Stanford

Stanford ARTIFICIAL INTELLIGENCE

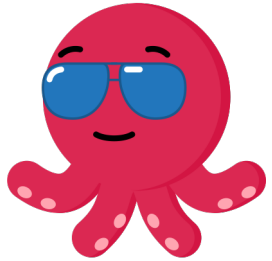
OctoTools: An Agentic Framework with Extensible Tools for Complex Reasoning

slack

Pan Lu

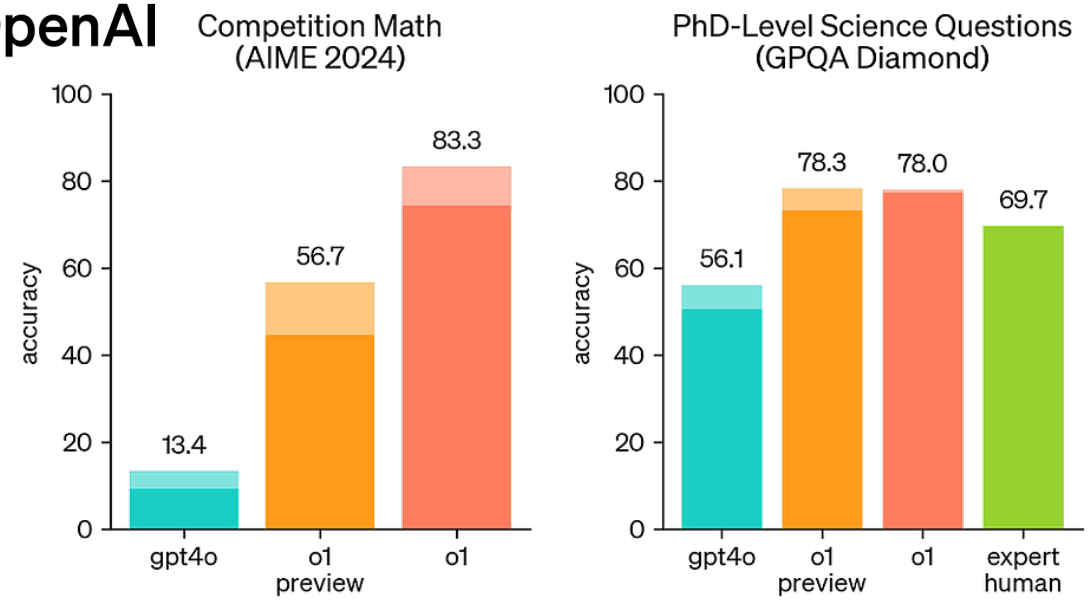
05/03/2025

https://octotools.github.io/

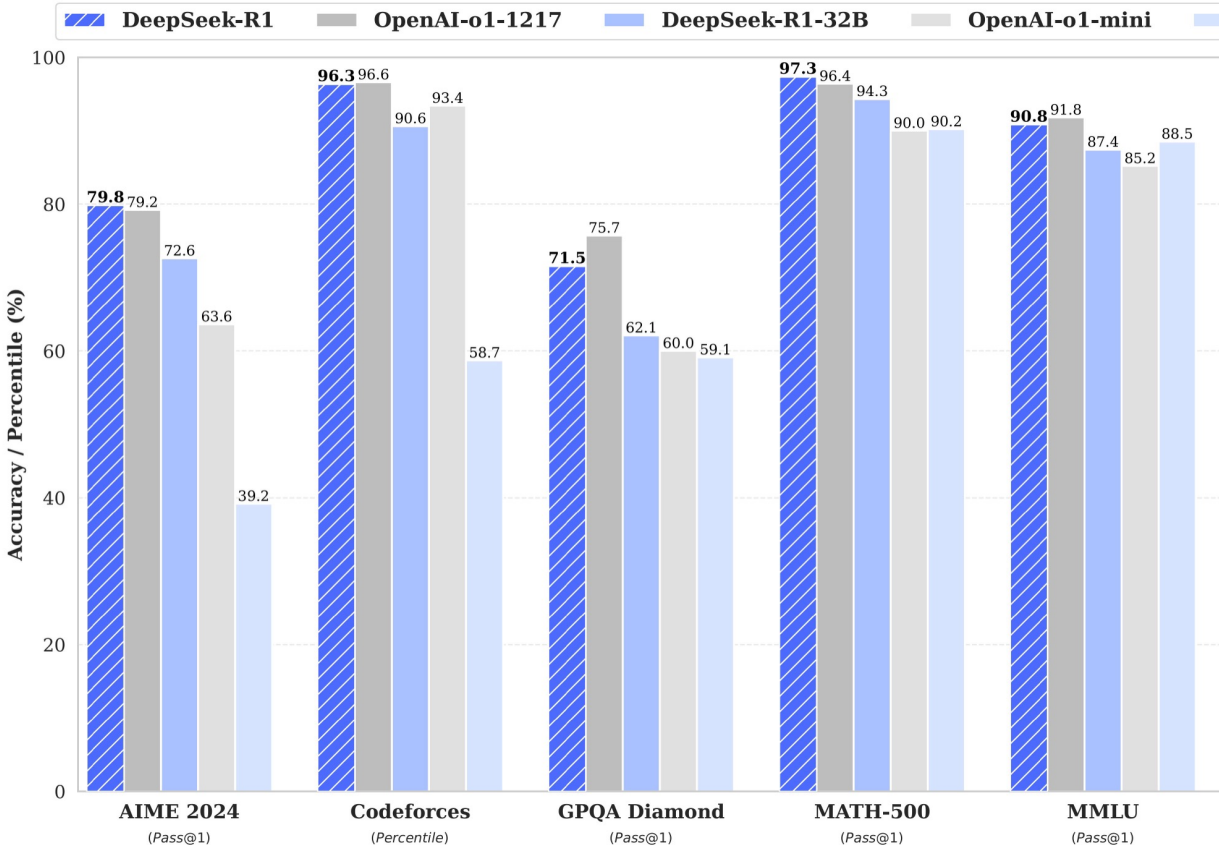


Latest Progress in LLMs

OpenAI

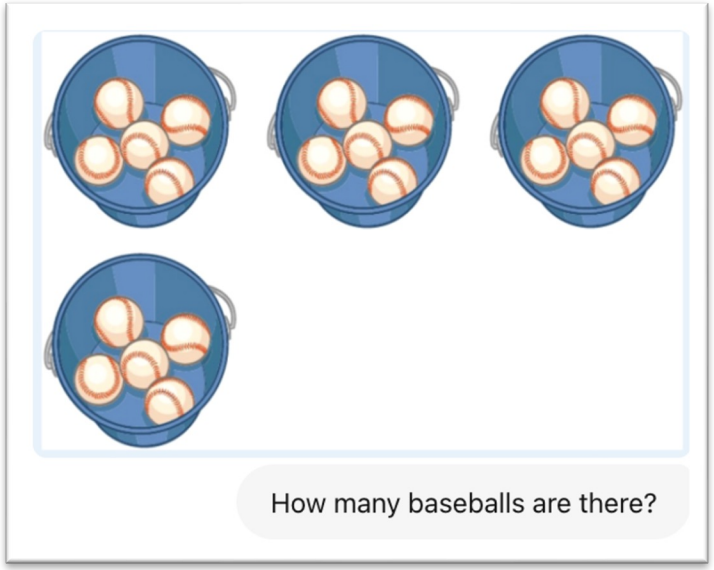


Alibaba.com



deepseek

These Models Are Far From Perfect...



GPT-4o

There are a total of 19 baseballs across the four buckets:

- The first bucket (top left) contains 4 baseballs.
- The second bucket (top right) contains 5 baseballs.
- The third bucket (bottom left) contains 4 baseballs.
- The fourth bucket (bottom right) contains 6 baseballs.

Adding them together: 4 + 5 + 4 + 6 = 19 baseballs.

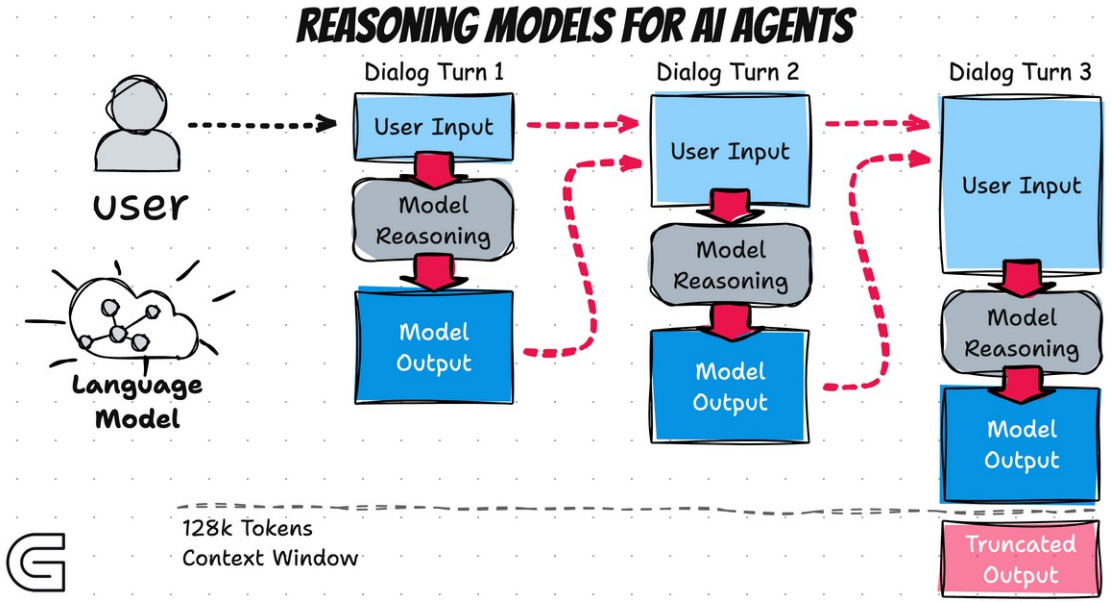
Question: What is the highest amount this glass measures?

