AI Agents

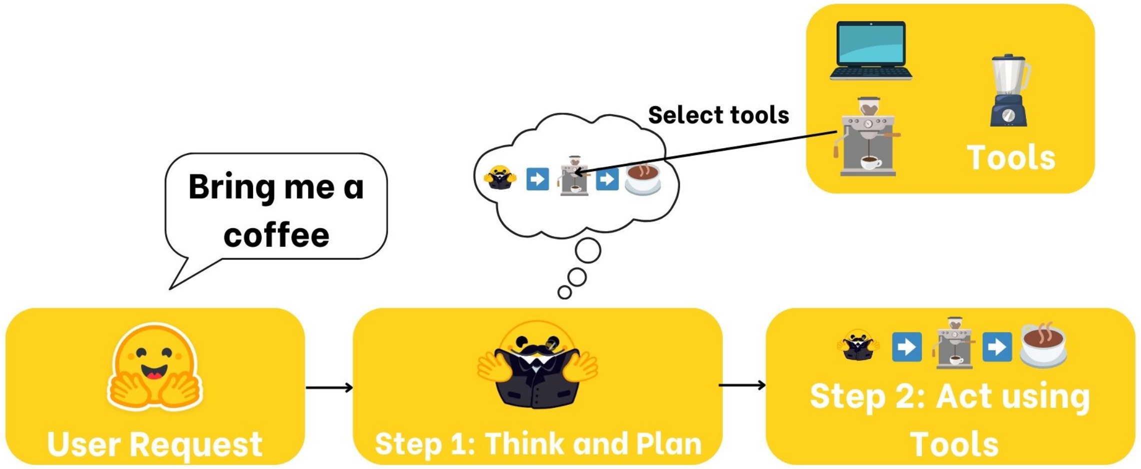
Tools, and Frameworks for Building Autonomous Workflows

Sergio Paniego Blanco

Machine Learning Engineer @ Hugging Face

What is an AI Agent?

System that leverages an AI model to interact with its environment in order to achieve a user-defined objective. It combines reasoning, planning, and the execution of actions (often via external tools) to fulfill tasks.



AI components

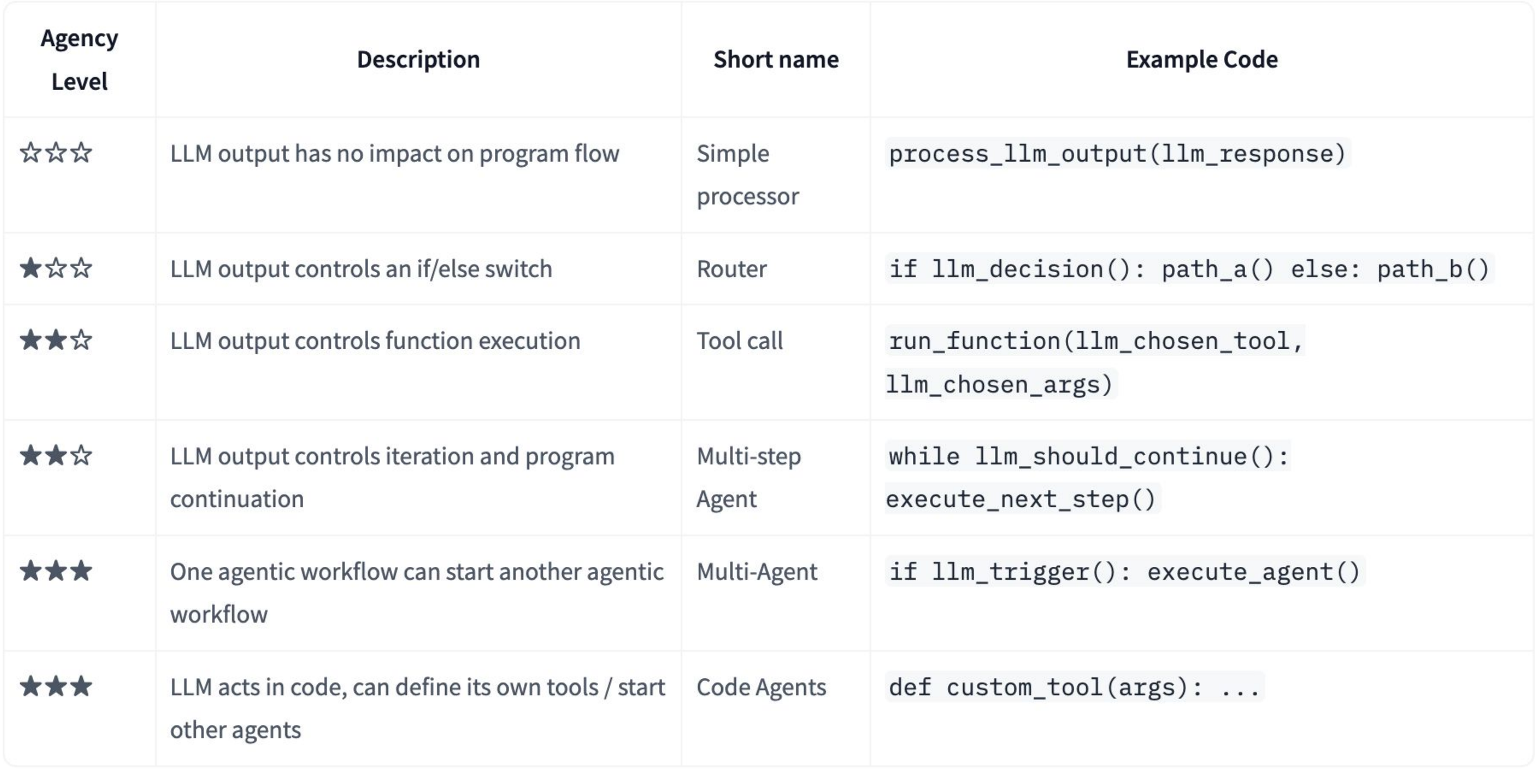
- Brain (AI model)
- Body (capabilities and tools)

AI models: LLM (text-to-text) or VLM (image-text-to-text).

- GPT 4 from OpenAI, Llama from Meta, Gemini from Google…

How do they take actions? → they have access to tools!

Levels of Agency

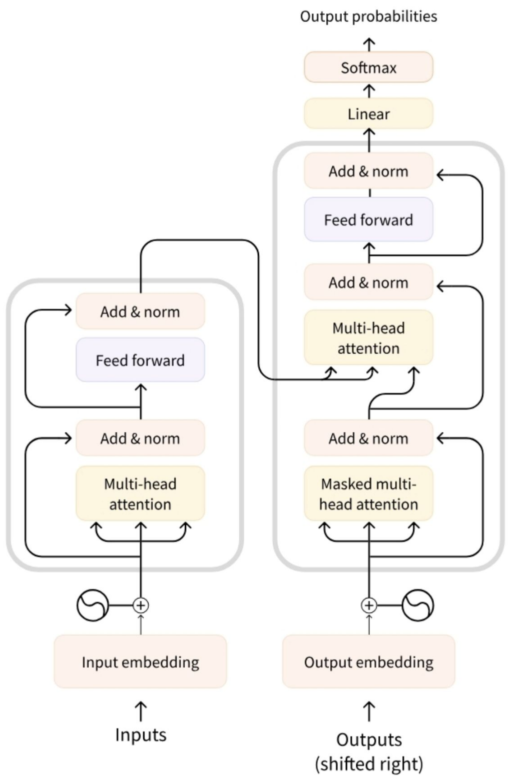


Agency Level	Description	Short name	Example Code
☆☆☆	LLM output has no impact on program flow	Simple processor	`process_llm_output(llm_response)`
★☆☆	LLM output controls an if/else switch	Router	`if llm_decision(): path_a() else: path_b()`
★★☆	LLM output controls function execution	Tool call	`run_function(llm_chosen_tool, llm_chosen_args)`
★★☆	LLM output controls iteration and program continuation	Multi-step Agent	`while llm_should_continue(): execute_next_step()`
★★★	One agentic workflow can start another agentic workflow	Multi-Agent	`if llm_trigger(): execute_agent()`
★★★	LLM acts in code, can define its own tools / start other agents	Code Agents	`def custom_tool(args): ...`

What is an LLM?

