breaks down complex questions into simpler ones.
- Slightly better than zero-shot models, but worse than supervised models.

		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

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"
```

Execution

```
bookcase_patches= image_patch.
    find("bookcase")
▶ bookcase_patches[0] = {ImagePatch}

▶ bookcase_patches[0].
    horizontal_center = {float} 239.0

...verify_property("bookcase","wood")
▶ is_wood = {bool} True
```



```
water_bottle_patches = image_patch.
    find("water bottle")
▶ water_bottle_patches[0]
    = {ImagePatches}

▶ water_bottle_patches[0].
    horizontal_center = {float} 608.5

▶ water_bottle_patch.horizontal_center >
bookcase_patch.horizontal_center =
{bool} True
Result: "yes"
```



Figure 4. Compositional image question answering on GQA.

Figure from: Surís, D., Menon, S., & Vondrick, C. (2023). Vipergpt: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 11888-11898).

Experiments and Results

3. External Knowledge-dependent Image Question Answering

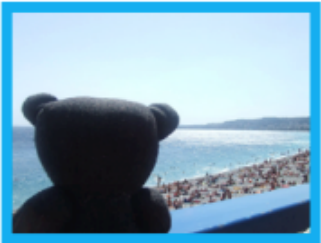
- Evaluates how well the model query external knowledge to solve problem.
- Better than zero-shot models, also similar as supervised models.

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]
ZS	54.5
	REVIVE [33]
	58.0
	PromptCap [21]
	58.8
ZS	PNP-VQA [52]
	35.9
	PICa [60]
	43.3
	BLIP-2 [30]
ZS	45.9
	Flamingo [1]
	50.6
ZS	ViperGPT (ours)
	51.9

Figure 5. Programmatic chain-of-thought with external knowledge for OK-VQA.

Experiments and Results

4. Video Causal/Temporal Reasoning

- Evaluates how well the model extrapolates to reasoning in video datasets.
- Better than supervised models (SOTA) despite seeing no video data.

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
NS	ViperGPT (ours)	49.8	56.4	60.0

Query: How does the black dog position himself at the end?

Generated code

```
def execute_command(video, question, possible_answers):
    video_segment = VideoSegment(video)
    last_frame = ImagePatch(video_segment, -1)
    last_caption = last_frame.simple_query("What is this?")
    dogs = last_frame.find("dog")
    if len(dogs) == 0:
        dogs = [last_frame]
    dog = dogs[0]
    dog_action = dog.simple_query("What is the dog doing?")
    info = {
        "Caption of last frame": last_caption,
        "Dog looks like he is doing": dog_action
    }
    answer = select_answer(info, question, possible_answers)
    return answer
```

29/68

Execution

In:



last_frame =
ImagePatch(video_segment, -1)
▶ last_frame = {ImagePatch}

▶ last_caption = {str} "a black dog sitting in the grass"

dogs = last_frame.find("dog")
▶ dog = {ImagePatch}

dog_action = dog.simple_query("What is the dog doing?")
▶ dog_action= {str} "sitting"

▶ answer = {str} "sit on the ground"

Result: "Sit on the ground"

Figure 6. Temporal reasoning on NeXT-QA.

Figure from: Surís, D., Menon, S., & Vondrick, C. (2023). ViperGPT: Visual inference via python execution for reasoning. In *Proceedings of the IEEE/CVF international conference on computer vision* (pp. 11888-11898).

Discussion

Why it works?

- Performs better when requires more reasoning.
- Existing chain-of-thought rely on LLM to perform both reasoning and computation.
- ViperGPT excels at programmatically composing specialized vision, language, math, and logic functions as subroutines.

Evaluation	Requires reasoning?	Result
1. Visual Grounding	★ Only requires visual understanding	★ Way worse than supervised models.
2. Compositional Image Question Answering	★★ Reasoning based on the given image	★★ Worse than supervised models.
3. External Knowledge-dependent Image Question Answering	★★★ Reasoning outside of the given image	★★★ Similar to supervised models.
4. Video Causal/Temporal Reasoning	★★★★★ Requires extrapolation of prior knowledge	★★★★★ State-of-the-art

More results

Think different based on contexts

- ViperGPT effectively produces different logic for each case (US vs. UK)

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:



Figure 8. **Contextual programs.** ViperGPT readily incorporates additional context into the logic of the generated programs.

Limitations & Societal Implications

Limitations

- Problem solving stage is interpretable, but we still do not know how the LLM generates the code.
- Potential dangers: What if the code includes “*os.rmdir()*”?
- Highly dependent on the performance of Codex and pretrained large models.

Societal Implications

- Improve trustworthiness of AI system.
- Can apply to real world problem by using generalization ability.

Summary of Strengths, Weaknesses, Relationships

Strength

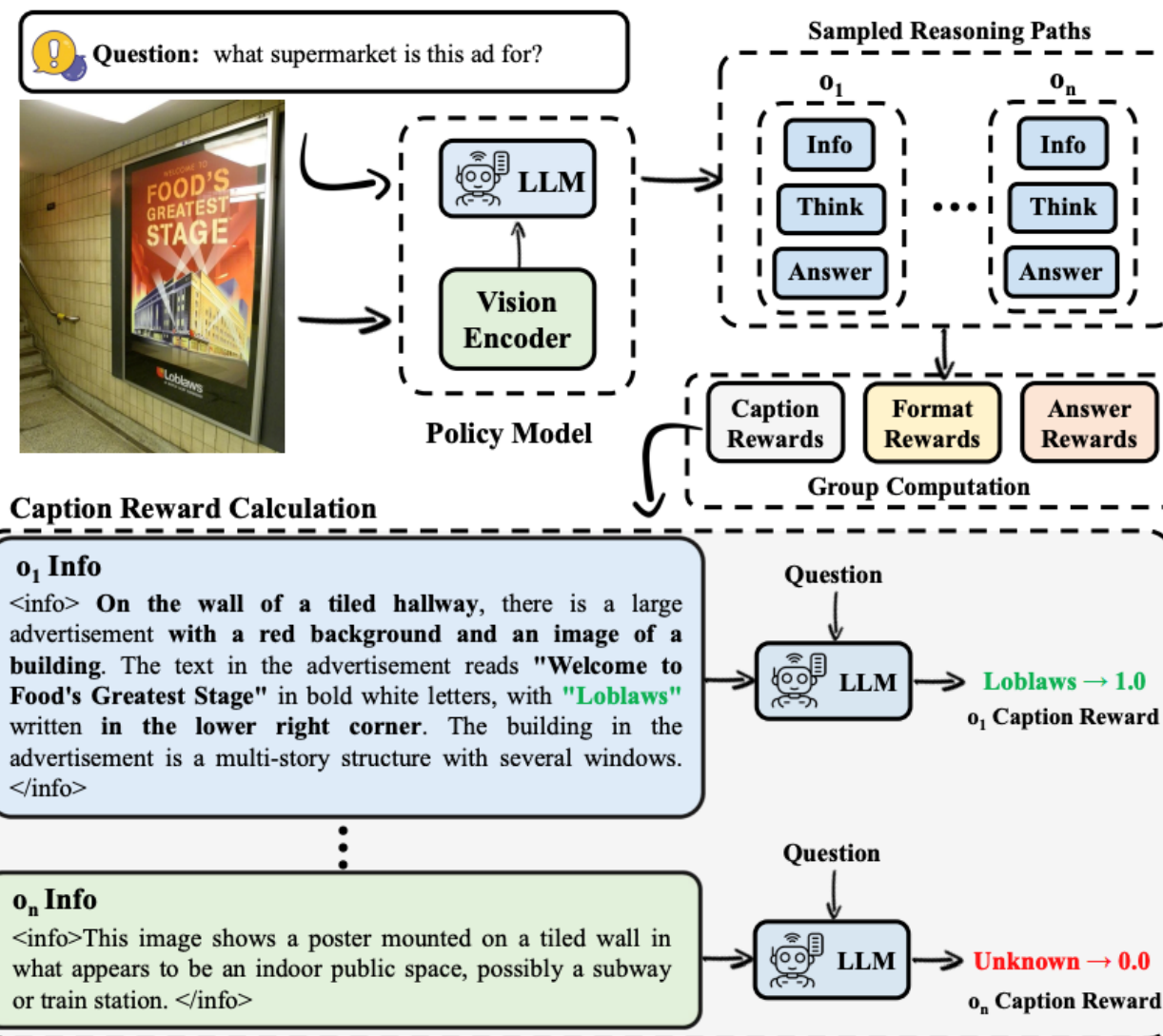
- Does not have a 'shortcut problem' because of line-by-line code execution.

Query: How many muffins can each kid have for it to be fair?