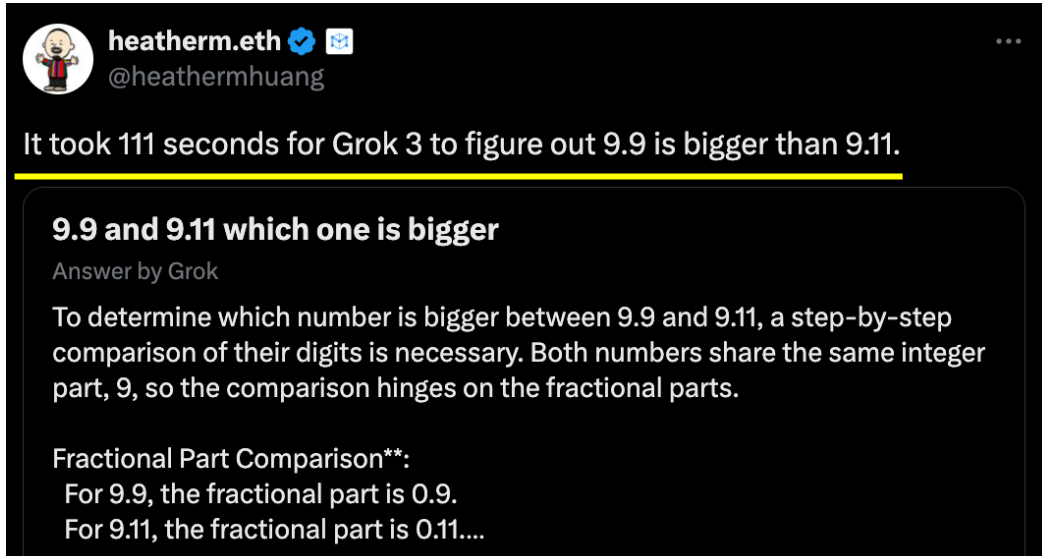
Ground truth: 400 ml

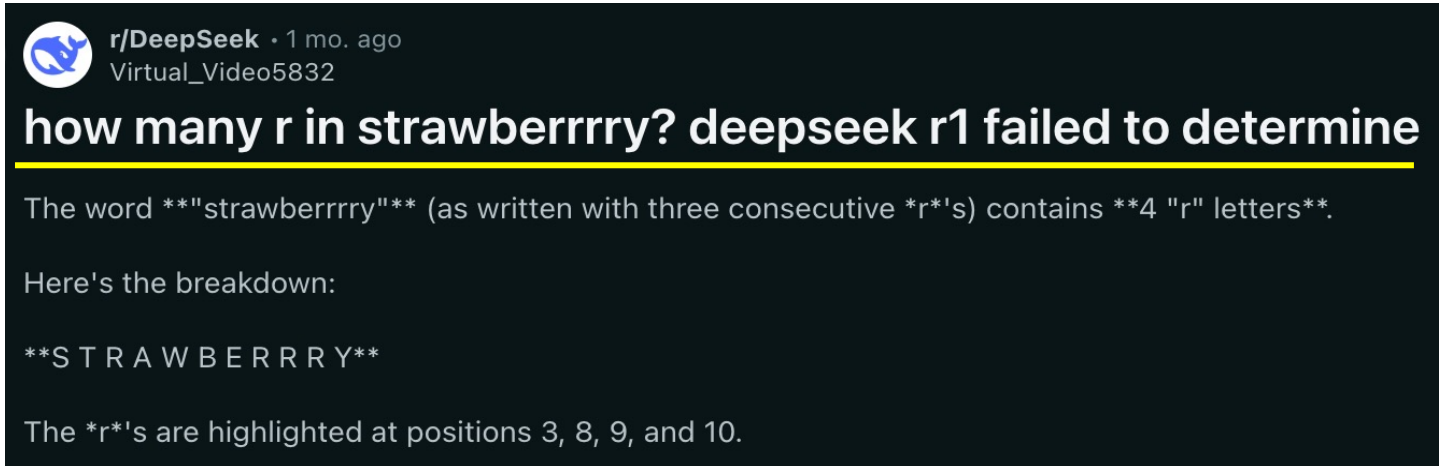
OpenAI GPT-4o

The highest amount this glass beaker measures is 600 milliliters (ml). The markings on the beaker go up to 600 ml, which indicates its maximum volume measurement.

Existing Solution: Test-Time Computation

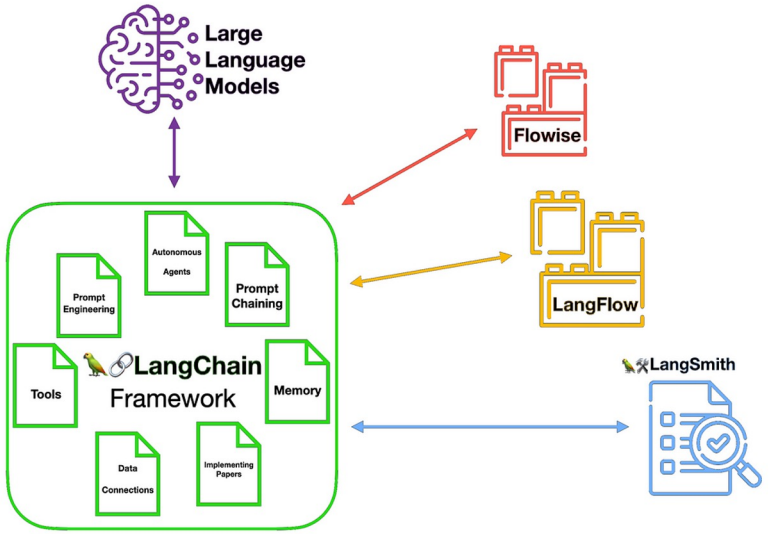






Existing Solution: Agentic Frameworks

LangChain Ecosystem



AutoGen 0.2

An Open-Source Programming Framework for Agentic AI

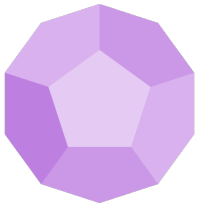
why we no longer use LangChain for building our AI agents

When abstractions do more harm than good: Lessons learned using LangChain in production and what we should've done instead

Our Solution

Complex Reasoning

Easy-to-use

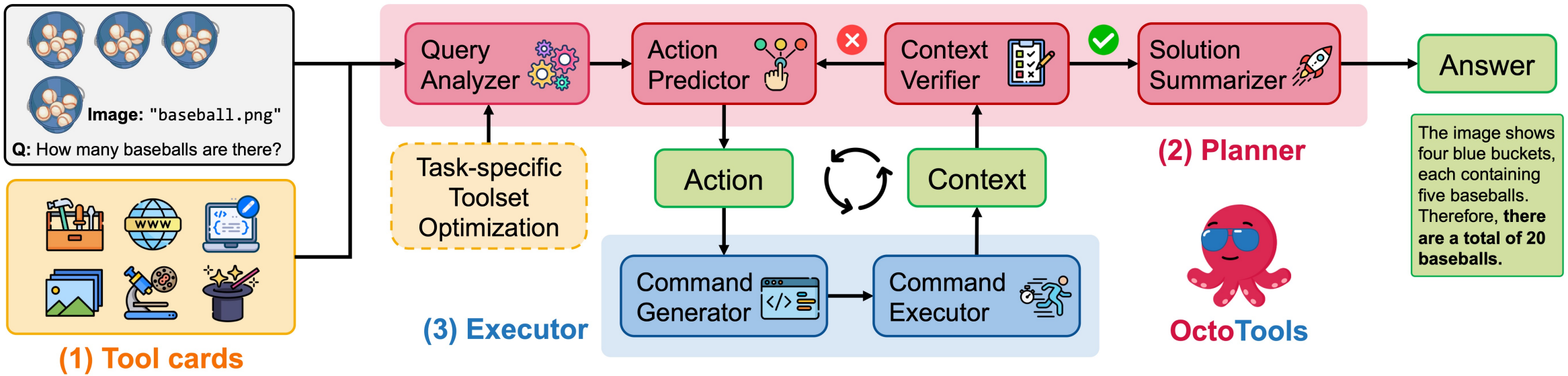


Extensibility

Evaluation

The OctoTools Framework

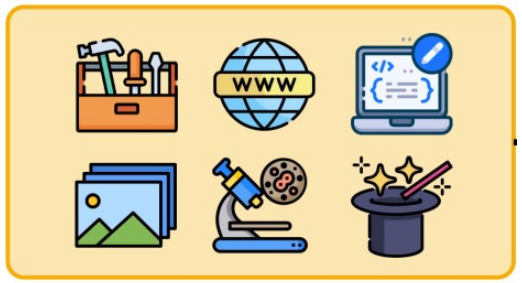
Planner: generate a high-level to address the global objective, a low-level plan to refine actions step by step