AI system that excels at understanding and generating human language.

- Trained on vast amounts of text data (multimodal too).
- They use transformers 🤖.
  - Encoders
  - Decoders
  - Seq2Seq (encoder-decoder)



Types of transformers

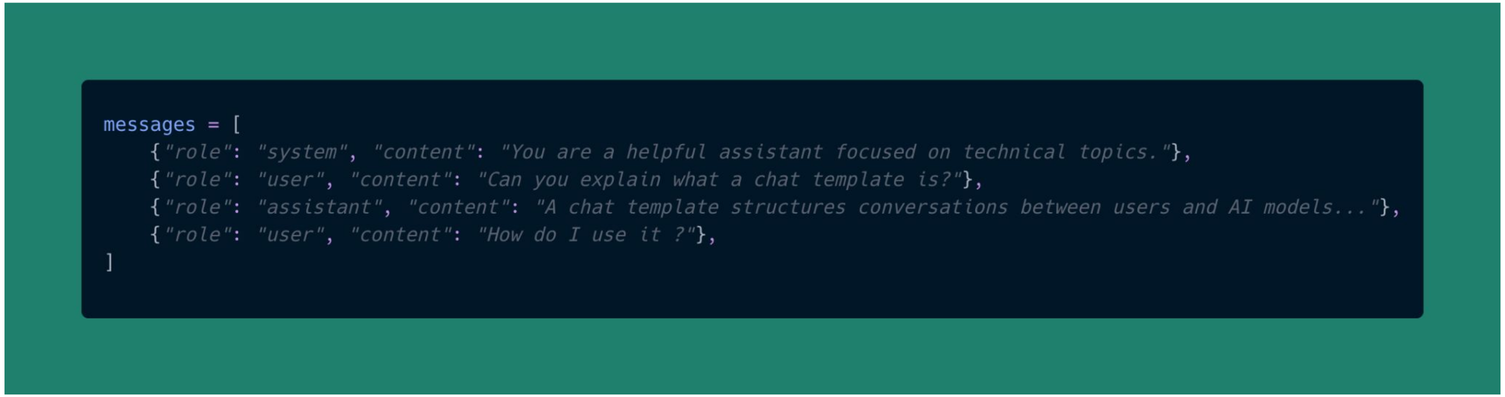
- **Encoders:**
  - Example: BERT
  - Use case: text classification, semantic search, NER.
  - Size: millions of params.
- **Decoders:**
  - Example: Llama, Gemini, GPT…
  - Use case: text generation, chatbots, code generation.
  - Size: billions of params.
- **Seq2Seq (encoder-decoder):**
  - Example: T5, BART
  - Use case: translation, summarization
  - Size: millions of params
- Typical LLM systems are decoders: DeepSeek R1, GPT, Gemini, Llama, Gemma, SmolLM2

Goal of LLMs

- **Goal of LLMs:** predict the next token, given a sequence of previous tokens (autoregressive).

Base models vs Instruct models

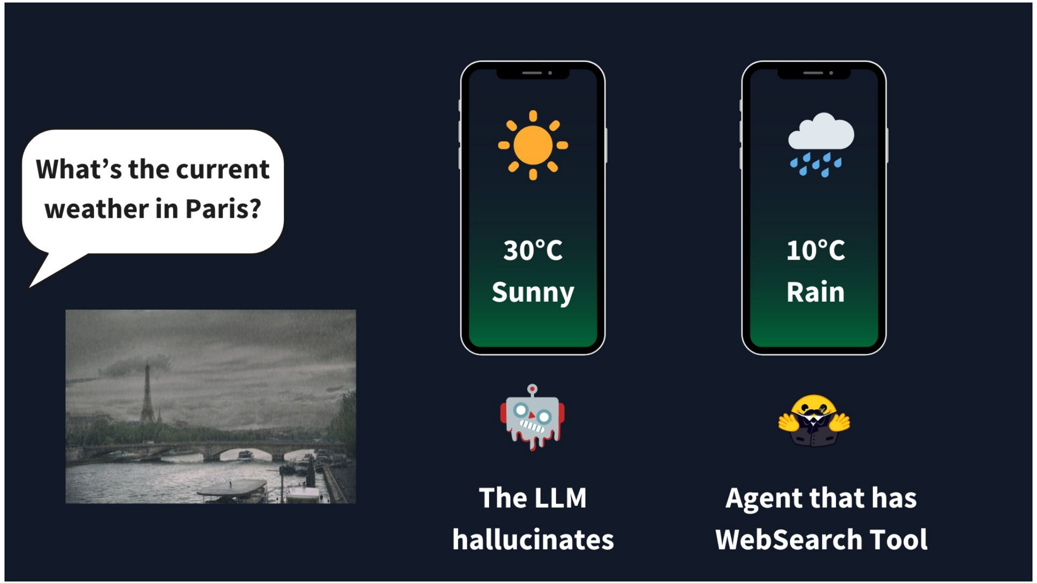
- **Base model:** trained on raw data to predict the next token.
- **Instruct model:** fine-tuned to follow instructions and engage in conversations.
    - They use chat templates



```
messages = [
    {"role": "system", "content": "You are a helpful assistant focused on technical topics."},
    {"role": "user", "content": "Can you explain what a chat template is?"},
    {"role": "assistant", "content": "A chat template structures conversations between users and AI models..."},
    {"role": "user", "content": "How do I use it ?"},
]
```

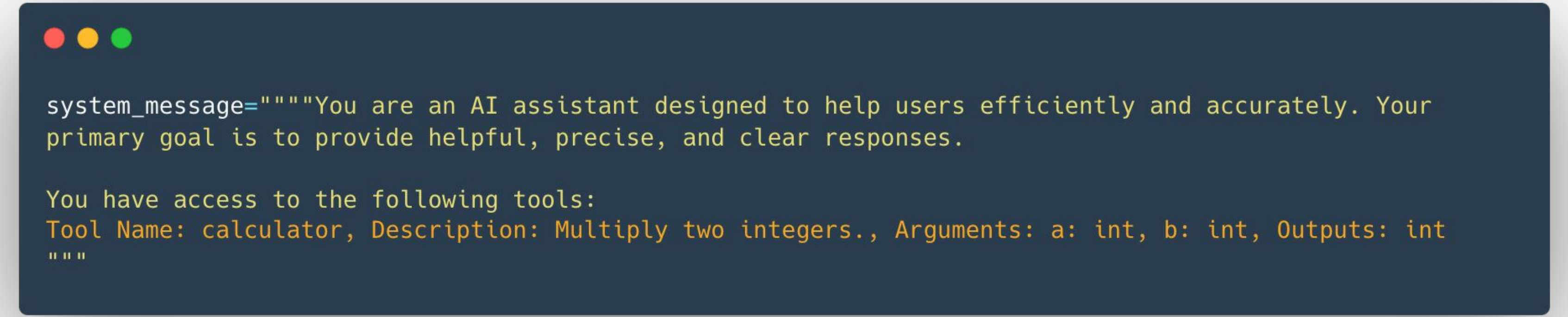
What are tools?