Generated Code

```
def execute_command(image):  
    image_patch = ImagePatch(image)  
    muffin_patches = image_patch.find("muffin")  
    kid_patches = image_patch.find("kid")  
    return str(len(muffin_patches) // len(kid_patches))
```



Summary of Strengths, Weaknesses, Relationships

Strength

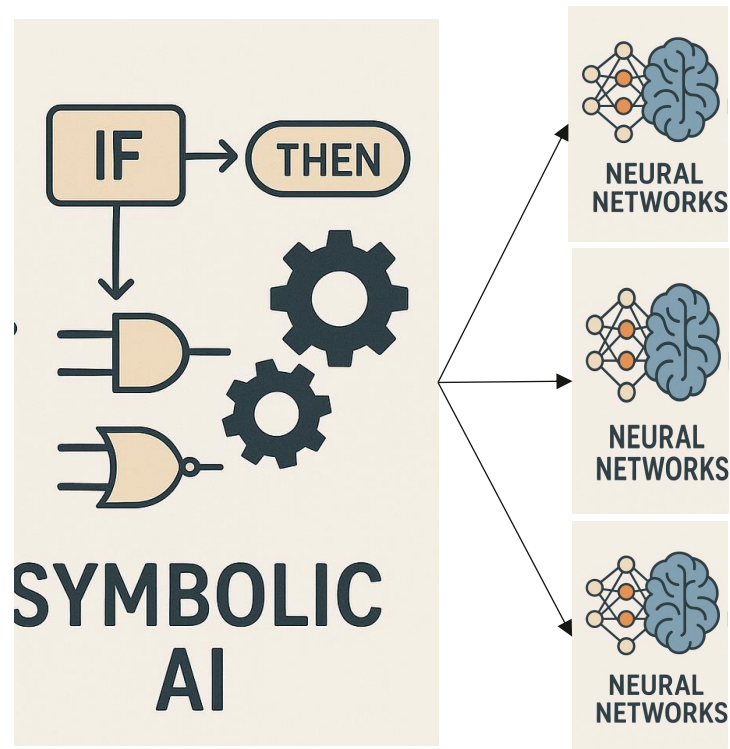
- Does not have 'shortcut problem' because of line-by-line code execution.
- ViperGPT enables vision and language to show capabilities beyond what any individual model can do on its own.
- As the pretrained models continue to improve, ViperGPT's results will also continue to improve in tandem.

Weaknesses

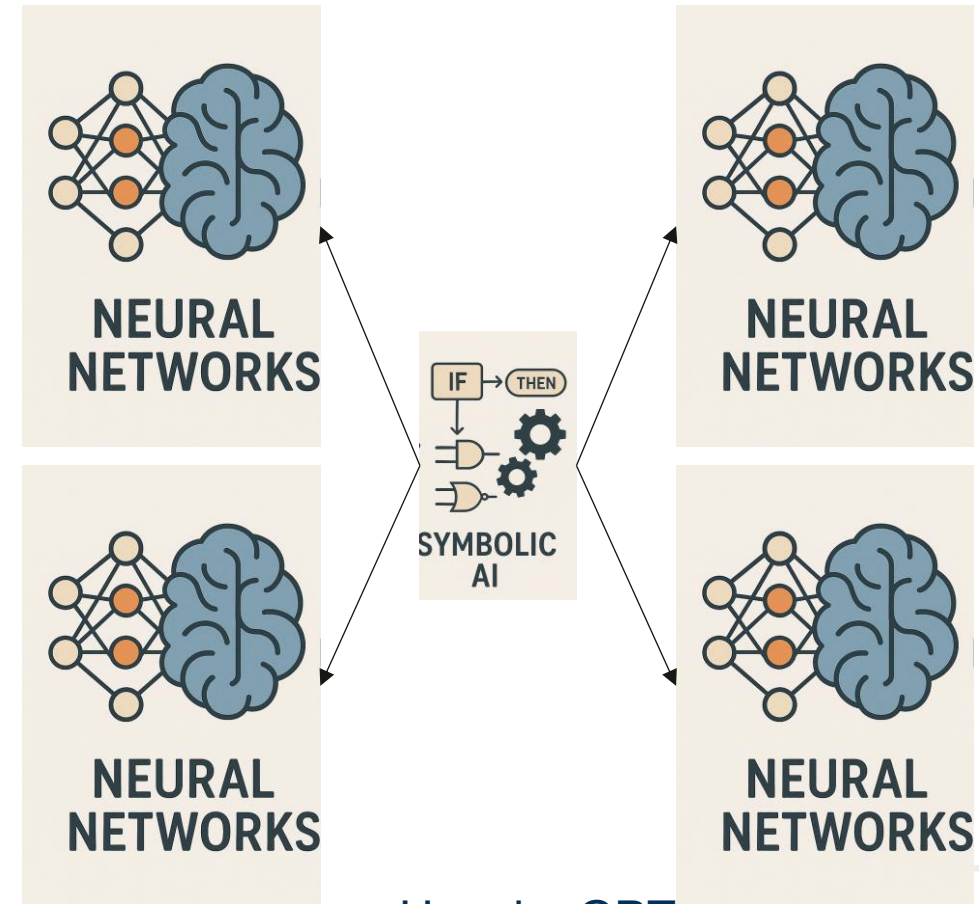
- Not enough evaluation metrics; only evaluated with accuracy.
- Not enough comparisons; for NExT-QA, only evaluated with two models.
- Performance is generally worse or slightly better in some cases when compared with supervised models.

Neuro-symbolic reasoning: ViperGPT vs. HuggingGPT

- ViperGPT focused more on **symbolic** reasoning.
- HuggingGPT covered massive scope with **more** tools and expert **neural networks**.



ViperGPT



HuggingGPT

HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face

**Yongliang Shen, Kaitao Song, Xu Tan, et al.
NeurIPS 2023**

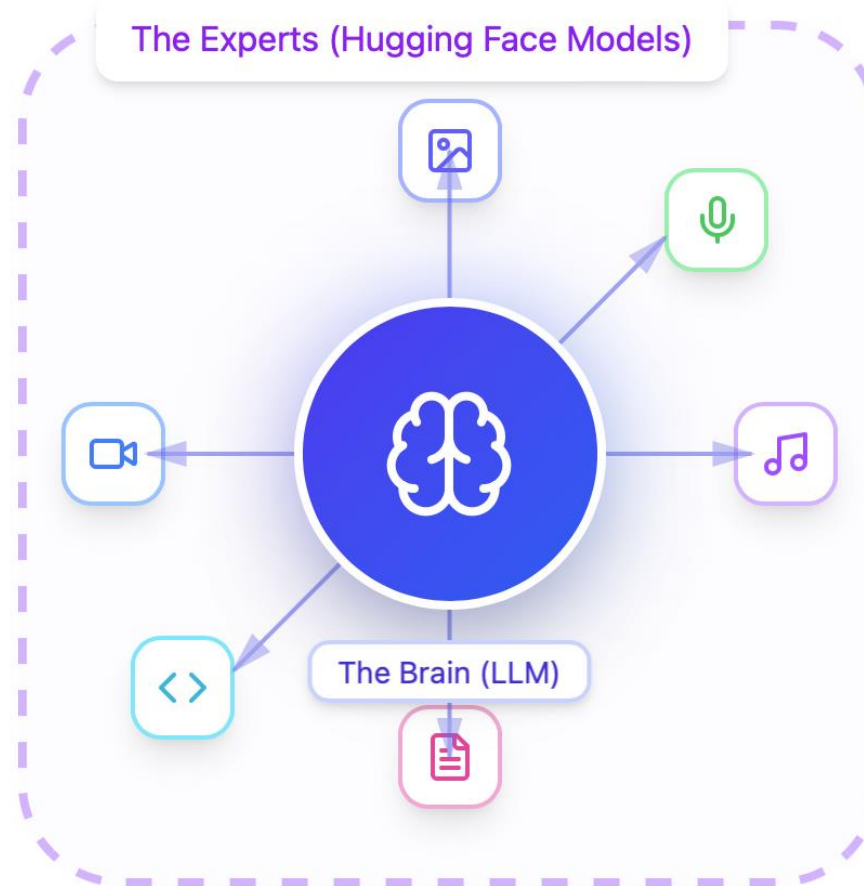
The HuggingGPT Concept

Core Idea

Use a Large Language Model (LLM) as an intelligent controller to orchestrate and manage specialized AI models from Hugging Face, combining their strengths to solve complex tasks.

Key Insight

Instead of one model doing everything, let the LLM act as "the brain" that understands tasks and delegates work to expert models—each specialized in specific domains like vision, audio, or text.



How HuggingGPT Works

Task Planning

- The LLM analyzes the user's complex request (e.g., “make an image and describe it”).
- It breaks the request down into a logical list of solvable sub-tasks (e.g., pose-detection, pose-to-image, image-to-text, text-to-speech).

Model Selection

- The LLM consults the “model cards” (descriptions) on Hugging Face.
- It assigns the best expert model for each specific sub-task.

Task Execution

- The selected expert models are called and run their individual tasks.
- The results (e.g., an image file, text, or an audio file) are sent back to the LLM.

Response Generation

- The LLM collects and integrates all the results from the expert models.