Tool cards: define tool-usage metadata, encapsulate tools

Executor: instantiate tool calls by generating executable commands, save structured results in the context

Tool Cards



(1) Tool cards

Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Image Captioner Tool: Metadata

```
tool_name="Image_Captioner_Tool",

tool_description="A tool that generates captions for images using OpenAI's multimodal
    model.",

input_types={
   "image": "str - The path to the image file.",
   "prompt": "str - The prompt to guide the image captioning (default: 'Describe this
       image in detail.')."
},

output_type="str - The generated caption for the image.",

demo_commands=[
   {
      "command": 'execution = tool.execute(image="path/to/image.png")',
      "description": "Generate a caption for an image using the default prompt and
          model."
   },
   {
      "command": 'execution = tool.execute(image="path/to/image.png", prompt="Explain
          the mood of this scene.")',
      "description": "Generate a caption focusing on the mood using a specific prompt
          and model."
   }
],

user_metadata = {
   "limitation": "The Image_Captioner_Tool provides general image descriptions but has
       limitations: 1) May make mistakes in complex scenes, counting, attribute
       detection, and understanding object relationships. 2) Might not generate
       comprehensive captions, especially for images with multiple objects or abstract
       concepts. 3) Performance varies with image complexity. 4) Struggles with
       culturally specific or domain-specific content. 5) May overlook details or
       misinterpret object relationships. For precise descriptions, consider: using it
       with other tools for context/verification, as an initial step before refinement,
        or in multi-step processes for ambiguity resolution. Verify critical
       information with specialized tools or human expertise when necessary."
}
```

📸 Image Captioner

Generate a caption for a given image with a text prompt.

Metadata | Code | Example

🔍 Relevant Patch Zoomer

Locate and zoom in relevant quarter patches in an image given a question.

Metadata | Code | Example

📝 Text Detector

Detect text with coordinates and confidence scores in an image by EasyOCR.

Metadata | Code | Example

👁️ Object Detector

Detect objects in an image using the Grounding DINO model.

Metadata | Code | Example

📚 Wikipedia Search

Search Wikipedia for relevant information based on a given query.

Metadata | Code | Example

🌐 Google Search

Search the Google website for relevant information based on a given query.

Metadata | Code | Example

🔗 URL Extractor

Visit the given URL and extract all text from that page.

Metadata | Code | Example

📄 ArXiv Paper Search

Search arXiv for the latest literature based on a given query.

Metadata | Code | Example

🏥 PubMed Paper Search

Search PubMed for the latest literature based on a given query.

Metadata | Code | Example

🌿 Nature News Search

Search the latest news articles from the Nature website.

Metadata | Code | Example

🔬 Pathology Classifier

Classify H&E-stained pathology images into one of the given options.

Metadata | Code | Example

🧱 Python Interpreter

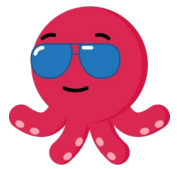
Generate and execute Python code snippets for basic calculations.

Metadata | Code | Example

🧙 Generalist Solutioner

Base tool that answers general questions without using any external tools.

Metadata | Code | Example

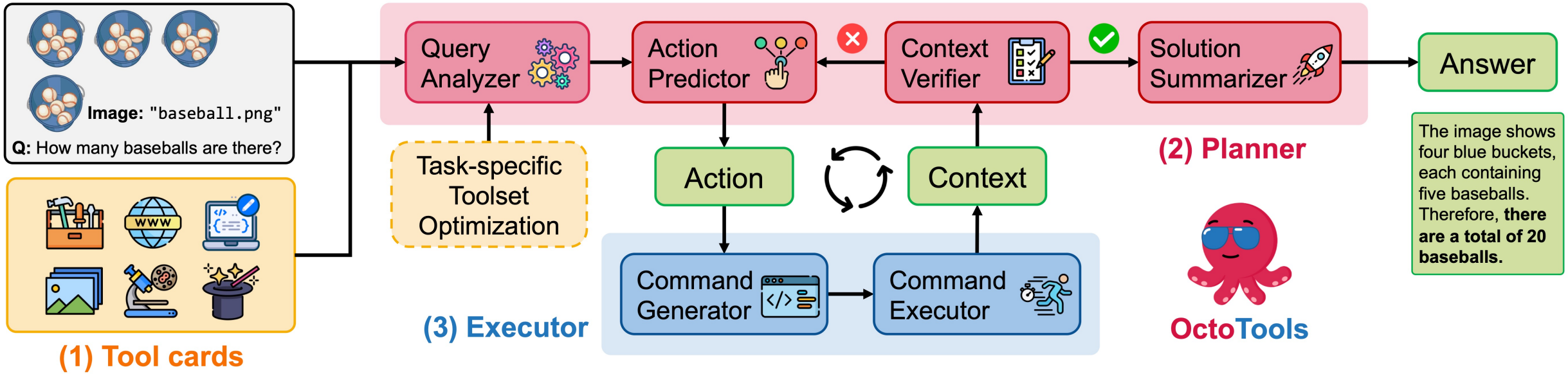


⚒️ More Tools ...

More tools can be added!

Metadata | Code | Example

https://octotools.github.io/#tool-cards



Wrong Prediction from GPT-4o

There are a total of 19 baseballs across the four buckets:

- The first bucket (top left) contains 4 baseballs.
- The second bucket (top right) contains 5 baseballs.
- The third bucket (bottom left) contains 4 baseballs.
- The fourth bucket (bottom right) contains 6 baseballs.

Adding them together: 4 + 5 + 4 + 6 = 19 baseballs.

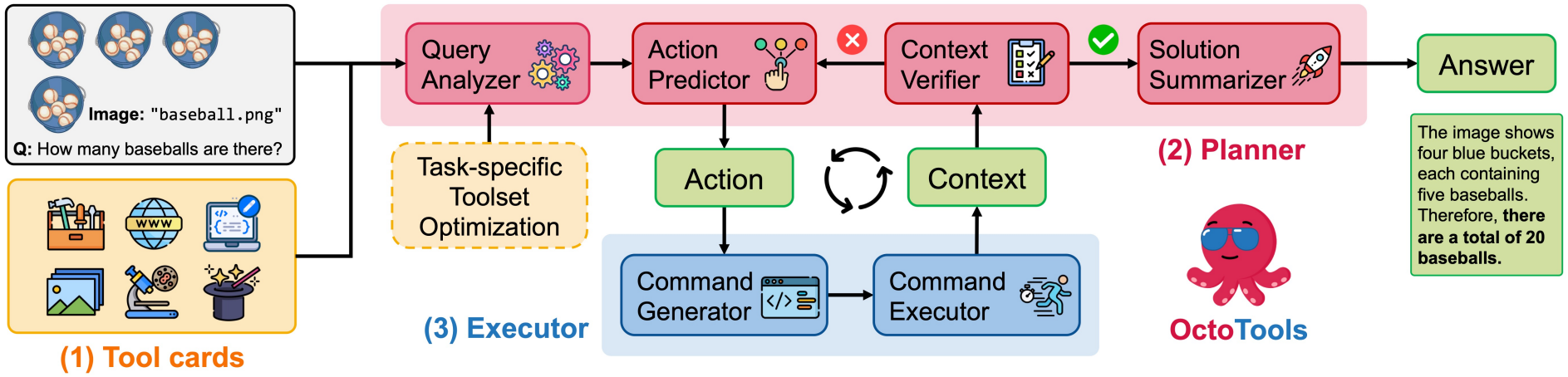


Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Object Detector Tool

tool_description="A tool that detects and saves objects in an image"

inputs={image,labels,threshold}

outputs={list: objects, scores, boxes}

demos=tool.execute(image="car.png", labels=["car"])

user_metadata={"limitation": "May not detect objects accurately."}

.....

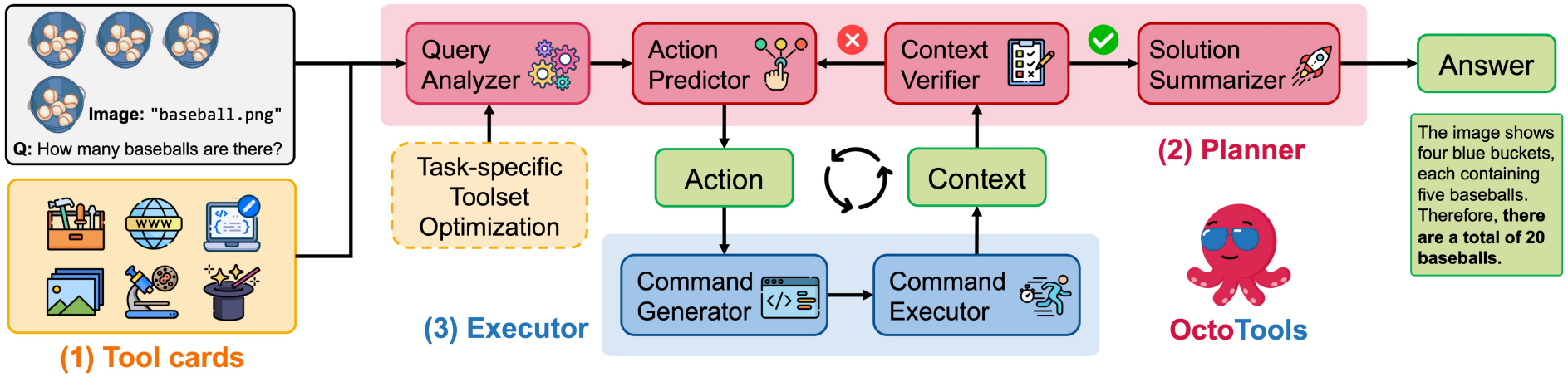


Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Object Detector Tool

tool_description="A tool that detects and saves objects in an image"

inputs={image,labels,threshold}

outputs={list: objects, scores, boxes}

demos=tool.execute(image="car.png", labels=["car"])

user_metadata={"limitation": "May not detect objects accurately."}

.....