- Function given to the LLM with a clear goal.
  - Web search, Image generation, Retrieval, API interface


What's the current weather in Paris?
30°C Sunny
10°C Rain
The LLM hallucinates
Agent that has WebSearch Tool

What are tools?

- Function given to the LLM with a clear goal.
    - Web search, Image generation, Retrieval, API interface



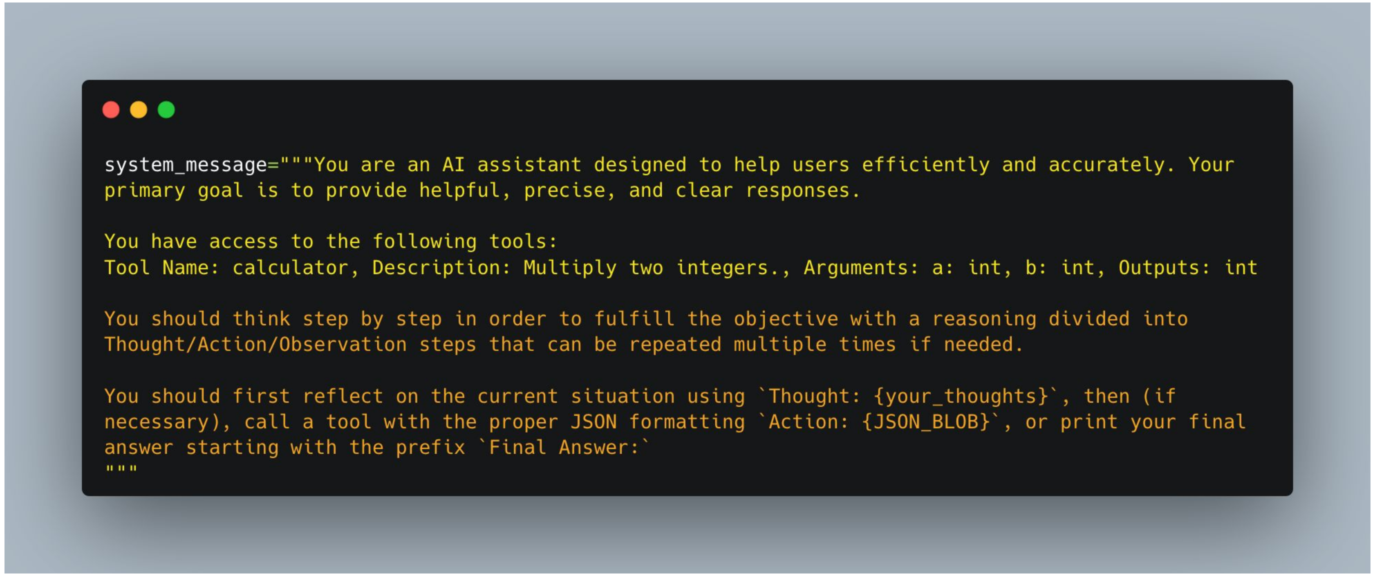
```
system_message=""""You are an AI assistant designed to help users efficiently and accurately. Your
primary goal is to provide helpful, precise, and clear responses.

You have access to the following tools:
Tool Name: calculator, Description: Multiply two integers., Arguments: a: int, b: int, Outputs: int
"""
```

AI Agent workflow

- Thinking (Thought) → Acting (Act) and observing (Observe)

AI Agent workflow

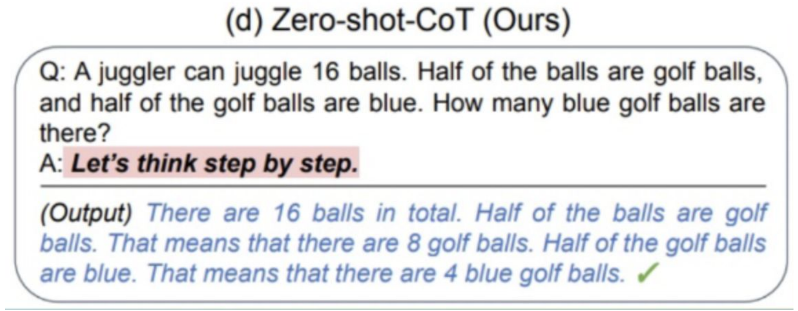
- Thinking (Thought) → Acting (Act) and observing (Observe)



```
system_message="""You are an AI assistant designed to help users efficiently and accurately. Your
primary goal is to provide helpful, precise, and clear responses.

You have access to the following tools:
Tool Name: calculator, Description: Multiply two integers., Arguments: a: int, b: int, Outputs: int

You should think step by step in order to fulfill the objective with a reasoning divided into
Thought/Action/Observation steps that can be repeated multiple times if needed.

You should first reflect on the current situation using `Thought: {your_thoughts}`, then (if
necessary), call a tool with the proper JSON formatting `Action: {JSON_BLOB}`, or print your final
answer starting with the prefix `Final Answer:`
"""
```

Thought: Internal Reasoning and ReAct

- Thoughts represent the Agent's internal reasoning and planning processes to solve the task
- **ReAct approach:** "Reasoning" (Think) + "Acting" (Act). Prompting technique that appends "Let's think step by step" before letting the LLM decode the next tokens.



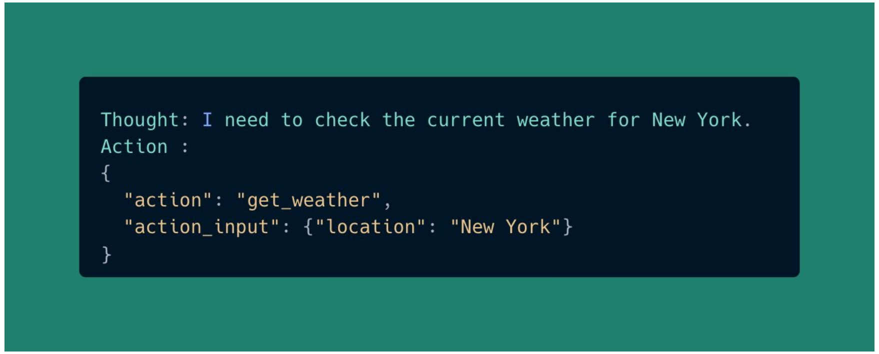
(d) Zero-shot-CoT (Ours)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) *There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls.* ✓