- It generates a final, detailed, and human-like response for the user.

Key Capabilities & Impact

- **Goes Beyond Text:** Enables the LLM to "see," "hear," and "create" by using specialized models, solving complex multi-modal tasks.
- **Automatic Planning:** Autonomously generates complex plans and coordinates multiple AI models to fulfill a single, high-level user request.
- **Combines Strengths:** Leverages the LLM's powerful reasoning and planning abilities with the high-accuracy performance of specialized "expert" models.
- **Continuously Scalable:** The system's capabilities can automatically grow and improve as new expert models are added to the Hugging Face community.

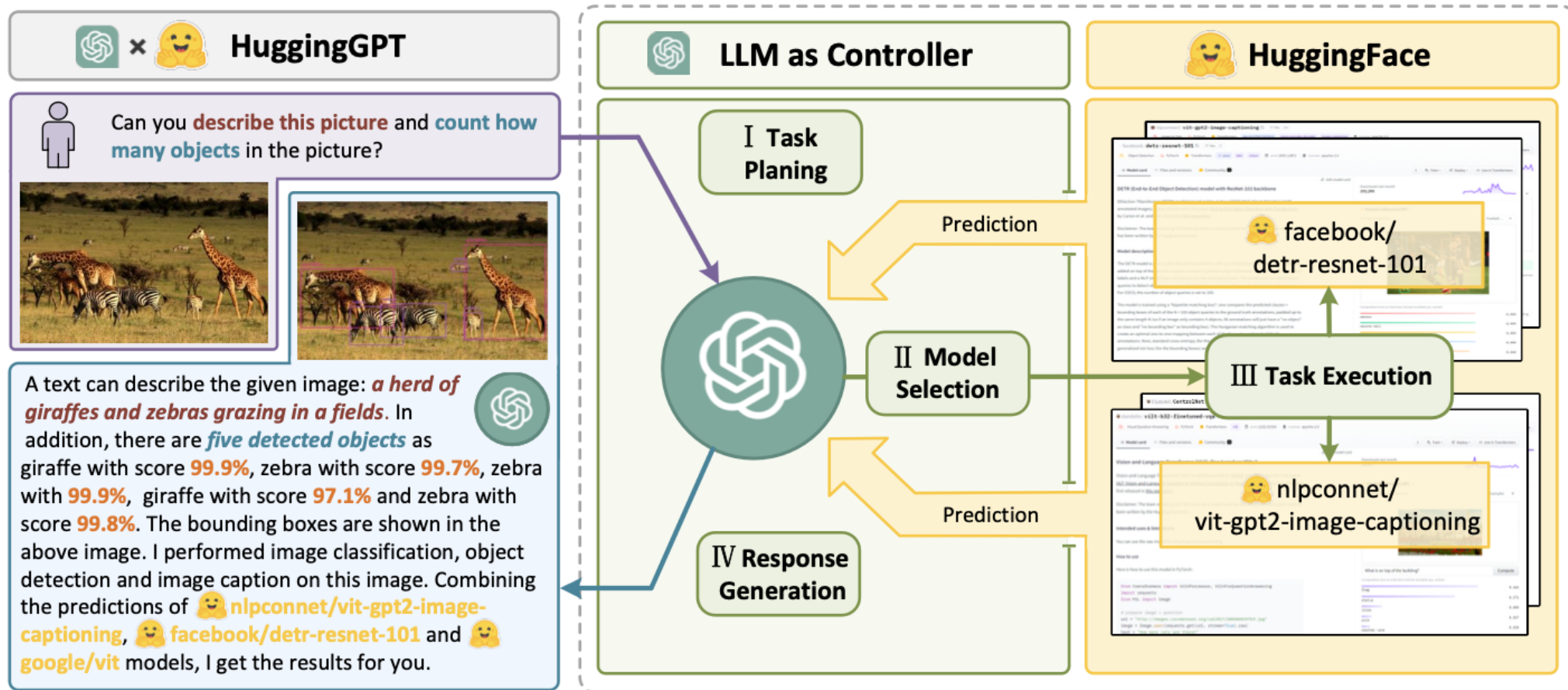


Figure 1: *Language serves as an interface for LLMs (e.g., ChatGPT) to connect numerous AI models (e.g., those in Hugging Face) for solving complicated AI tasks. In this concept, an LLM acts as a controller, managing and organizing the cooperation of expert models. The LLM first plans a list of tasks based on the user request and then assigns expert models to each task. After the experts execute the tasks, the LLM collects the results and responds to the user.*

Contribution

Proposed HuggingGPT Fram

- The paper introduces a novel system, HuggingGPT, that leverages a Large Language Model (LLM) as a central “brain” for planning and decision-making.

Inter-Model Cooperation Protocol

- The paper details a new protocol that enables the LLM to automatically invoke, coordinate, and execute numerous expert models from the Hugging Face hub to solve specific sub-tasks.

Generalized, Multi-Modal Task Solving

- This system provides a new solution for tackling complex AI tasks that span multiple modalities and domains (e.g., language, vision, speech, and cross-modality).

Evaluation of LLM Planning

- The paper highlights the importance of task planning and model selection and formulates new experimental evaluations to measure the capability of LLMs in these areas.

Demonstrated Capability

- The paper validates the system's potential through extensive experiments on challenging, multi-step AI tasks, demonstrating a new path toward general AI solutions.

Related Work

Branch 1: Unified End-to-End Multimodal Models

- Models like Flamingo, BLIP-2, and Kosmos-1 build a *single, large system* that directly combines vision and language capabilities.

Branch 2: LLMs Integrated with Tools

- Models like Toolformer, Visual ChatGPT, and ViperGPT teach an LLM to *use external tools* or APIs, often by generating code to call them.

How HuggingGPT is Different

- **LLM as a "Controller," not just a user:** HuggingGPT uses the LLM to *manage and route* requests to many different "expert" models, acting as a high-level planner.
- **General and Flexible:** It's not limited to specific tools. By reading model descriptions from Hugging Face, it can plan tasks and use *any* expert model for *any* modality (vision, speech, etc.).
- **Open and Scalable:** New expert models from the AI community can be integrated easily just by adding their description. This allows the system to continuously grow smarter without being rebuilt.

Approach

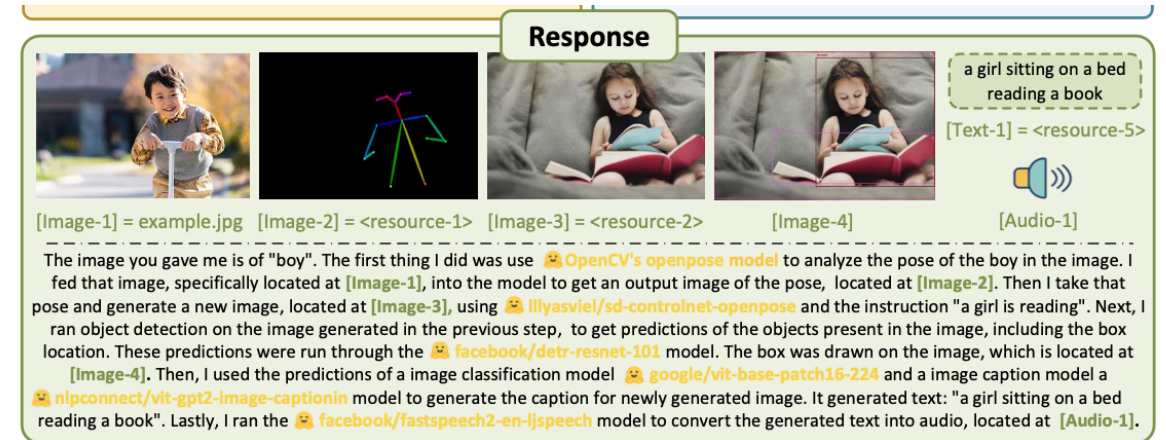
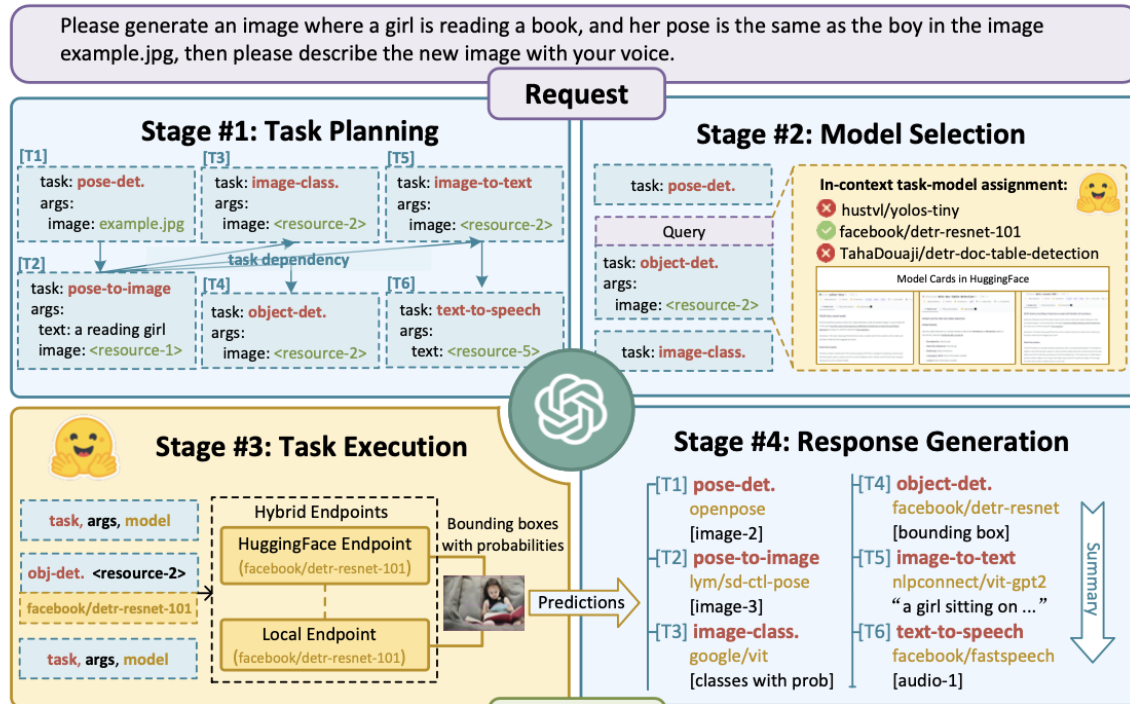


Figure 2: Overview of HuggingGPT. With an LLM (e.g., ChatGPT) as the core controller and the expert models as the executors, the workflow of HuggingGPT consists of four stages: 1) **Task planning**: LLM parses the user request into a task list and determines the execution order and resource dependencies among tasks; 2) **Model selection**: LLM assigns appropriate models to tasks based on the description of expert models on Hugging Face; 3) **Task execution**: Expert models on hybrid endpoints execute the assigned tasks; 4) **Response generation**: LLM integrates the inference results of experts and generates a summary of workflow logs to respond to the user.