Step 0: Query Analyzer

Summary: determine the total number of baseballs in the image.

Required skills:

1. understanding and interpreting visual content.
2. ability to operate and execute commands using the tools.
3. evaluating tool outputs and making decisions.

Relevant tools:

`Image_Captioner_Tool`

`Object_Detector_Tool`

Additional considerations: Consider the limitations of each tool, e.g., potential inaccuracies in object detection in complex scenes or object detection. Verify results and be aware of the need for supplementary tools for precise counting.

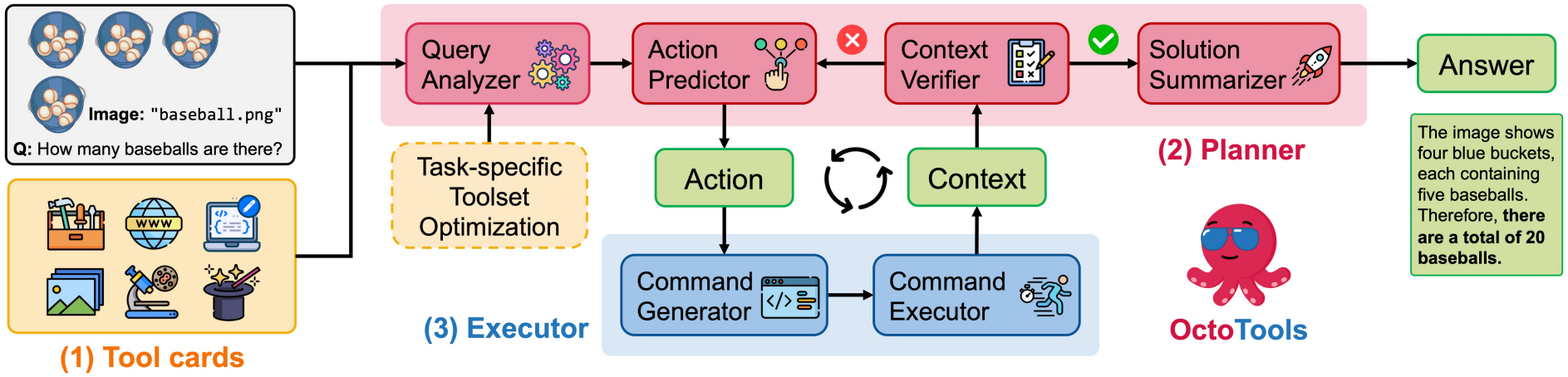


Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Object Detector Tool

tool_description="A tool that detects and saves objects in an image"

inputs={image,labels,threshold}

outputs={list: objects, scores, boxes}

demos=tool.execute(image="car.png", labels=["car"])

user_metadata={"limitation": "May not detect objects accurately."}

.....

Step 0: Query Analyzer

Summary: determine the total number of baseballs in the image.

Required skills:

1. understanding and interpreting visual content.
2. ability to operate and execute commands using the tools.
3. evaluating tool outputs and making decisions.

Relevant tools:
`Image_Captioner_Tool`
`Object_Detector_Tool`

Additional considerations:
Consider the limitations of each tool, e.g., potential inaccuracies in object detection in complex scenes or object detection. Verify results and be aware of the need for supplementary tools for precise counting.

Step 1: Action Predictor

Context: Image path: "`baseball.png`"

Sub-goal: Generate a description of the image to identify baseballs and relevant objects.

Tool name: `Image_Captioner_Tool`

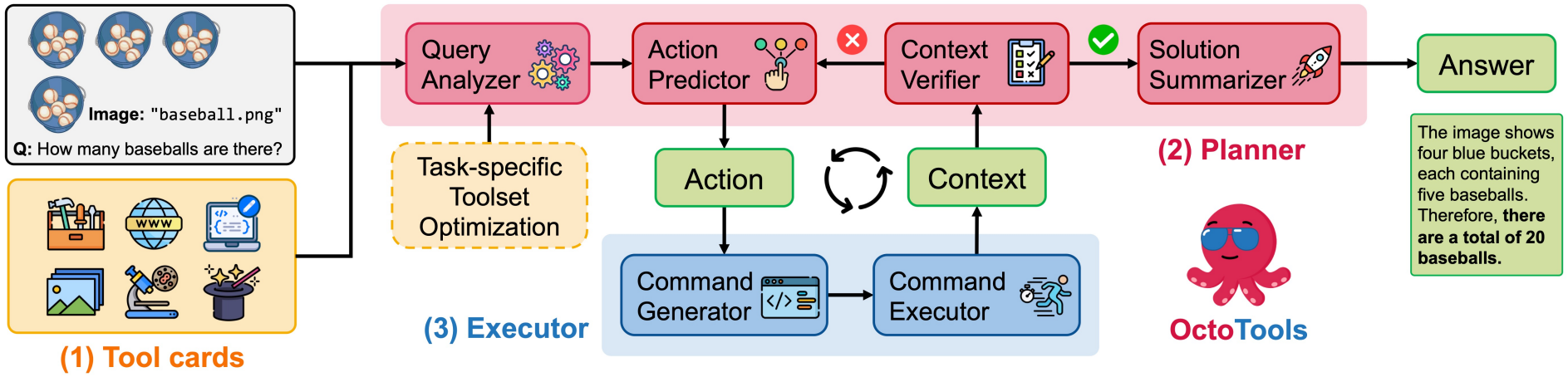


Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Object Detector Tool

tool_description="A tool that detects and saves objects in an image"

inputs={image,labels,threshold}

outputs={list: objects, scores, boxes}

demos=tool.execute(image="car.png", labels=["car"])

user_metadata={"limitation": "May not detect objects accurately."}

.....

Step 0: Query Analyzer

Summary: determine the total number of baseballs in the image.

Required skills:

1. understanding and interpreting visual content.
2. ability to operate and execute commands using the tools.
3. evaluating tool outputs and making decisions.

Relevant tools:
`Image_Captioner_Tool`
`Object_Detector_Tool`

Additional considerations:
Consider the limitations of each tool, e.g., potential inaccuracies in object detection in complex scenes or object detection. Verify results and be aware of the need for supplementary tools for precise counting.

Step 1: Action Predictor

Context: Image path: "baseball.png"

Sub-goal: Generate a description of the image to identify baseballs and relevant objects.

Tool name: `Image_Captioner_Tool`

Step 1: Command Generator

Analysis: The task requires using `Image Captioner Tool` to generate a description of an image to identify baseballs. The metadata specifies that it needs an image path and a prompt.

Command: `tool.execute(image="baseball.png", prompt="Describe this image.")`

Step 1: Command Executor

The image shows four blue buckets, each containing five baseballs. Each bucket has a handle on the side.

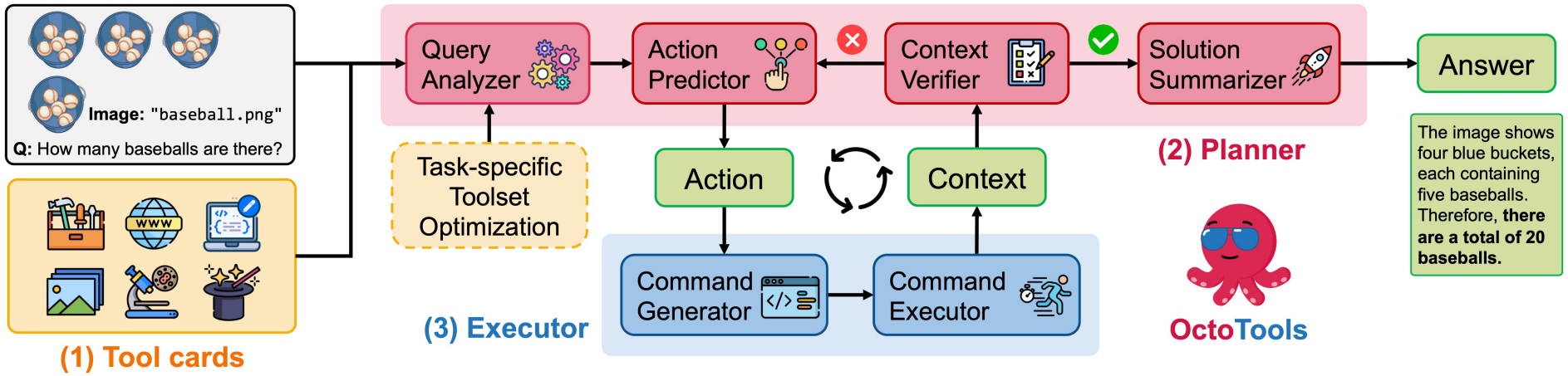


Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Object Detector Tool

tool_description="A tool that detects and saves objects in an image"

inputs={image,labels,threshold}

outputs={list: objects, scores, boxes}

demos=tool.execute(image="car.png", labels=["car"])

user_metadata={"limitation": "May not detect objects accurately."}

.....

Step 0: Query Analyzer

Summary: determine the total number of baseballs in the image.