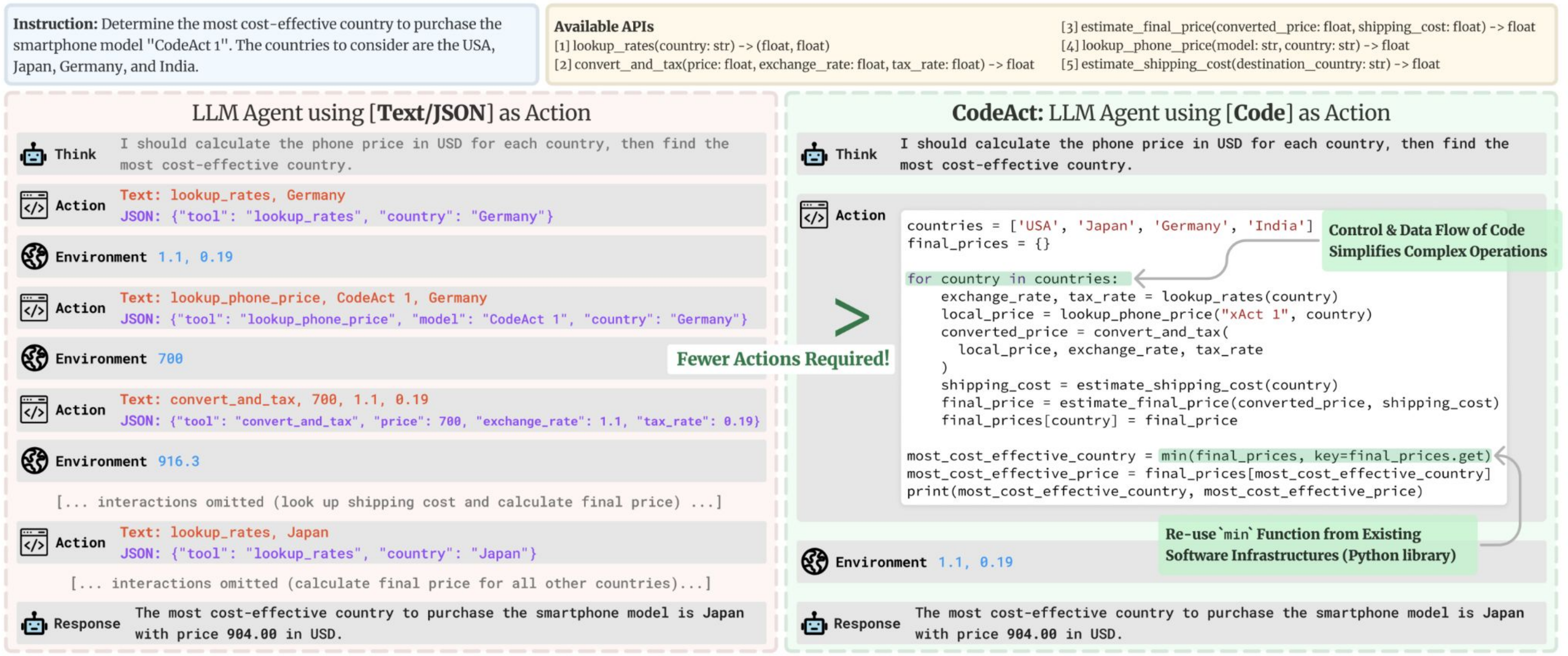
Actions: Engaging with environment

- Actions are concrete steps an AI agent takes to interact with its environment.
- Types of Agent Actions:
  - **JSON Agent:** action is specified in JSON format.
  - **Code Agent:** agent writes code block that is interpreted externally.



```
Thought: I need to check the current weather for New York.
Action :
{
  "action": "get_weather",
  "action_input": {"location": "New York"}
}
```

Actions: Engaging with environment



Instruction: Determine the most cost-effective country to purchase the smartphone model "CodeAct 1". The countries to consider are the USA, Japan, Germany, and India.

Available APIs
[1] lookup_rates(country: str) -> (float, float)
[2] convert_and_tax(price: float, exchange_rate: float, tax_rate: float) -> float
[3] estimate_final_price(converted_price: float, shipping_cost: float) -> float
[4] lookup_phone_price(model: str, country: str) -> float
[5] estimate_shipping_cost(destination_country: str) -> float

LLM Agent using [Text/JSON] as Action

```
Think        I should calculate the phone price in USD for each country, then find the
             most cost-effective country.

Action       Text: lookup_rates, Germany
             JSON: {"tool": "lookup_rates", "country": "Germany"}

Environment  1.1, 0.19

Action       Text: lookup_phone_price, CodeAct 1, Germany
             JSON: {"tool": "lookup_phone_price", "model": "CodeAct 1", "country": "Germany"}

Environment  700

Action       Text: convert_and_tax, 700, 1.1, 0.19
             JSON: {"tool": "convert_and_tax", "price": 700, "exchange_rate": 1.1, "tax_rate": 0.19}

Environment  916.3

     [... interactions omitted (look up shipping cost and calculate final price) ...]

Action       Text: lookup_rates, Japan
             JSON: {"tool": "lookup_rates", "country": "Japan"}

      [... interactions omitted (calculate final price for all other countries)...]

Response     The most cost-effective country to purchase the smartphone model is Japan
             with price 904.00 in USD.
```

Fewer Actions Required!

>

CodeAct: LLM Agent using [Code] as Action

```
Think        I should calculate the phone price in USD for each country, then find the
             most cost-effective country.

Action
countries = ['USA', 'Japan', 'Germany', 'India']
final_prices = {}

for country in countries:
    exchange_rate, tax_rate = lookup_rates(country)
    local_price = lookup_phone_price("xAct 1", country)
    converted_price = convert_and_tax(
      local_price, exchange_rate, tax_rate
    )
    shipping_cost = estimate_shipping_cost(country)
    final_price = estimate_final_price(converted_price, shipping_cost)
    final_prices[country] = final_price

most_cost_effective_country = min(final_prices, key=final_prices.get)
most_cost_effective_price = final_prices[most_cost_effective_country]
print(most_cost_effective_country, most_cost_effective_price)

Environment  1.1, 0.19

Response     The most cost-effective country to purchase the smartphone model is Japan
             with price 904.00 in USD.
```

Control & Data Flow of Code Simplifies Complex Operations

Re-use `min` Function from Existing Software Infrastructures (Python library)

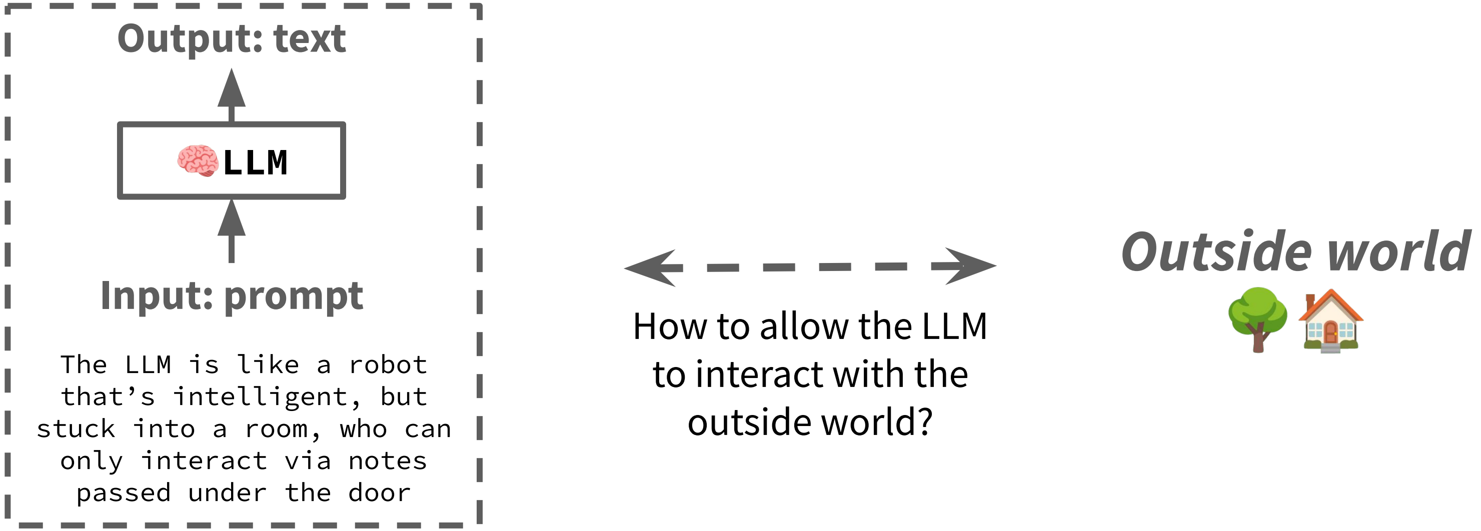
Observe: Integrating Feedback to Reflect and Adapt

- How an agent perceives the consequences of its actions.
- Observation phase:
    - Agent collects feedback
    - Appends results
    - Adapts its strategy
- How are results appended?
    - Parse the action
    - Execute the action
    - Append the result as an observation