3.1 Task Planning

Method 1

Specification-based Instruction

The LLM is required to follow a strict template for its output.

This JSON plan includes four main slots:

- task** the job to do
- id** a unique number
- dep** which tasks must finish first
- args** the inputs for the task

Method 2

Demonstration-based Parsing

The prompt includes several examples (demonstrations).

These examples pair user requests with their correct, final JSON task plans.

This helps "teach" the LLM the correct logic and format.

- **Main Goal:** To analyze a complex user request and break it down into a structured, step-by-step plan in JSON format.
- **Key Feature: Dependencies:** This plan determines the *execution order* and *dependencies* for each task (e.g., Task 2 can't start until Task 1, which creates a needed image, is finished).
- **Supports Multi-Turn Chat:** The system can also include the {{ Chat Logs }} in the prompt, allowing it to remember and use resources (like images or text) from earlier in the conversation.

3.1 Task Planning

Task Planning	Prompt	
	#1 Task Planning Stage - The AI assistant performs task parsing on user input, generating a list of tasks with the following format: [{"task": task, "id", task_id, "dep": dependency_task_ids, "args": {"text": text, "image": URL, "audio": URL, "video": URL}}]. The "dep" field denotes the id of the previous task which generates a new resource upon which the current task relies. The tag "<resource>-task_id" represents the generated text, image, audio, or video from the dependency task with the corresponding task_id. The task must be selected from the following options: {{ Available Task List }}. Please note that there exists a logical connections and order between the tasks. In case the user input cannot be parsed, an empty JSON response should be provided. Here are several cases for your reference: {{ Demonstrations }}. To assist with task planning, the chat history is available as {{ Chat Logs }}, where you can trace the user-mentioned resources and incorporate them into the task planning stage.	
	Demonstrations	
	Can you tell me how many objects in e1.jpg? In e2.jpg, what's the animal and what's it doing? First generate a HED image of e3.jpg, then based on the HED image and a text "a girl reading a book", create a new image as a response.	<pre>[{"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "e1.jpg" }}] [{"task": "image-to-text", "id": 0, "dep": [-1], "args": {"image": "e2.jpg" }}, {"task": "image-cls", "id": 1, "dep": [-1], "args": {"image": "e2.jpg" }}, {"task": "object-detection", "id": 2, "dep": [-1], "args": {"image": "e2.jpg" }}, {"task": "visual-quesrion-answering", "id": 3, "dep": [-1], "args": {"text": "what's the animal doing?", "image": "e2.jpg" }}] [{"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" }}]</pre>

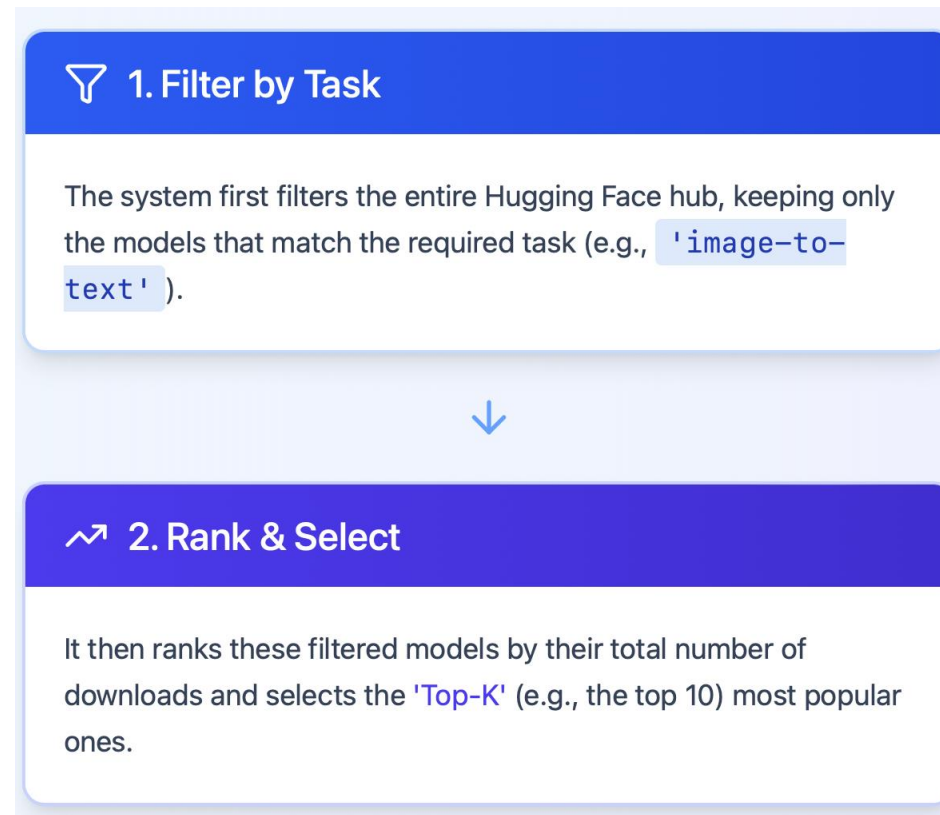
Task	Args	Candidate Models	Descriptions
NLP Tasks			
Text-CLS	text	[cardiffnlp/twitter-roberta-base-sentiment, ...]	["This is a RoBERTa-base model trained on 58M tweets ...", ...]
Token-CLS	text	[dslim/bert-base-NER, ...]	["bert-base-NER is a fine-tuned BERT model that is ready to...", ...]
Text2text-Generation	text	[google/flan-t5-xl, ...]	["If you already know T5, FLAN-T5 is just better at everything...", ...]
Summarization	text	[bart-large-cnn, ...]	["BART model pre-trained on English language, and fine-tuned...", ...]
Translation	text	[t5-base, ...]	["With T5, we propose reframing all NLP tasks into a unified...", ...]
Question-Answering	text	[deepset/roberta-base-squad2, ...]	["This is the roberta-base model, fine-tuned using the SQuAD2.0...", ...]
Conversation	text	[PygmalionAI/pygmalion-6b, ...]	["Pymalion 6B is a proof-of-concept dialogue model based on...", ...]
Text-Generation	text	[gpt2, ...]	["Pretrained model on English ...", ...]
Tabular-CLS	text	[matth/flowformer, ...]	["Automatic detection of blast cells in ALL data using transformers...", ...]

Audio Tasks			
Text-to-Speech	text	[espnet/kan-bayashi_ljspeech_vits, ...]	["his model was trained by kan-bayashi using ljspeech/tts1 recipe in...", ...]
Audio-CLS	audio	[TalTechNLP/voxlingua107-epaca-tdnn, ...]	["This is a spoken language recognition model trained on the...", ...]
ASR	audio	[jonatasgrosman/wav2vec2-large-xlsr-53-english, ...]	["Fine-tuned XLSR-53 large model for speech recognition in English ...", ...]
Audio-to-Audio	audio	[speechbrain/metricgan-plus-voicebank, ...]	["MetricGAN-trained model for Enhancement...", ...]