Required skills:

1. understanding and interpreting visual content.
2. ability to operate and execute commands using the tools.
3. evaluating tool outputs and making decisions.

Relevant tools:
`Image_Captioner_Tool`
`Object_Detector_Tool`

Additional considerations:
Consider the limitations of each tool, e.g., potential inaccuracies in object detection in complex scenes or object detection. Verify results and be aware of the need for supplementary tools for precise counting.

Step 1: Action Predictor

Context: Image path: "`baseball.png`"

Sub-goal: Generate a description of the image to identify baseballs and relevant objects.

Tool name: `Image_Captioner_Tool`

Step 1: Command Generator

Analysis: The task requires using `Image Captioner Tool` to generate a description of an image to identify baseballs. The metadata specifies that it needs an image path and a prompt.

Command:
```
tool.execute(image="baseball.png",
prompt="Describe this image.")
```

Step 1: Command Executor

The image shows four blue buckets, each containing five baseballs. Each bucket has a handle on the side.

Step 1: Context Verifier

Completeness: Context provides a description of the image.

Inconsistencies: There are no inconsistencies.

Verification: Given limitations of `Image Captioner Tool` in counting, verify the count using `Object Detector Tool`.

Ambiguities: There are no ambiguities.

Conclusion: `CONTINUE`

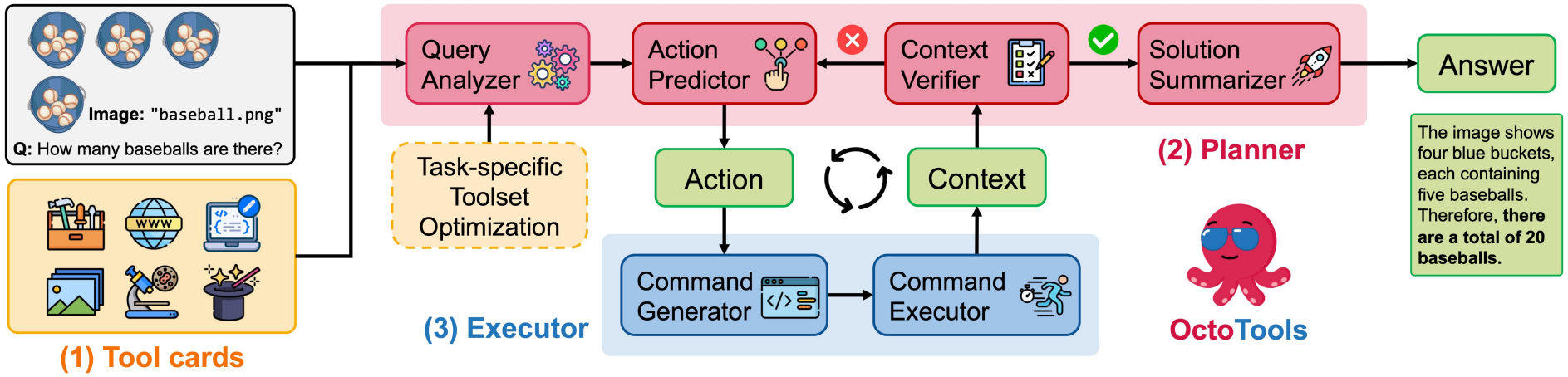


Image Captioner Tool

tool_description="A tool that generates captions for images."

inputs={image, prompt}

outputs={str: "The generated caption."}

demos=tool.execute(image="image.png", prompt="Describe the image.")

user_metadata={"limitation": "May make mistakes in complex scenes & counting."}

Object Detector Tool

tool_description="A tool that detects and saves objects in an image"

inputs={image,labels,threshold}

outputs={list: objects, scores, boxes}

demos=tool.execute(image="car.png", labels=["car"])

user_metadata={"limitation": "May not detect objects accurately."}

.....

Step 0: Query Analyzer

Summary: determine the total number of baseballs in the image.

Required skills:

1. understanding and interpreting visual content.
2. ability to operate and execute commands using the tools.
3. evaluating tool outputs and making decisions.

Relevant tools:
Image_Captioner_Tool
Object_Detector_Tool

Additional considerations:
Consider the limitations of each tool, e.g., potential inaccuracies in object detection in complex scenes or object detection. Verify results and be aware of the need for supplementary tools for precise counting.

Step 1: Action Predictor

Context: Image path: "baseball.png"
Sub-goal: Generate a description of the image to identify baseballs and relevant objects.
Tool name: Image_Captioner_Tool

Step 1: Command Generator

Analysis: The task requires using Image Captioner Tool to generate a description of an image to identify baseballs. The metadata specifies that it needs an image path and a prompt.
Command: tool.execute(image="baseball.png", prompt="Describe this image.")

Step 1: Command Executor

The image shows four blue buckets, each containing five baseballs. Each bucket has a handle on the side.

Step 1: Context Verifier

Completeness: Context provides a description of the image.
Inconsistencies: There are no inconsistencies.
Verification: Given limitations of Image Captioner Tool in counting, verify the count using Object Detector Tool.
Ambiguities: There are no ambiguities.
Conclusion: CONTINUE

Step 2: Action Predictor

Context: Image path: "baseball.png"
Sub-goal: Detect and count the number of baseballs.
Tool name: Object_Detector_Tool

Step 2: Command Generator

Analysis: The task requires detecting baseballs using Object Detector Tool. It needs an image path and a 'baseball' label.
Command: tool.execute(image="baseball.png", labels=["baseball"])

Step 2: Command Executor


```
{
   "label": "baseball",
   "confidence score": 0.69,
   "box": [558, 48, 615, 107],
   "saved_path": "baseball_1.png"
},
...
```


Step 2: Context Verifier

Completeness: Context addresses the query using tools.
Inconsistencies: There are no inconsistencies in outputs.
Verification: Consistent results reduce need for verification.
Ambiguities: There are no ambiguities in the results.
Conclusion: STOP

Prediction from the OctoTool Framework

Planner Components | Executor Components | Final Answer

Step 0: Initial Plan from Query Analyzer

Step 1: Action from Action Predictor

Step 1: Command from Command Generator

Step 1: Result from Command Executor

Step 1: Verification from Context Verifier

Step 2: Action from Action Predictor

Step 2: Command from Command Generator

Step 2: Result from Command Executor

Step 2: Verification from Context Verifier

Full Trajectory in the Context

Summary from Solution Summarizer

Final Answer

Summary: The query aimed to determine the number of baseballs in an image. The process involved using two tools: an `Image_Captioner_Tool` to describe the image and an `Object_Detector_Tool` to count the baseballs.

Detailed Analysis

Step 1: `Image_Captioner_Tool`

- Result: The image contains four blue buckets, each with five baseballs, arranged in a grid pattern.

Step 2: `Object_Detector_Tool`

- Result: Detected 20 baseballs with varying confidence scores.

Key Findings

- The image contains a total of 20 baseballs, distributed evenly across four buckets.
- Each bucket contains five baseballs, as confirmed by both tools.

Answer to the Query

The image shows four blue buckets, each containing five baseballs. Therefore, **there are a total of 20 baseballs.**

Additional Insights

- The consistent results from both tools reinforce the accuracy of the analysis.
- The arrangement of the buckets and baseballs is clear and well-organized, aiding in accurate detection.

Wrong Prediction from GPT-4o

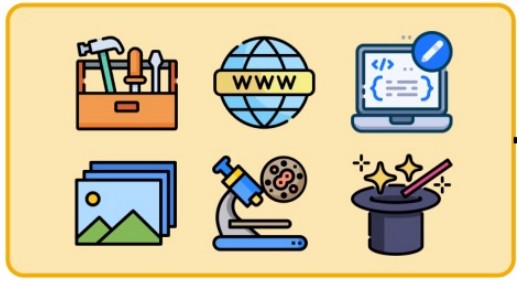
There are a total of 19 baseballs across the four buckets:

- The first bucket (top left) contains 4 baseballs.
- The second bucket (top right) contains 5 baseballs.
- The third bucket (bottom left) contains 4 baseballs.
- The fourth bucket (bottom right) contains 6 baseballs.