ReAct agent

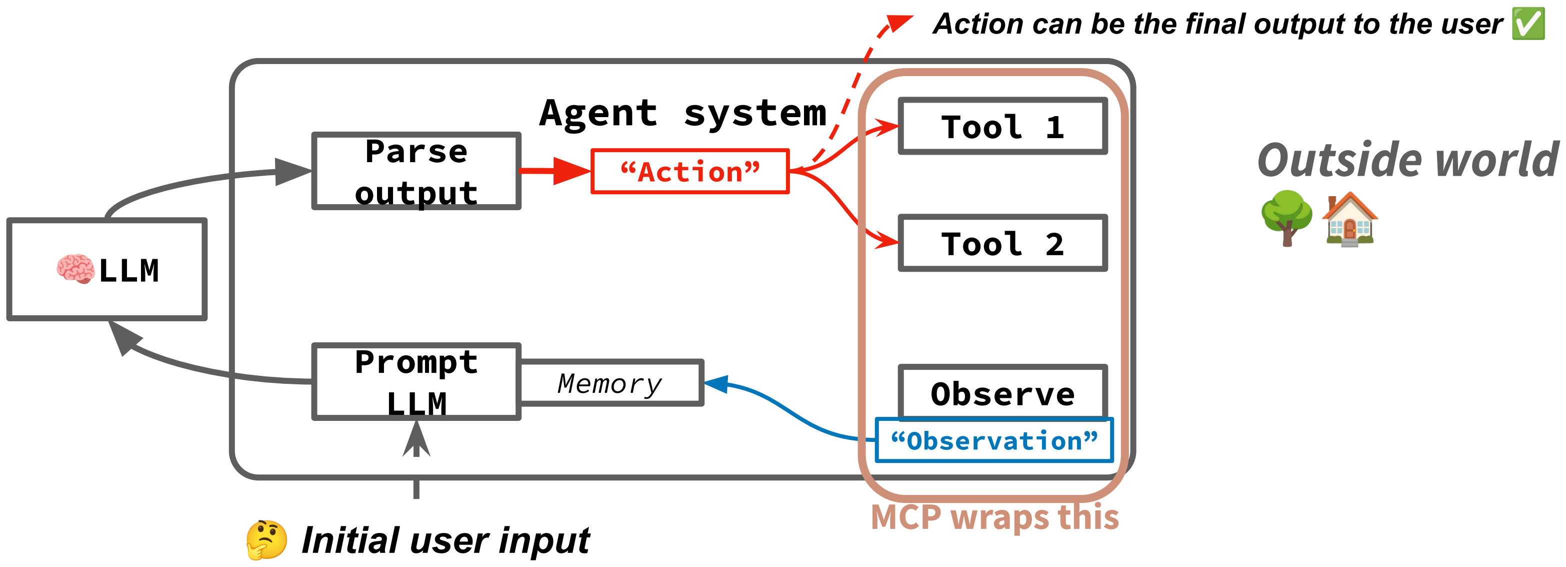
Why an agent?

- Need to de-isolate an LLM from the world.


Output: text
🧠LLM
Input: prompt
The LLM is like a robot that's intelligent, but stuck into a room, who can only interact via notes passed under the door
How to allow the LLM to interact with the outside world?
Outside world

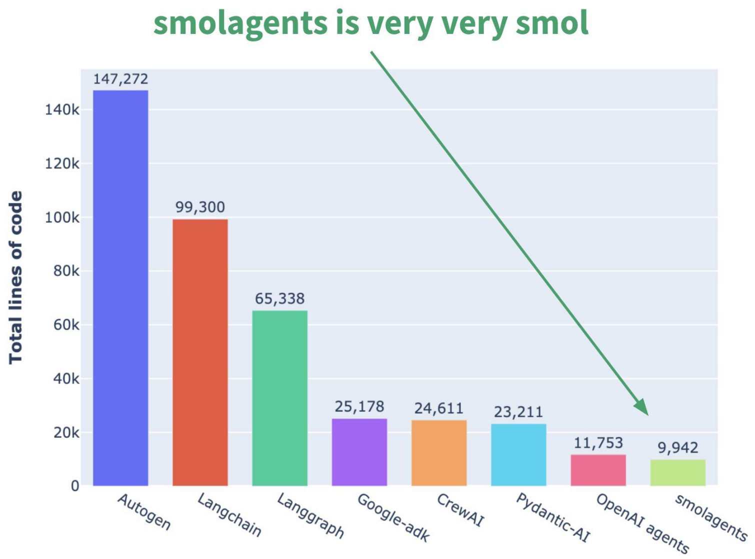
What is MCP?

- Protocol that proposes an easy wrapper on tools (and prompts)



Agentic Frameworks

- smolagents by Hugging Face
- LangGraph by LangChain
- Llama-Index
- OpenAI Agents SDK by OpenAI
- Autogen by Microsoft



smolagents

LlamaIndex

LangGraph

AG

Why smolagents

- **smolagents** is a simple yet powerful framework for building AI agents.
- **Advantages:**
  - Simplicity
  - Flexible LLM Support (transformers, HfApiModel, vLLM, OpenAI…)
  - Code-First Approach
  - HF Hub integration

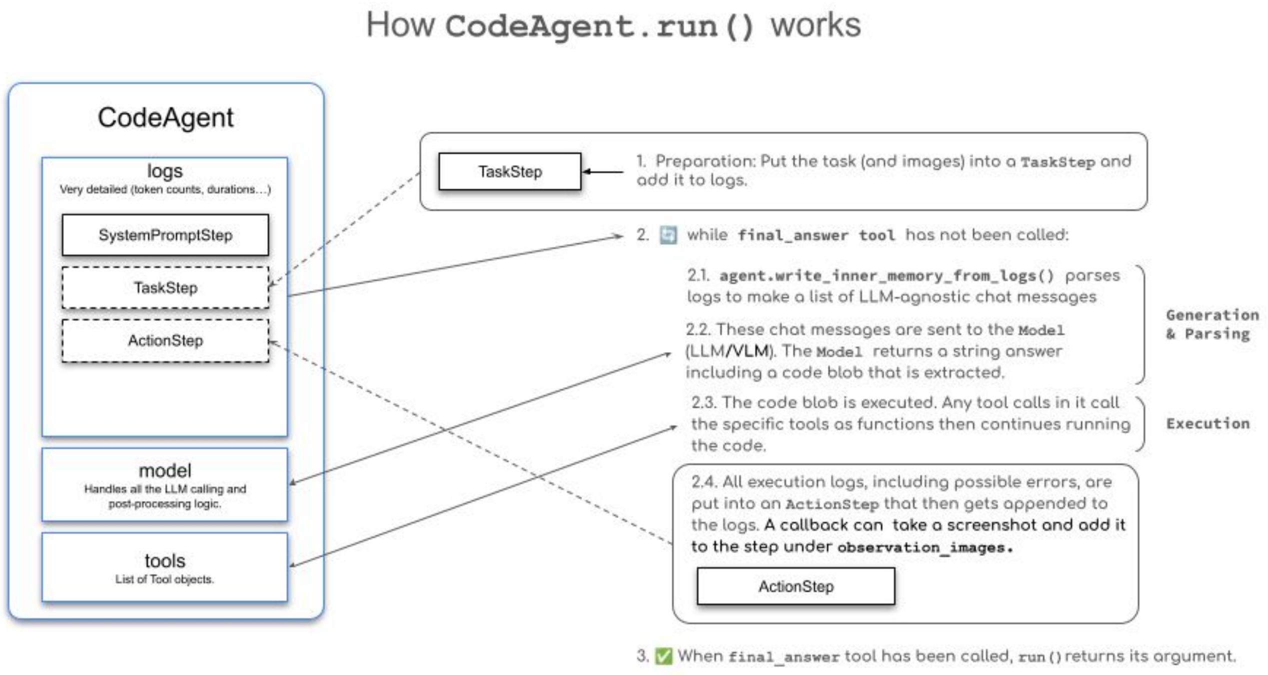
Why smolagents

- In smolagents, agents operate as multi-step agents:
  - One thought
  - One tool call and execution
- Type of agents:
  - CodeAgent! 🧑‍💻
  - ToolCallingAgent 🔧