Video Tasks			
Text-to-Video	text	[damo-vilab/text-to-video-ms-1.7b, ...]	["his model is based on a multi-stage text-to-video generation...", ...]
Video-CLS	video	[MCG-NJU/vidoevae-base, ...]	["VideoMAE model pre-trained on Kinetics-400 for 1600 epochs...", ...]

Table 13: The task list used in HuggingGPT. The first and second columns are the name and arguments of the corresponding task. The third and fourth columns provide some examples of the candidate models and their model descriptions.

CV Tasks			
Image-to-Text	image	[nlpconnect/vit-gpt2-image-captioning, ...]	["This is an image captioning model trained by @ydsieh in flax...", ...]
Text-to-Image	image	[runwayml/stable-diffusion-v1-5, ...]	["Stable Diffusion is a latent text-to-image diffusion model...", ...]
VQA	text + image	[dandelin/vilt-b32-finetuned-vqa, ...]	["Vision-and-Language Transformer (ViLT) model fine-tuned on...", ...]
Segmentation	image	[facebook/detr-resnet-50-panoptic, ...]	["DEtection TRansformer (DETR) model trained end-to-end on ...", ...]
DQA	text + image	[impira/layoutlm-document-qa, ...]	["This is a fine-tuned version of the multi-modal LayoutLM model ...", ...]
Image-CLS	image	[microsoft/resnet-50, ...]	["ResNet model pre-trained on...", ...]
Image-to-image	image	[radames/stable-diffusion-v1-5-img2img, ...]	["Stable Diffusion is a latent text-to-image diffusion model...", ...]
Object-Detection	image	[facebook/detr-resnet-50, ...]	["DEtection TRansformer (DETR) model trained end-to-end on ...", ...]
ControlNet-SD	image	[lllyasviel/sd-controlnet-canny, ...]	["ControlNet is a neural network structure to control diffusion...", ...]

3.2 Model Selection



- **Main Goal:** For each task in the plan, select the *most appropriate* expert model from the thousands available on Hugging Face.
- **Core Challenge:** An LLM's prompt has a limited context length (token limit), so it's impossible to show all available models at once.
- **Final Choice:** The descriptions of *only* these Top-K models are put into the prompt. The LLM then reads these descriptions and makes the final choice for the task.

3.2 Model Selection

Model Selection	Prompt
	#2 Model Selection Stage - Given the user request and the call command, the AI assistant helps the user to select a suitable model from a list of models to process the user request. The AI assistant merely outputs the model id of the most appropriate model. The output must be in a strict JSON format: {"id": "id", "reason": "your detail reason for the choice"}. We have a list of models for you to choose from {{ <i>Candidate Models</i> }}. Please select one model from the list.
	Candidate Models
	{"model_id": model id #1, "metadata": meta-info #1, "description": description of model #1 } {"model_id": model id #2, "metadata": meta-info #2, "description": description of model #2 } ... {"model_id": model id #K, "metadata": meta-info #K, "description": description of model #K }

3.3 Task Execution

The "Placeholder" Solution

1. During Planning

The plan uses a placeholder symbol for the input, like "`<resource>-1`", to show it depends on the output of task 1.



2. During Execution

The system dynamically replaces this placeholder with the **actual** resource (e.g., the file path of the new image) generated by task 1.

⚡ Parallel & Hybrid Execution



Tasks with **no dependencies** are run in parallel (at the same time) to improve efficiency.



The system uses a **hybrid endpoint** (a mix of local and public models) for better speed and stability.

- **Main Goal:** To run the chosen expert models with the correct inputs and handle the flow of data between dependent tasks.
- **Key Challenge: Resource Dependencies**

50/68 • A task often needs the output from a *previous* task (e.g., an image generated by task 1 is needed for task 2).

3.4 Response Generation

FINAL PROMPT CONTAINS:

The user's original request

The full JSON task plan the LLM created

The list of models it selected for each task

The structured inference results (e.g., bounding boxes, generated text) from all the executed models

- **Main Goal:** To synthesize the outputs from all the executed expert models into a single, coherent, natural language response for the user.
- **Method: Comprehensive Context Prompting** - The LLM is called one final time with a complete summary of the entire process.
- **The LLM's Job:** To interpret and summarize all these results in a way that directly and thoughtfully answers the user's initial question, not just list the data.

3.4 Response Generation

Response Generation	Prompt
	#4 Response Generation Stage - With the input and the inference results, the AI assistant needs to describe the process and results. The previous stages can be formed as - User Input: {{ <i>User Input</i> }}, Task Planning: {{ <i>Tasks</i> }}, Model Selection: {{ <i>Model Assignment</i> }}, Task Execution: {{ <i>Predictions</i> }}. You must first answer the user's request in a straightforward manner. Then describe the task process and show your analysis and model inference results to the user in the first person. If inference results contain a file path, must tell the user the complete file path. If there is nothing in the results, please tell me you can't make it.

Table 1: The details of the prompt design in HuggingGPT. In the prompts, we set some injectable slots such as {{ *Demonstrations* }} and {{ *Candidate Models* }}. These slots are uniformly replaced with the corresponding text before being fed into the LLM.

4. Experiments



Test Setup



LLM Controllers Tested

gpt-3.5-turbo

text-davinci-003

gpt-4



LLM Settings (for Reliability)

`temperature=0`

Makes the LLM's output more stable and predictable.

`logit_bias=0.2`

Increases the probability of outputting "{" and "}", ensuring it generates the required JSON format.



Qualitative Results (How it Performed)

1

DESCRIBE & COUNT

Correctly planned **3 tasks** (image classification, captioning, object detection) and used **3 different models**.

2

COMPLEX DEPENDENCIES

For a complex **"pose-to-image-to-speech"** request, it correctly planned **6 tasks** and, most importantly, handled the dependencies (e.g., text-to-image task *waited for* the pose-detection task to finish).

3

EMERGENT PROBLEM-SOLVING

For a simple request like **"Describe this image in detail,"** the system *autonomously* planned and executed **5 different tasks** (captioning, classification, detection, etc.) and combined all the results into one detailed paragraph.

Query: based on the pose image of example1.jpg and the description of example2.jpg, please generate a new one.



Response: Based on the pose image of [example1.jpg](#) and the description of [example2.jpg](#), I have generated a new image [s5f8.jpg](#) for you. The process of generating this image involved using the [openpose-control](#) model to extract the pose information from the basketball image, and then using the [nlpconnect/vit-gpt2-image-captioning](#) to generate a description of the boy in the image. Finally, I used the [llyasviel/sd-controlnet-openpose](#) model to combine the pose information and the description to generate the final image. The complete path of the generated image is [s5f8.jpg](#). If you have any other questions, please let me know.

Task Planning:

- 1: {'task': 'pose-detection', 'id': 0, 'dep': [-1], 'args': {'image': 'example1.jpg'}}
- 2: {'task': 'image-to-text', 'id': 1, 'dep': [-1], 'args': {'image': 'example2.jpg'}}
- 3: {'task': 'pose-text-to-image', 'id': 2, 'dep': [0, 1], 'args': {'text': "<resource>-1", 'image': "<resource>-0"}}



Execution Results:

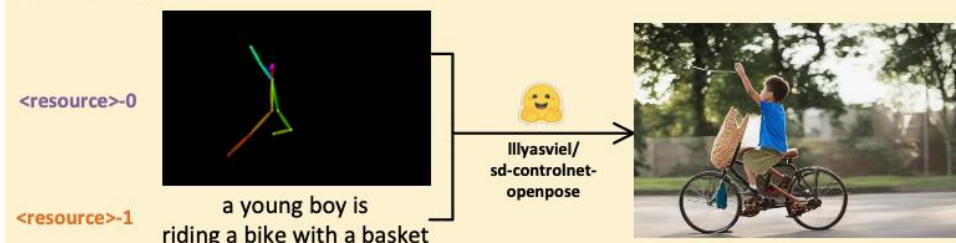
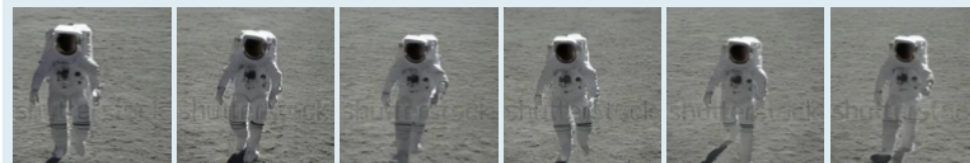


Figure 9: Qualitative analysis of multi-model cooperation with resource dependency.

Query: generate a video with the title "An astronaut is walking in space" and dub it.