Adding them together: 4 + 5 + 4 + 6 = 19 baseballs.

https://octotools.github.io/#visualization

Task-Specific Toolset Optimization



Tool cards

A few examples

Validation

Google_Search
Wikipedia_Knowledge_Searcher

Python_Code_Generator

Relevant_Patch_Zoomer

Algorithm 1 Task-specific Toolset Optimization

Input: Toolbox $\mathcal{D} = \{d_i\}_{i=1}^{n}$, base toolset $\mathcal{D}_{\text{base}}$
Output: Optimized toolset $\mathcal{D}^*$
Stage 1: Baseline setup
$\text{Acc}_{\text{baseline}} \leftarrow \text{Acc}(\mathcal{D}_{\text{base}})$
# Stage 2: Individual tool evaluation
for each d_i in D **such that** $d_i \notin \mathcal{D}_{\text{base}}$ **do**
  $\mathcal{D}_i \leftarrow \mathcal{D}_{\text{base}} \cup \{d_i\}$
  $\text{Acc}_i \leftarrow \text{Acc}(\mathcal{D}_i)$
  $\Delta_{d_i} \leftarrow \text{Acc}_i - \text{Acc}_{\text{baseline}}$
  if $\Delta_{d_i} > 0$ **then**
    $\mathcal{D}_{\text{beneficial}} \leftarrow \mathcal{D}_{\text{beneficial}} \cup \{d_i\}$
  end if
end for
# Stage 3: Select optimized toolset
$\mathcal{D}^* \leftarrow \mathcal{D}_{\text{beneficial}} \cup \mathcal{D}_{\text{base}}$
Return $\mathcal{D}^*$

Comprehensive Evaluations Across 16 Tasks

Datasets	Modality	Domain				
VQA 2.0	Vision	General	✓			
Hallusion-VD	Vision	General	✓			
AlgoPuzzleVQA	Vision	General	✓			✓
PuzzleVQA	Vision	General	✓			
Game of 24	Text	Mathematical		✓		✓
Omni-MATH	Text	Mathematical		✓	✓	
CLEVR-Math	Vision	Mathematical	✓	✓		
MathVista	Vision	Mathematical	✓	✓	✓	✓
GPQA	Text	Scientific			✓	✓
MMLU-Pro	Text	Scientific			✓	✓
SciFIBench	Vision	Scientific	✓		✓	
MedQA	Text	Medical			✓	
PathCLS	Vision	Medical	✓		✓	
PathVQA	Vision	Medical	✓		✓	✓
SLAKE	Vision	Medical	✓		✓	
GAIA-Text	Text	Agentic		✓	✓	✓

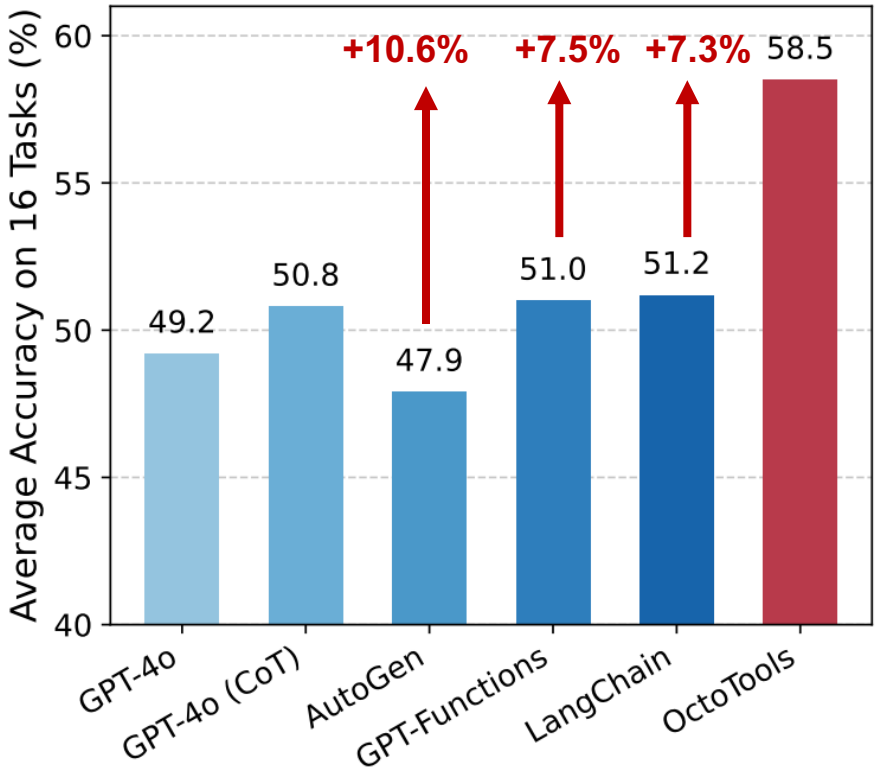
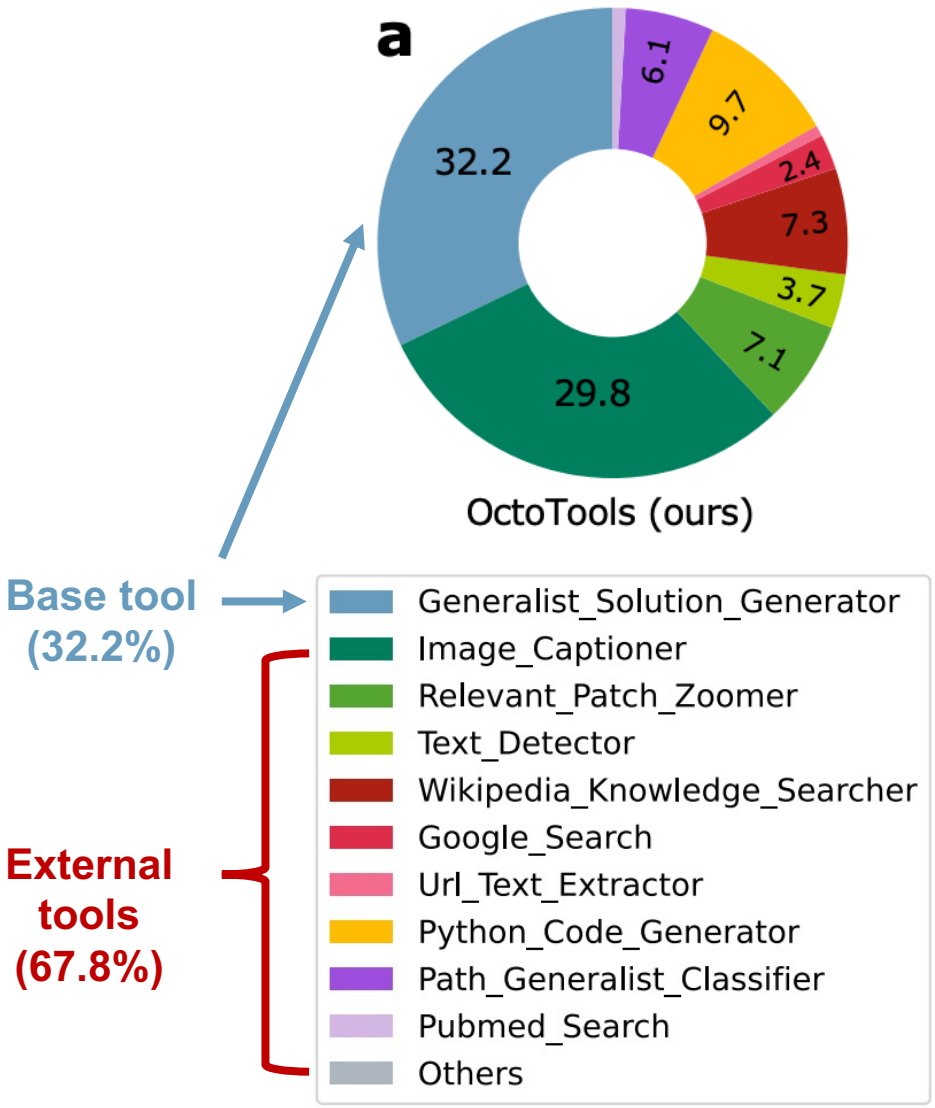
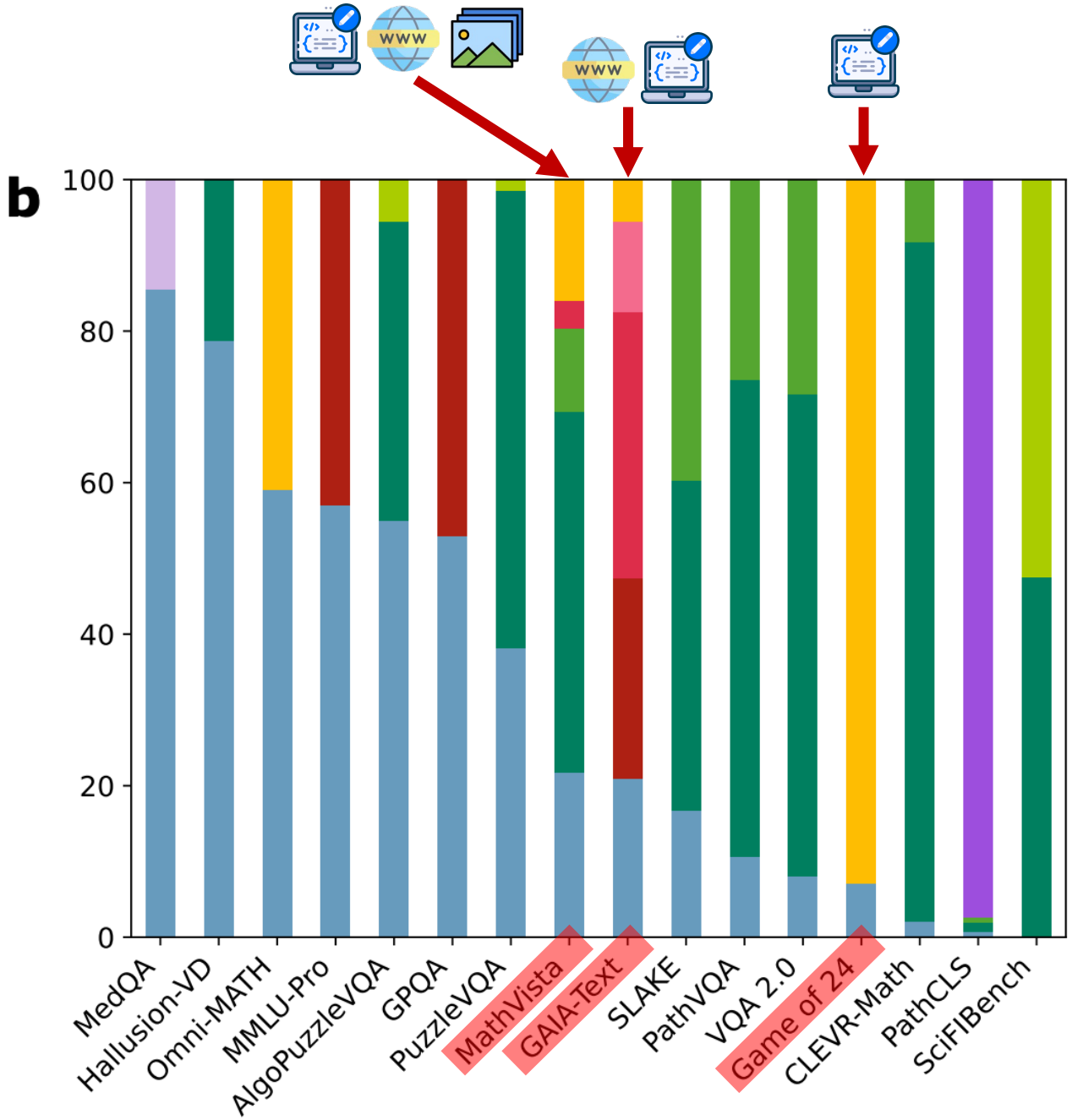