Building Agents that use code

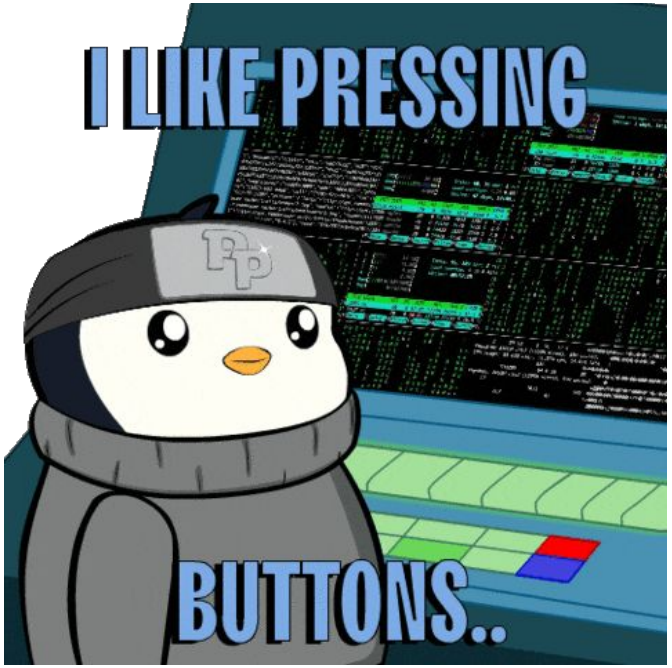
- **Why Code Agents?**
  - **Composability:** easily combine and reuse actions
  - **Object Management:** work directly with complex structures like images
  - **Generality:** express any computationally possible task
  - **Natural for LLMs:** High-quality code is already present in LLM training data.

Building Agents that use code



How CodeAgent.run() works
CodeAgent
logs
Very detailed (token counts, durations…)
SystemPromptStep
TaskStep
ActionStep
model
Handles all the LLM calling and post-processing logic.
tools
List of Tool objects.
TaskStep
1. Preparation: Put the task (and images) into a TaskStep and add it to logs.
2. 🔄 while final_answer tool has not been called:
2.1. agent.write_inner_memory_from_logs() parses logs to make a list of LLM-agnostic chat messages
2.2. These chat messages are sent to the Model (LLM/VLM). The Model returns a string answer including a code blob that is extracted.
Generation & Parsing
2.3. The code blob is executed. Any tool calls in it call the specific tools as functions then continues running the code.
Execution
2.4. All execution logs, including possible errors, are put into an ActionStep that then gets appended to the logs. A callback can take a screenshot and add it to the step under observation_images.
ActionStep
3. ✅ When final_answer tool has been called, run() returns its argument.

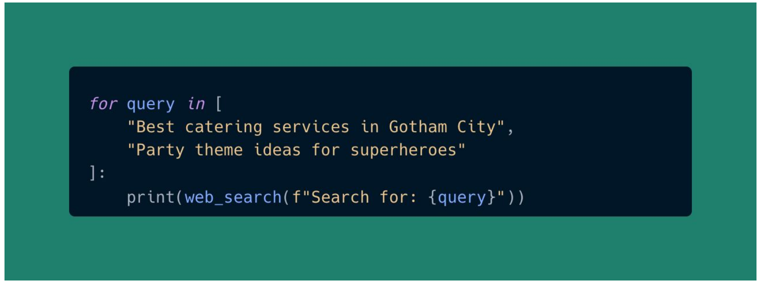
Building Agents that use code



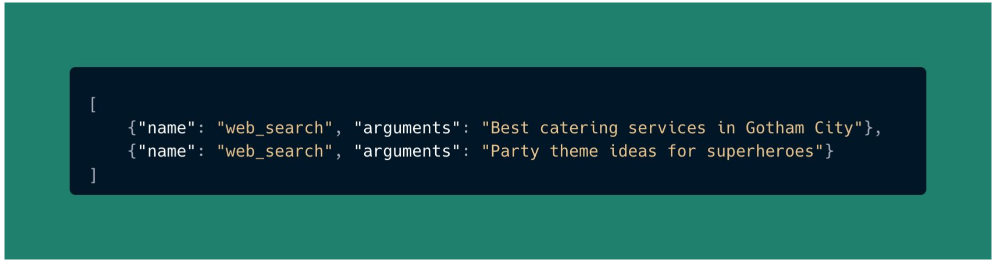
Notebook

Writing actions as code snippets or JSON blobs

- Use built-in tool-calling capabilities of LLM providers to generate tool calls as JSON structures



```python
for query in [
    "Best catering services in Gotham City",
    "Party theme ideas for superheroes"
]:
    print(web_search(f"Search for: {query}"))
```



```json
[
    {"name": "web_search", "arguments": "Best catering services in Gotham City"},
    {"name": "web_search", "arguments": "Party theme ideas for superheroes"}
]
```

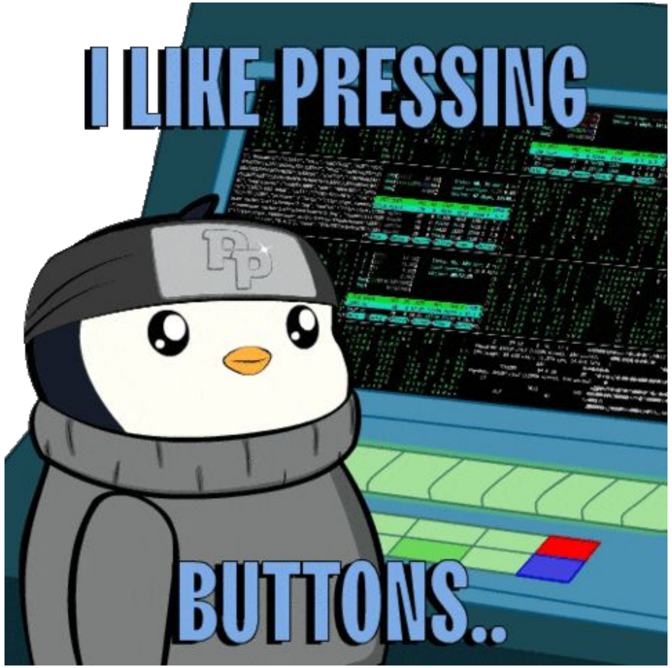
Tools

- The LLM need an interface description with:
    - **Name:** web_search
    - **Tool description:** searches the web for specific queries
    - **Input types and descriptions:** query (string)
    - **Output types:** string containing the search results.
- How are they created:
    - @tool decorator
    - Subclass of Tool

Tools

- smolagents also has a default toolbox:
  - PythonInterpreterTool
  - FinalAnswerTool
  - UserInputTool
  - DuckDuckGoSearchTool
  - GoogleSearchTool
  - VisitWebpageTool
- Tools can be shared and imported:
  - To/from HF Hub
  - From HF Spaces
  - From LangChain Tools
  - From MCP Servers