Response: Sure. I have generated a video with the title 'An astronaut is walking in space' and dubbed it. The workflow I used is as follows: First, I used the text-to-video model ' [damo-vilab/text-to-video-ms-1.7b](#) ' to generate a video from the text. The inference result is the generated video ' [/videos/5696.mp4](#) '. Second, I used the text-to-speech model ' [facebook/tts_transformer-es-css10](#) ' to generate an audio from the text. The inference result is the generated audio ' [/audios/3f9e.flac](#) '. Finally, I combined the video and audio to generate the dubbed video. I hope this answers your question. Is there anything else I can help you with?



Task Planning:

- 1: {'task': 'text-to-video', 'id': 0, 'dep': [-1], 'args': {'text': "An astronaut is walking in space" }}
- 2: {'task': 'text-to-speech', 'id': 1, 'dep': [-1], 'args': {'text': "An astronaut is walking in space" }}

Execution Results:

1. {"generated video": "/videos/5696.mp4"}
2. {"generated audio": "/audios/3f9e.flac"}

Query: read the image example.jpg for me.

INVOICE NO : 18301/102/T0305

Response: Sure. I have read the image [example.jpg](#) for you. The inference result is 'INVOICE NO : 18301/102/T0305'. The model used for this task is [microsoft/trocr-base-printed](#), which is best suited for the task of image-to-text as it is fine-tuned on the SROIE dataset which is specifically designed for this task. Then I have converted the text to audio, the generated audio is [/audios/da5s.wav](#) and the model used for this task is [facebook/fastspeech2-en-ljspeech](#), which is a FastSpeech 2 text-to-speech model which is suitable for the task of text-to-speech. Is there anything else I can help you with?

Task Planning:

- 1: {'task': 'image-to-text', 'id': 0, 'dep': [-1], 'args': {'image': "example.jpg"}}
- 2: {'task': 'text-to-speech', 'id': 1, 'dep': [0], 'args': {'text': "<resource>-0"}}

Execution Results:

1. {"generated text": "INVOICE NO : 18301/102/T0305"}
2. {"generated audio": "/audios/da5s.wav"}

Figure 10: Qualitative analysis of multi-model cooperation on video and audio modalities.

Quantitative Evaluation

1. Created a Dataset

Built a new dataset of user requests.



2. Classified Task Complexity

Requests were grouped into three sub-types:

SINGLE TASK

One simple request (e.g., "classify this image").

SEQUENTIAL TASK

A multi-step request where order matters.

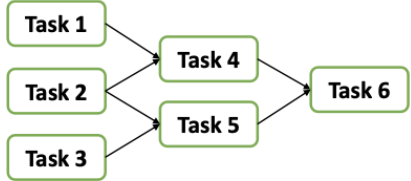
GRAPH TASK

A complex request with multiple dependencies.

Main Goal: To scientifically measure the *Task Planning* capability of different LLMs when acting as the system's "brain."

Used Smart Metrics: For complex "Graph Tasks," a simple score isn't enough (multiple plans can be correct). So, they created a "**GPT-4 Score**" by using GPT-4 itself as a "judge" to score the quality of a plan.

Quantitative Evaluation

Task Type	Diagram	Example	Metrics
Single Task		Show me a funny image of a cat	Precision, Recall, F1, Accuracy
Sequential Task		Replace the cat with a dog in example.jpg	Precision, Recall, F1 Edit Distance
Graph Task		Given a collection of image A: a.jpg, B: b.jpg, C: c.jpg, please tell me which image is more like image B in terms of semantic, A or C?	Precision, Recall, F1 GPT-4 Score

LLM	Acc ↑	Pre ↑	Recall ↑	F1 ↑
Alpaca-7b	6.48	35.60	6.64	4.88
Vicuna-7b	23.86	45.51	26.51	29.44
GPT-3.5	52.62	62.12	52.62	54.45

Table 3: Evaluation for the single task. “Acc” and “Pre” represents Accuracy and Precision.

LLM	ED ↓	Pre ↑	Recall ↑	F1 ↑
Alpaca-7b	0.83	22.27	23.35	22.80
Vicuna-7b	0.80	19.15	28.45	22.89
GPT-3.5	0.54	61.09	45.15	51.92

Table 4: Evaluation for the sequential task. “ED” means Edit Distance.

Quantitative Evaluation

The Core Finding: The "Brain" is the Bottleneck

- The capability of the central LLM controller is the **single most important factor** for success.
- **Massive Performance Gap:** On complex Graph Tasks, **GPT-3.5 (F1 score: 51.9)** dramatically outperformed open-source models like **Vicuna-7b (F1 score: 18.7)**.
- **Room to Grow:** On a high-quality, *human-annotated* dataset, even **GPT-4** showed a "substantial gap" from a perfect score, proving that improving the LLM's planning ability is a key area for future research.

LLM	GPT-4 Score ↑	Pre ↑	Recall ↑	F1 ↑
Alpaca-7b	13.14	16.18	28.33	20.59
Vicuna-7b	19.17	13.97	28.08	18.66
GPT-3.5	50.48	54.90	49.23	51.91

Table 5: Evaluation for the graph task.

LLM	Sequential Task		Graph Task	
	Acc ↑	ED ↓	Acc ↑	F1 ↑
Alpaca-7b	0	0.96	4.17	4.17
Vicuna-7b	7.45	0.89	10.12	7.84
GPT-3.5	18.18	0.76	20.83	16.45
GPT-4	41.36	0.61	58.33	49.28

Ablation Study

- The authors tested if adding more examples (few-shot demonstrations) to the prompt could improve the "brain's" planning performance.
- **Finding 1 (Number):** Performance improves with 1-4 examples but then quickly plateaus.
- **Finding 2 (Variety):** Increasing the *variety* of task types in the examples moderately improves performance.
- **Conclusion:** This shows that simply adding more examples cannot overcome the core reasoning limitations of a less-capable LLM. The "brain's" inherent power is what matters most.

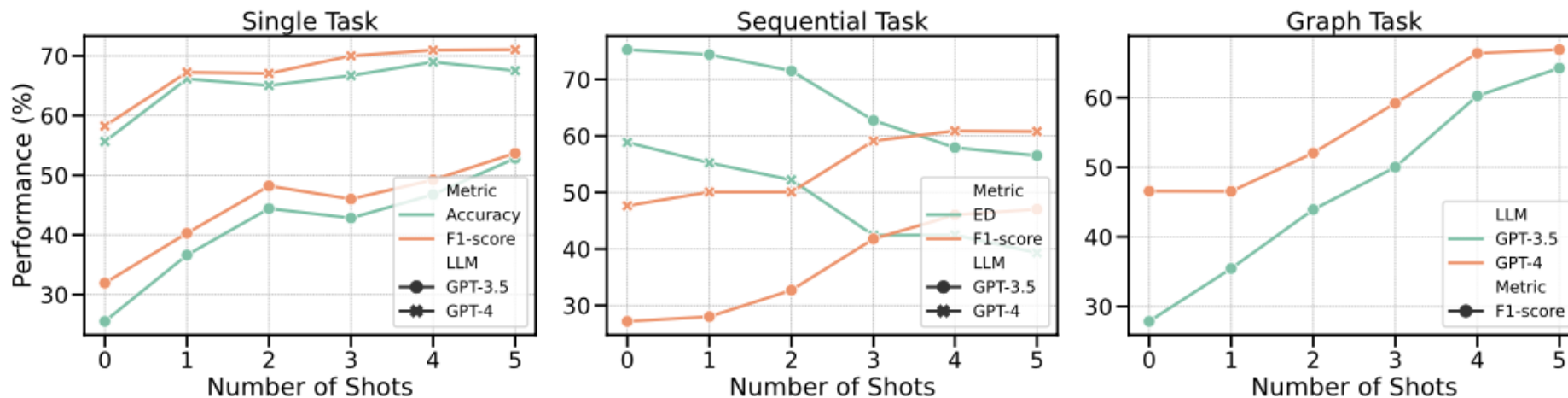


Figure 3: Evaluation of task planning with different numbers of demonstrations.

Human Evaluation

TEST

Human experts evaluated the entire system's performance on **130 diverse requests**.

METRICS

Passing Rate, Rationality, and Success Rate.