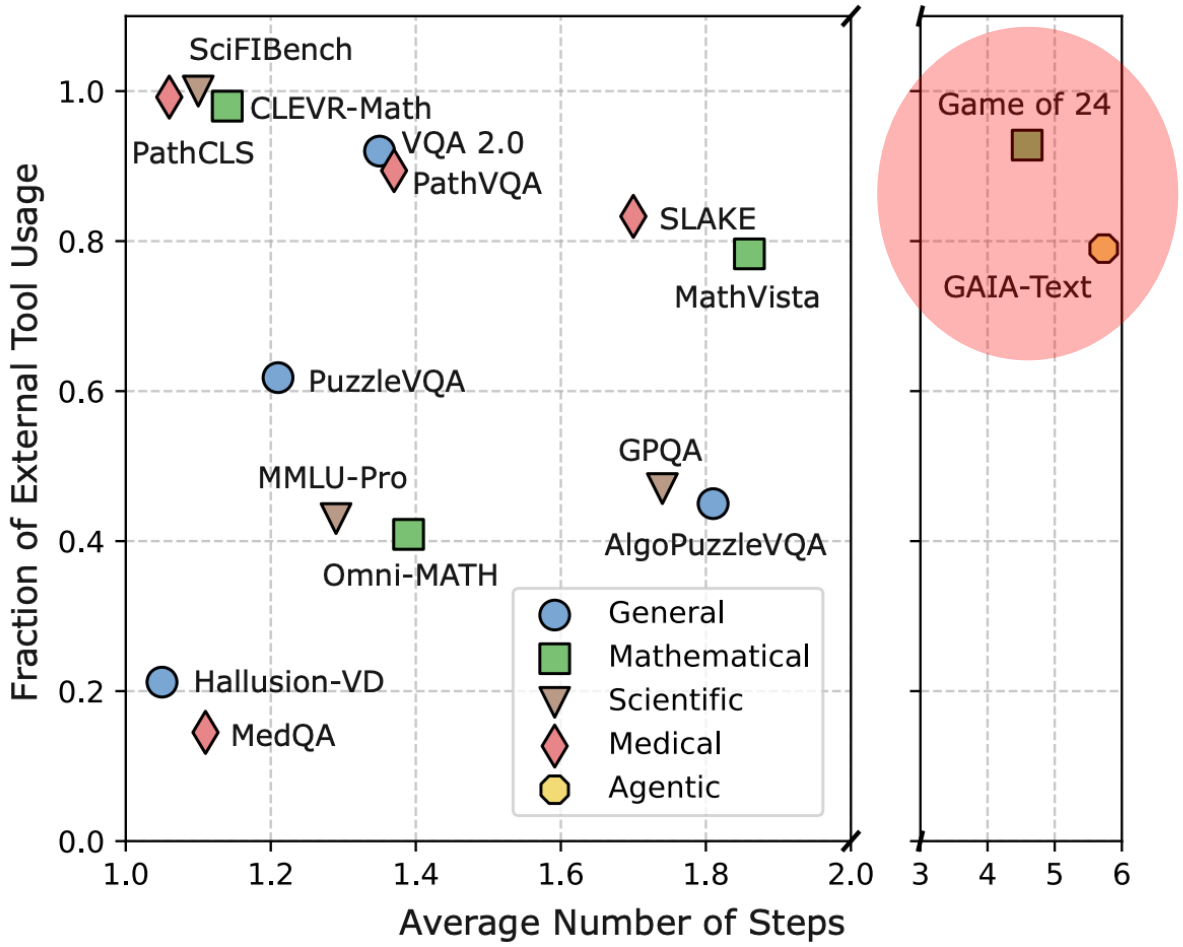
Figure 1. Performance comparison across 16 benchmarks. On average, our OctoTools system achieves an average accuracy gain of 9.3% over GPT-4o and 7.3% over LangChain.

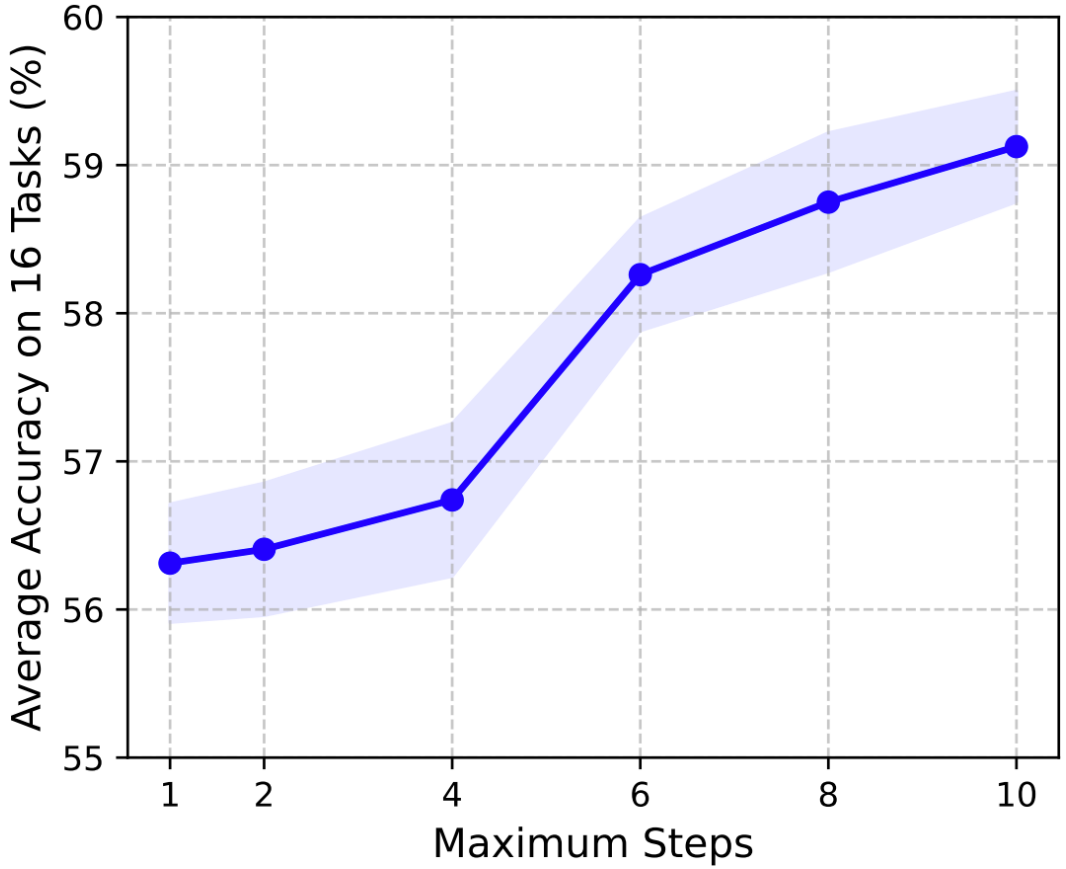
Tool Usage Study





External Tool Usage and Reasoning Steps





Spaces | OctoTools/octotools | like 123 | Running on A10G | Logs | App | Files | Community | Settings

Chat with OctoTools: An Agentic Framework with Extensive Tools for Complex Reasoning

OctoTools is a training-free, user-friendly, and easily extensible open-source agentic framework designed to tackle complex reasoning across diverse domains. It introduces standardized **tool cards** to encapsulate tool functionality, a **planner** for both high-level and low-level planning, and an **executor** to carry out tool usage.