Tools

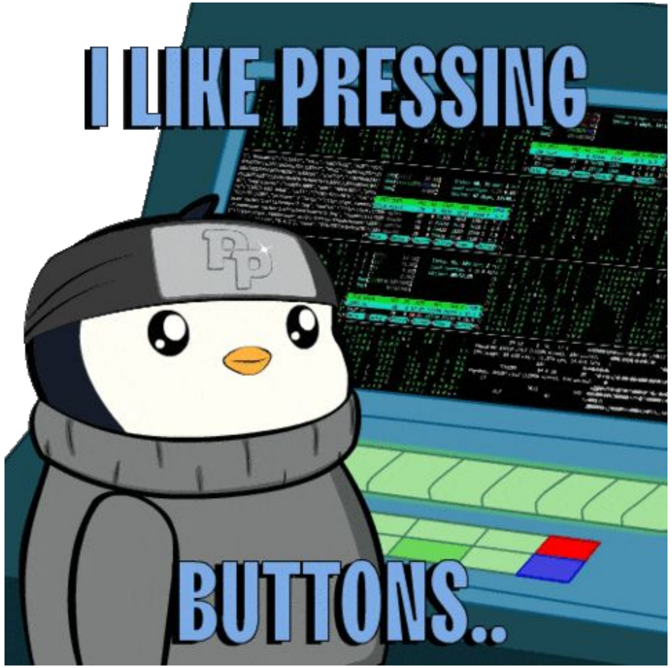


Notebook

Retrieval Agents

- Retrieval Augmented Generation (RAG) systems combine the capabilities of data retrieval and generation models to provide context-aware responses.
- **Agentic RAG** → combines autonomous agents with dynamic knowledge retrieval.
- We could retrieve from:
  - The web
  - Custom knowledge bases using a specific tool
  - We could enhance these systems with more capabilities.

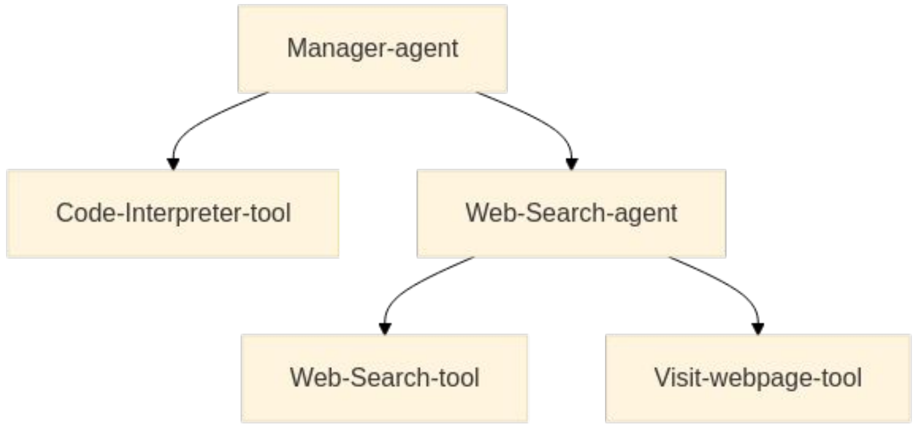
Retrieval Agents



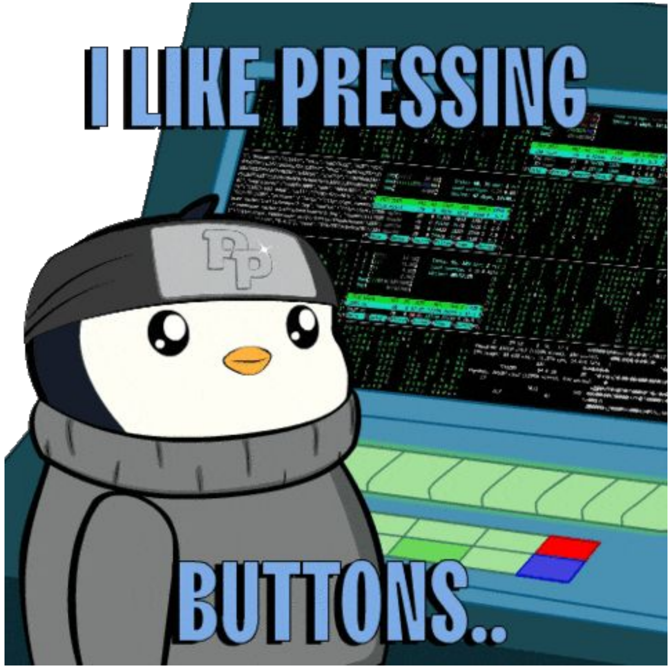
Notebook

Multi-Agent Systems

- Enable specialized agents to collaborate on complex tasks.



Multi-Agent Systems



Notebook

Vision and Browser Agents

- Let’s imagine all the previous but with vision! 👀
- We could:
  - Provide images at the start of the agent’s execution
  - Provide images with dynamic retrieval!

Vision and Browser Agents

Wikipedia

Wonder Woman

From Wikipedia, the free encyclopedia

[illegible]

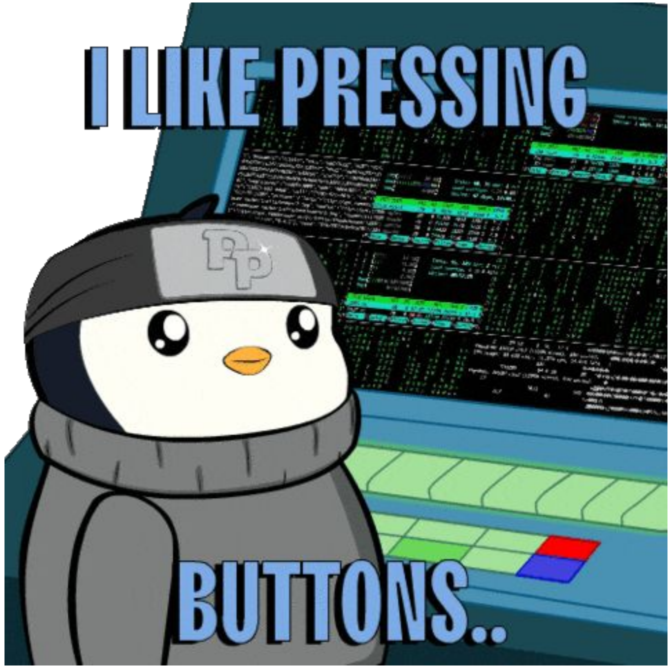
Princess Diana of Themyscira
Wonder Woman

[illegible]