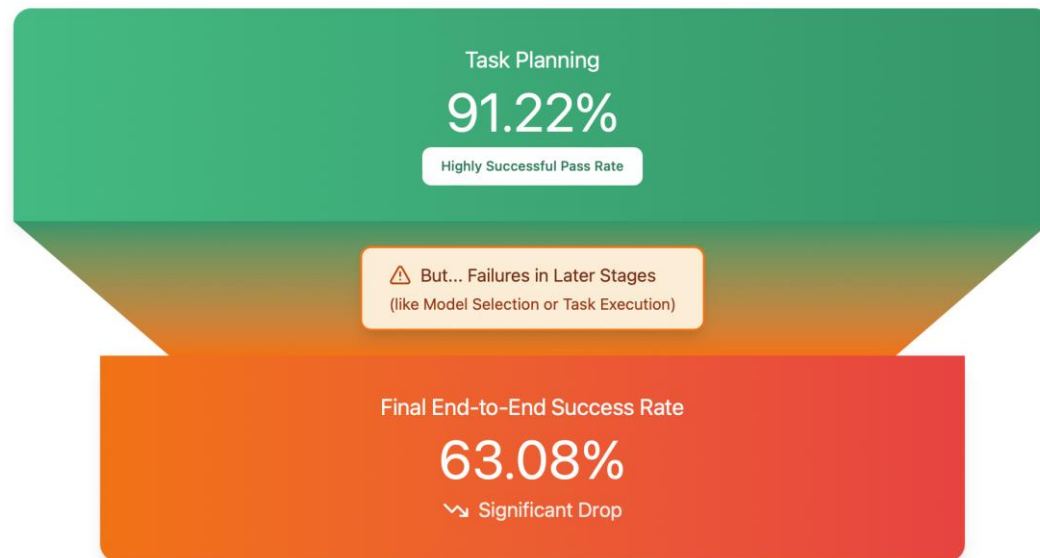
COMPARISON

GPT-3.5 **significantly outperformed** open-source LLMs (like Alpaca-13b and Vicuna-13b) across all stages.

LLM	Sequential Task		Graph Task	
	Acc ↑	ED ↓	Acc ↑	F1 ↑
Alpaca-7b	0	0.96	4.17	4.17
Vicuna-7b	7.45	0.89	10.12	7.84
GPT-3.5	18.18	0.76	20.83	16.45
GPT-4	41.36	0.61	58.33	49.28

Table 6: Evaluation on the human-annotated dataset.

Main Finding: The Bottleneck Effect



LLM	Task Planning		Model Selection		Response
	Passing Rate ↑	Rationality ↑	Passing Rate ↑	Rationality ↑	Success Rate ↑
Alpaca-13b	51.04	32.17	-	-	6.92
Vicuna-13b	79.41	58.41	-	-	15.64
GPT-3.5	91.22	78.47	93.89	84.29	63.08

Table 8: Human Evaluation on different LLMs. We report two metrics, passing rate (%) and rationality (%), in the task planning and model selection stages and report a straightforward success rate (%) to evaluate whether the request raised by the user is finally resolved.

- This proves that while planning is a major hurdle, a powerful LLM is critical for *every stage* of the process.

CV > Depth Estimate

Query: please estimate the depth of ks4j.jpg



Response: I have estimated the depth of the image located at examples/ks4j.jpg using the 🤖 [intel/dpt-large](#) model. This model is specifically trained on 1.4 million images and is a Dense Prediction Transformer (DPT) model. The generated depth image can be found at '📁 [j45s.jpg](#)'.



NLP > Named Entity Recognition

Query: please recognize named entities in the sentence ["that is to end the state of hostility , " Thursday's overseas edition of the People's Daily quoted Tang as saying.]

Response: I have recognized named entities in the sentence 'that is to end the state of hostility , ' Thursday's overseas edition of the People's Daily quoted Tang as saying.'. The entities recognized are 'People's Daily' and 'Tang'. Both of them are organizations and people respectively. The model I used is '🤖 [dslim/bert-base-NER](#)', which is a fine-tuned BERT model that is ready to recognize named entities in the sentence. The inference results are very accurate, with a score of 0.9992 for 'People's Daily' and 0.998 for 'Tang'.

NLP > Text Generation

Query: please complete a sentence [COVID-19 is transmitted via droplets, air-borne,]

Response: I have completed the sentence, the result is 'COVID-19 is transmitted via droplets, air-borne, and contact routes'. I used a 🤖 [microsoft/biogpt](#) model, a domain-specific generative Transformer language model pre-trained on large-scale biomedical literature.

CV > Document Question Answering

Query: look at the document doc.jpg, what is the purchase amount?

Response: Sure. I have used 🤖 [mpira/layoutlm-document-qa](#) model to analyze the document 📁 [doc.jpg](#) you provided. After the analysis, I have found that the purchase amount is \$1,000,000,000. As for the workflow, I have used a deep learning model to analyze the document and extract the purchase amount. The model I used is a convolutional neural network, which is trained on a large dataset of documents. The inference results are the purchase amount of \$1,000,000,000. If you need more information, please let me know.



CV > Image to Image

Query: give me a similar image with example.jpg

Sure. For your request, I used the model 🤖 [lambdalabs/sd-image-variations-diffusers](#) to generate an image from the image you provided. The generated image is located at 📁 [/images/5373.jpg](#). If there is nothing in the results, I will tell you that I can't make it. Is there anything else I can help you with?



Video > Generation

Query: please generate a video based on "Spiderman is surfing"

Response: Sure, I can do that. I have generated a video based on your prompt "Spiderman is surfing". The video is saved at the following path: 📁 ["/videos/4076.mp4"](#). 1. Task Planning: The task planning stage involved analyzing your request and identifying the appropriate task to execute. In this case, the task was "text-to-video". 2. Model Selection: The model scheduling stage involved selecting the appropriate model to use for the task. In this case, the only available model was 🤖 [damo-vilab/text-to-video-ms-1.7b](#). 3. Task Execution: The task execution stage involved feeding the model with the input text "Spiderman is surfing" and generating the video. The inference result was a generated video saved at the path 📁 ["/videos/4076.mp4"](#). I hope this response meets your requirements.



Audio > Generation

Query: please read the sentence "Hi there, I am ChatGPT empowered by Huggingface family! Yes, I can provide more than one thousand models for dozens of tasks. For more fun and creativity, I have invited Diffusers family to join our team. Feel free to experience it!" for me.

Response: Sure, I can do that. For your request, I used the 🤖 [Facebook/FastSpeech2-en-USpeech](#) model to generate an audio file from the text you provided. The generated audio file is located at 📁 ["/audios/d5f3.wav"](#). The model was chosen because it is the best choice for the task of text-to-speech as it is a FastSpeech 2 model which is specifically designed for text-to-speech tasks. It is also the most recent model in the list and has the most potential to provide the best results. I hope this answers your question. Is there anything else I can help you with?

Query #1: Please generate a canny image based on example.jpg

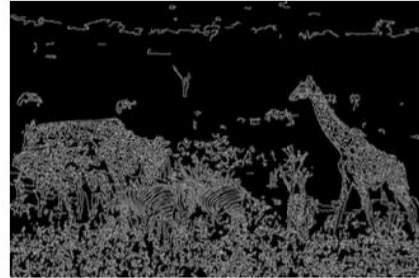
Response #1: I have generated a canny image for you based on the image located at example.jpg. The task process involved selecting the 🤖 **canny-control** model for image processing, as it covers common image processing functions. The inference results include a generated image, which can be found at 📁 **images/df8t.jpg**.

Task Planning #1:

1: {"task": "image-to-text", "id": 0, dep: [-1], "args": {"image": "example.jpg"}}
2: {"task": "control-canny", "id": 0, dep: [-1], "args": {"image": "example.jpg"}}

Execution Results #1:

1. {"generated text": "a herd of giraffes and zebras grazing in a field"}
2. {"generated image": "images/df8t.jpg"}



Query #2: then based on the above canny image and a prompt "a photo of a zoo", generate a new image.

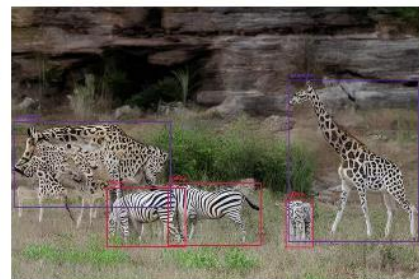
Response #2: I have received your request to generate a canny image of a photo of a zoo. I have selected the model 🤖 **Ilyasviel/sd-controlnet-canny** as it is the best model for this task. I have processed the image and generated a new image located at 📁 **images/ja2f.jpg**. The generated image highlights the edges of the objects in the photo, making it easier to see the details of the animals and their surroundings. Overall, the task was to convert a text prompt 'a photo of a zoo' into a canny image using the selected model.

Task Planning #2:

1: {"task": "canny-text-to-image", "id": 0, dep: [-1], "args": {"text": "a photo of a zoo", "image": "example.jpg"}}

Execution Results #2:

1. {"generated image": "images/ja2f.jpg"}



Query #3: please detect objects in the above image

Response #3:

Sure, here is the result of your command. I have successfully detected objects in the image you provided. **The detected objects are giraffes and zebras.** I used the 🤖 **facebook/detr-resnet-101** model for object detection, which is specifically designed for object detection tasks and has a ResNet-101 backbone, which is a powerful feature extractor. Additionally, the DETR architecture used in this model is an end-to-end object detection model that has shown promising results in various benchmarks. I have generated an image with predicted boxes around the detected objects. You can find the generated image at the following path: 📁 **images/dif2.jpg**. The predicted objects and their corresponding scores are as follows: - Giraffe with a score of 0.999- Zebra with a score of 0.998- Zebra with a score of 0.998- Giraffe with a score of 0.969- Zebra with a score of 0.997. I hope this information is helpful to you.