Website | Github | arXiv | Paper | Tool Cards | Example Visualizations | Discord

LLM Model: gpt-4o

Max Steps: 8 (1 – 10)

Max Time (seconds): 240 (60 – 300)

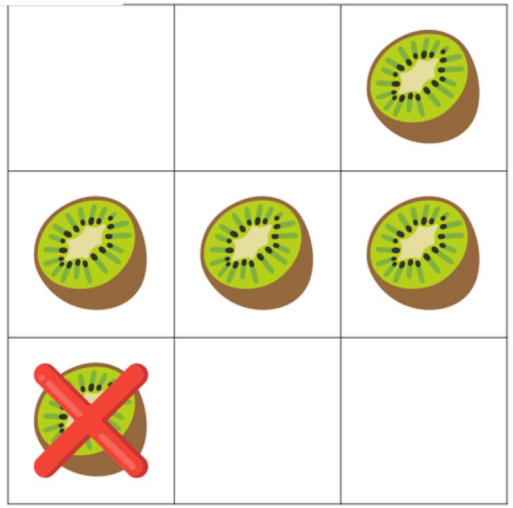
Selected Tools

- [ ] Generalist_Solution_Generator_Tool
- [x] Image_Captioner_Tool
- [ ] Object_Detector_Tool
- [ ] Relevant_Patch_Zoomer_Tool
- [ ] Text_Detector_Tool
- [ ] Python_Code_Generator_Tool
- [ ] ArXiv_Paper_Searcher_Tool
- [ ] Google_Search_Tool
- [ ] Nature_News_Fetcher_Tool
- [ ] Pubmed_Search_Tool
- [ ] URL_Text_Extractor_Tool
- [ ] Wikipedia_Knowledge_Searcher_Tool

Select All Tools

Clear All Tools

Upload an Image (Optional)



Question (Required)

You are given a 3 x 3 grid in which each cell can contain either no kiwi, one fresh kiwi, or one rotten kiwi. Every minute, any fresh kiwi that is 4-directionally adjacent to a rotten kiwi also becomes rotten. What is the minimum number of minutes that must elapse until no cell has a fresh kiwi?

Submit and Run

Step-wise Problem-Solving Output

s that must elapse until no cell has a fresh kiwi?

Image Uploaded

Reasoning Steps from OctoTools (Deep Thinking...)

- Step 0: Query Analysis
- Step 1: Action Prediction (Image_Captioner_Tool)
- Step 1: Command Generation (Image_Captioner_Tool)
- Step 1: Command Execution (Image_Captioner_Tool)
- Step 1: Context Verification

Analysis: The task involves analyzing a 3x3 grid from an image to determine how fresh kiwis become rotten over time. The selected tool is the Image_Captioner_Tool, which can generate captions based on the image content. This tool requires an image path and optionally a prompt to guide the captioning. The image path is provided in the query as 'solver_cache/20250219_113812_9a7459c9/query_image.jpg'. The sub-goal is to manually analyze the grid, so the prompt should focus on describing the grid's initial state, specifically noting the positions of fresh and rotten kiwis.

Conclusion: STOP ✅

Upvote | Downvote

Comment (Type and press Enter to submit.)

Feel free to add any comments here. Thanks for using OctoTools!

Try these examples with suggested tools.

https://huggingface.co/spaces/OctoTools/octotools

Examples

Category	Upload an Image (Optional)	Question (Required)	Selected Tools	Reference Answer
Logical Reasoning		How many r letters are in the word strawberry?	Generalist_Solution_Generator_Tool, Python_Code_Generator_Tool	3

Broad Community Attention

OctoTools

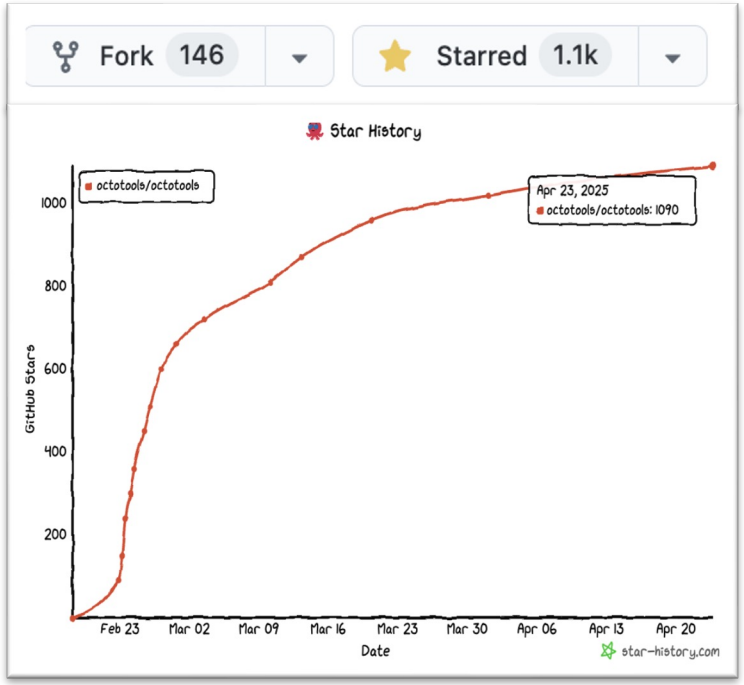
An Agentic Framework with Extensible Tools for Complex Reasoning

arXiv | Code | PyPI | Demo | Visualize | YouTube | Twitter | Slack

gradio

All time visits

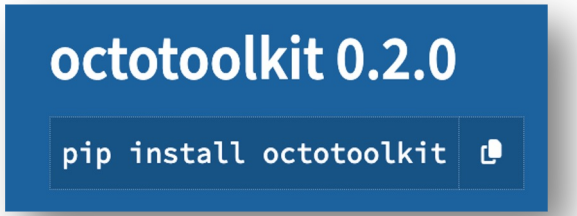
20,542



Support for Multiple LLMs

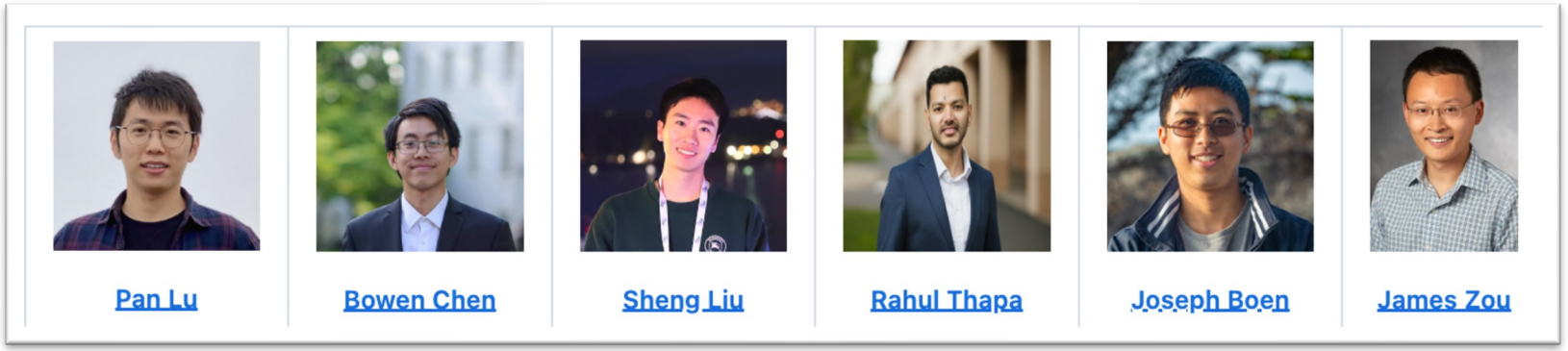
Model Family	Engines (Multi-modal)	Engines (Text-Only)	Official Model List
OpenAI	`gpt-4-turbo`, `gpt-4o`, `gpt-4o-mini`, `gpt-4.1`, `gpt-4.1-mini`, `gpt-4.1-nano`, `o1`, `o3`, `o1-pro`, `o4-mini`	`gpt-3.5-turbo`, `gpt-4`, `o1-mini`, `o3-mini`	OpenAI Models
Anthropic	`claude-3-haiku-20240307`, `claude-3-sonnet-20240229`, `claude-3-opus-20240229`, `claude-3-5-sonnet-20240620`, `claude-3-5-sonnet-20241022`, `claude-3-5-haiku-20241022`, `claude-3-7-sonnet-20250219`		Anthropic Models
TogetherAI	Most multi-modal models, including `meta-llama/Llama-4-Scout-17B-16E-Instruct`, `Qwen/QwQ-32B`, `Qwen/Qwen2-VL-72B-Instruct`	Most text-only models, including `meta-llama/Llama-3-70b-chat-hf`, `Qwen/Qwen2-72B-Instruct`	TogetherAI Models
DeepSeek		`deepseek-chat`, `deepseek-reasoner`	DeepSeek Models
Gemini	`gemini-1.5-pro`, `gemini-1.5-flash-8b`, `gemini-1.5-flash`, `gemini-2.0-flash-lite`, `gemini-2.0-flash`, `gemini-2.5-pro-preview-03-25`		Gemini Models
Grok	`grok-2-vision-1212`, `grok-2-vision`, `grok-2-vision-latest`	`grok-3-mini-fast-beta`, `grok-3-mini-fast`, `grok-3-mini-fast-latest`, `grok-3-mini-beta`, `grok-3-mini`, `grok-3-mini-latest`, `grok-3-fast-beta`, `grok-3-fast`, `grok-3-fast-latest`, `grok-3-beta`, `grok-3`, `grok-3-latest`	Grok Models

PyPI Package Support




```python
# Import the solver
from octotools.solver import construct_solver

# Set the LLM engine name
llm_engine_name = "gpt-4o"

# Construct the solver
solver = construct_solver(llm_engine_name=llm_engine_name)

# Solve the user query
output = solver.solve("What is the capital of France?")
print(output["direct_output"])

# Similarly, you could pass in a photo
output = solver.solve("What is the name of this item in French?", image_path="<PATH_TO_IMG>")
print(output["direct_output"])
```

Join the OctoTools Developer Community!



slack

https://octotools.github.io/