Publication information

```
[illegible]
```

Vision and Browser Agents

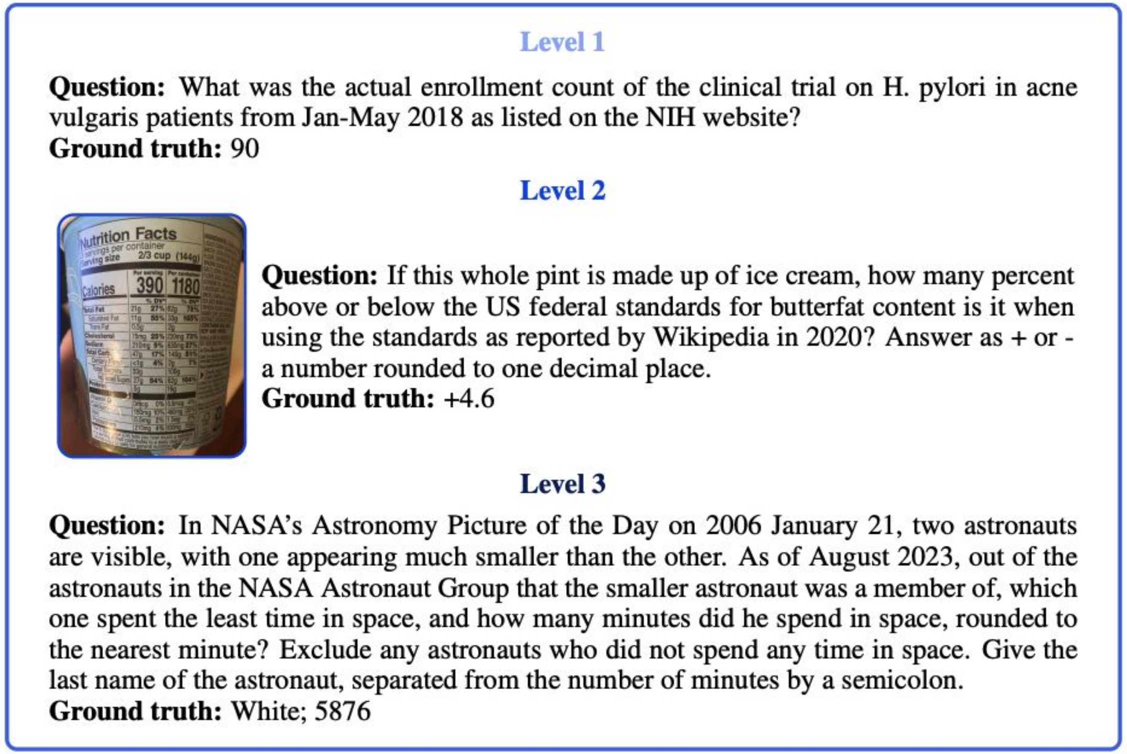


Notebook and code

Benchmarking!

- GAIA benchmark: designed to evaluate AI assistant on real-world tasks.
  - Humans: ~92% success rate
  - GPT-4 with plugins: ~15%
  - Deep Research (OpenAI): 67.36% on the validation set
- There are also tools for observability and evaluation:
  - Langfuse
  - Weave

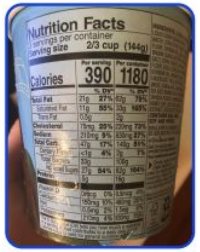
Benchmarking!



Level 1

Question: What was the actual enrollment count of the clinical trial on H. pylori in acne vulgaris patients from Jan-May 2018 as listed on the NIH website?
Ground truth: 90

Level 2

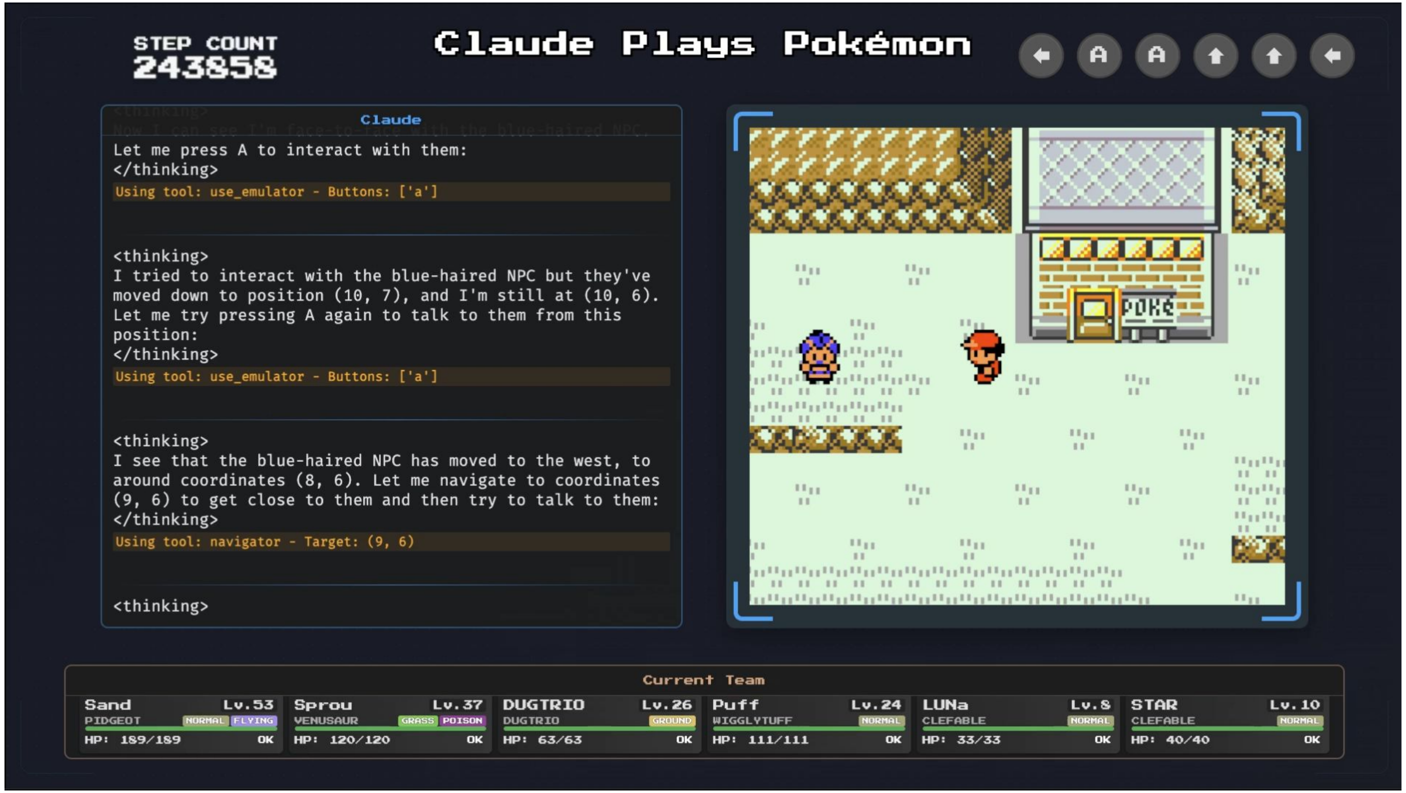


Question: If this whole pint is made up of ice cream, how many percent above or below the US federal standards for butterfat content is it when using the standards as reported by Wikipedia in 2020? Answer as + or - a number rounded to one decimal place.
Ground truth: +4.6

Level 3

Question: In NASA's Astronomy Picture of the Day on 2006 January 21, two astronauts are visible, with one appearing much smaller than the other. As of August 2023, out of the astronauts in the NASA Astronaut Group that the smaller astronaut was a member of, which one spent the least time in space, and how many minutes did he spend in space, rounded to the nearest minute? Exclude any astronauts who did not spend any time in space. Give the last name of the astronaut, separated from the number of minutes by a semicolon.
Ground truth: White; 5876

Interesting use cases



```
<thinking>
Now I can see I'm face-to-face with the blue-haired NPC.
Let me press A to interact with them:
</thinking>
Using tool: use_emulator - Buttons: ['a']

<thinking>
I tried to interact with the blue-haired NPC but they've
moved down to position (10, 7), and I'm still at (10, 6).
Let me try pressing A again to talk to them from this
position:
</thinking>
Using tool: use_emulator - Buttons: ['a']

<thinking>
I see that the blue-haired NPC has moved to the west, to
around coordinates (8, 6). Let me navigate to coordinates
(9, 6) to get close to them and then try to talk to them:
</thinking>
Using tool: navigator - Target: (9, 6)

<thinking>
```

Current Team

Sand Lv.53	Sprou Lv.37	DUGTRIO Lv.26	Puff Lv.24	LUNa Lv.8	STAR Lv.10
PIDGEOT NORMAL FLYING	VENUSAUR GRASS POISON	DUGTRIO GROUND	WIGGLYTUFF NORMAL	CLEFABLE NORMAL	CLEFABLE NORMAL
HP: 189/189 OK	HP: 120/120 OK	HP: 63/63 OK	HP: 111/111 OK	HP: 33/33 OK	HP: 40/40 OK

If you want to know more!

[Agents Course]

Thanks! 🎉