Task Planning #3:

1: {"task": "object-detection", "id": 0, "dep": [-1], "args": {"image": "images/ja2f.png"}}

Execution Results #3:

1. {"generated image with predicted box": "/images/dif2.jpg", "predicted": [{"box": {"xmax": 760, "xmin": 515, "ymax": 437, "ymin": 136}, "label": "giraffe", "score": 0.9992127418518066}, {"box": {"xmax": 561, "xmin": 510, "ymax": 447, "ymin": 357}, "label": "zebra", "score": 0.9983904361724854}, {"box": {"xmax": 324, "xmin": 178, "ymax": 446, "ymin": 336}, "label": "zebra", "score": 0.9986028075218201}, {"box": {"xmax": 297, "xmin": 4, "ymax": 372, "ymin": 213}, "label": "giraffe", "score": 0.9694724082946777}]}

Query: Describe this image example.jpg in as much detail as possible.

Response: The image you provided appears to show a family of four dogs playing in the grass. For this task, I ran multiple CV tasks, including image-to-text, image-classification, object-detection, visual-question-answering, and document-question-answering. I chose specific models for each task that were best suited to addressing your query, based on their training and performance characteristics.

For image-to-text, I chose the 🤖 [nlpconnect/vit-gpt2-image-captioning](#) model. This model generates an image caption with the given image. I applied this model to the input image and the developed output was: **a family of four dogs are playing in the grass**

For image-classification, I selected the 🤖 [google/vit-base-patch16-224](#) model. This model is trained on natural images dataset and it can predict the label of the image output. I applied this model to the image and get the results showing the risk of each label. It shows the highest risk at **"Rhodesian ridgeback"** with a score of 93.8%

For object-detection, I used 🤖 [facebook/detr-resnet-101](#) model. This model is trained on COCO panoptic segmentation for object detection. I used this model to generate an image file with the predicted box. The generated image with predicted box is at: 📁 [/images/sd48.jpg](#)

Regarding your request for image segmentation, I have completed the task using a pre-trained model called 🤖 [facebook/detr-resnet-50-panoptic](#). The segmentation mask shows the different regions of the image that belong to different classes. In this case, the model has identified two main classes in the image, which are **"dog"**, **"grass"** and **"tree"**. The generated image with the segmentation mask can be found at 📁 [/images/t4s1.jpg](#)

For visual question answering, I used 🤖 [dandelin/vilt-b32-finetuned-vqa](#) model. This model is a Vision-and-Language Transformer (ViLT) model, which is fine-tuned on VQAv2. It is suitable for visual-question-answering tasks, and is likely to provide accurate results for the given image. For the question "what are in the image?" It reply with a prediction **"dogs"** and the confidence is 84.9%.

Task Planning:

1. {"id": "0", "dep": [-1], "task": "image-to-text", "args": {"image": "example.jpg" }},
2. {"id": "1", "dep": [-1], "task": "image-classification", "args": {"image": "example.jpg" }},
3. {"id": "2", "dep": [-1], "task": "object-detection", "args": {"image": "example.jpg" }},
4. {"id": "3", "dep": [-1], "task": "image-segmentation", "args": {"image": "example.jpg" }},
5. {"id": "4", "dep": [-1], "task": "visual-question-answering", "args": {"text": "describe this image in detail", "image": "example.jpg" }},

Execution Results:

1. {"generated_text": "a family of four dogs are playing in the grass "}
2. [{"score": 0.938717246055603, "label": "Rhodesian ridgeback"}, {"score": 0.01652258075773716, "label": "boxer"}, {"score": 0.006381669547408819, "label": "Great Dane"}, {"score": 0.006234415341168642, "label": "vizsla, Hungarian pointer"}, {"score": 0.005944834090769291, "label": "bull mastiff"}]
3. [{"generated image with predicted box": "/images/sd48.jpg", "predicted": [{"box": {"xmax": 463, "xmin": 373, "ymax": 267, "ymin": 199}, "label": "bus", "score": 0.9981155395507812}, {"box": {"xmax": 292, "xmin": 124, "ymax": 224, "ymin": 175}, "label": "airplane", "score": 0.9983609318733215}, {"box": {"xmax": 554, "xmin": 487, "ymax": 277, "ymin": 226}, "label": "person", "score": 0.9910836219787598}}]
4. {"generated image with segmentation": "/images/t4s1.jpg", "predicted": [{"score": 0.989, "label": "grass"}, {"score": 0.999, "label": "dog"}, {"score": 0.999, "label": "tree"}, {"score": 0.999, "label": "dog"}]}
5. [{"answer": 'dogs', 'score': 0.8488452434539795}, {"answer": 'dog', 'score': 0.04168461635708809}]



Strengths

- **Extreme Extensibility and Scalability:** The system is not limited to a fixed set of capabilities. You can add a new state-of-the-art model for any task (e.g., a new 3D generation model) simply by making its description available. It can theoretically grow and improve every day as the community adds new models to Hugging Face.
- **Optimal Tool Usage:** Instead of relying on a single, generalist multi-modal model, HuggingGPT can always select the absolute best, most specialized, state-of-the-art model for a specific sub-task, leading to potentially higher quality results.
- **Separation of Concerns:** The architecture cleanly separates general reasoning (the LLM's job) from specialized execution (the expert models' job). This is a powerful and flexible engineering paradigm that can be adapted for many domains beyond just AI models (e.g., using web APIs, databases, or scientific instruments as tools).
- **Complex Task Decomposition:** The system shows a remarkable emergent ability to break down vague, complex human requests into a logical sequence of concrete, machine-executable steps. This planning capability is a significant step towards more autonomous AI.

Limitations

- **The "Brain" as a Central Bottleneck**

- The system's performance is *strictly capped* by the planning and reasoning capability of the LLM controller. As the quantitative results showed, even GPT-4 is not a perfect planner.

- **Efficiency & High Latency**

- The system is slow. It requires multiple sequential interactions with the LLM (for planning, selection, and generation), which adds significant time costs (latency) to every request.

- **Token Length Limits**

- The LLM's limited context window (max tokens) makes it impossible to review all available models. This forces the system to use imperfect shortcuts, like only considering the "top-K downloaded" models.

- **Instability & Brittleness**

- The workflow is a "fragile chain." Because LLMs can be uncontrollable and fail to follow instructions perfectly, a single malformed output or error at any stage can cause the entire process to fail.

Conclusion

- **The HuggingGPT system is proposed** to solve complex AI tasks by using language as a universal interface.
- **It proves a new concept:** An LLM can act as a "controller" or "brain" to manage and orchestrate specialized expert models from ML communities like Hugging Face.
- **The LLM's reasoning is key:** It allows the system to dissect user intent, decompose tasks, assign the best model, and integrate the final results into a single answer.
- **This approach paves a new pathway toward AGI** by successfully leveraging the collective power of the entire machine learning community.

Conclusion

How can we build Artificial General Intelligence:

- **Path A: The Polymath**

- Do we build a *single, massive, end-to-end model* that tries to learn every skill (e.g., Kosmos, LLaVA)?

- **Path B: The Architect**

- Do we build a *generalist reasoning “brain”* that delegates tasks to an ever-growing ecosystem of specialist tools (e.g., HuggingGPT, Toolformer)?

How this work is going

These early works led to today's **LLM Agents** and **Tool Use**, evolving in three main directions:

1. Native Tool Use (Function Calling)

- **What:** Models (like Gemini) are *trained* to call specific functions (e.g., `get_weather()`).
- **Why:** More robust, reliable, and integrated.

2. Autonomous Agent Frameworks (e.g., LangChain)

- **What:** The "controller" idea in a *continuous loop*. Agents plan, act, reflect, and re-plan to achieve complex goals (e.g., "Write a research report").

3. Natively Multi-Modal Models (e.g., Gemini)

- **What:** The LLM *becomes* the expert (it can "see" and "hear" directly), reducing the need for many external models.

The Future: Convergence The most powerful systems (like Gemini Advanced) combine all three: **multi-modal** input, **native tools**, and an **agentic loop**.

Discussion

- **The Quality Proxy:** The system ranks candidate models by their number of downloads on Hugging Face. What are the potential risks of using "popularity" as a proxy for "quality" and "suitability" in model selection?
- **The Future of LLM Training:** This paper uses a pre-trained, text-only LLM. If you were to train a new LLM from scratch specifically for this "coordinator" role, what kind of data would you train it on to make it a better planner and tool user?
- **Beyond AI Models:** This framework uses Hugging Face models as its tools. What other "toolsets" could you plug into this architecture? What would a "TravelAgentGPT" that uses APIs from Expedia, Kayak, and Uber look like? Or a "ScientistGPT" connected to computational chemistry tools?
- **Is This "Real" Reasoning?:** Is HuggingGPT demonstrating true problem-solving, or is it a very clever feat of prompt engineering that creates a powerful *illusion* of understanding by stitching together tools it doesn't truly comprehend? Where do you draw the line?