LLM Agents

CS 4804: Introduction to AI

Fall 2025

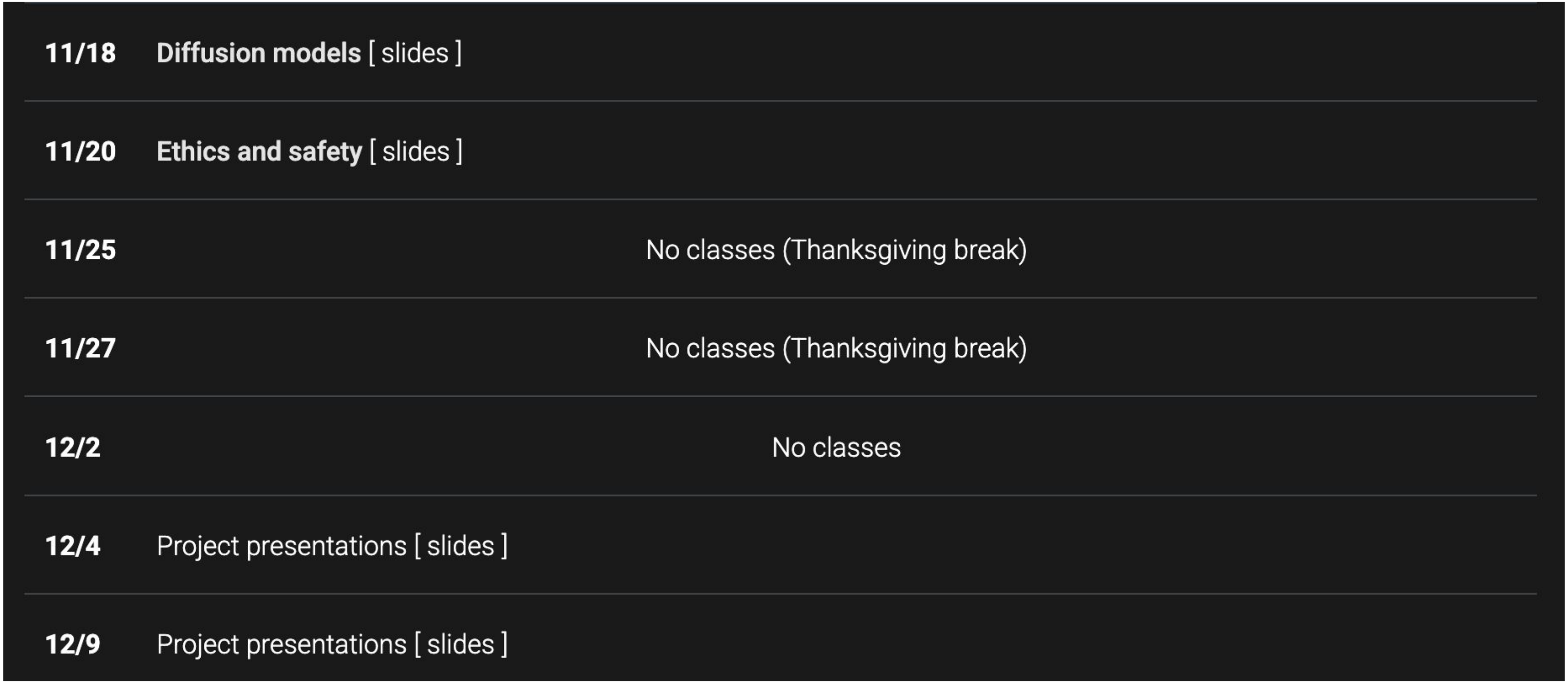
https://tuvllms.github.io/ai-fall-2025/

Tu Vu

VIRGINIA TECH.

Logistics

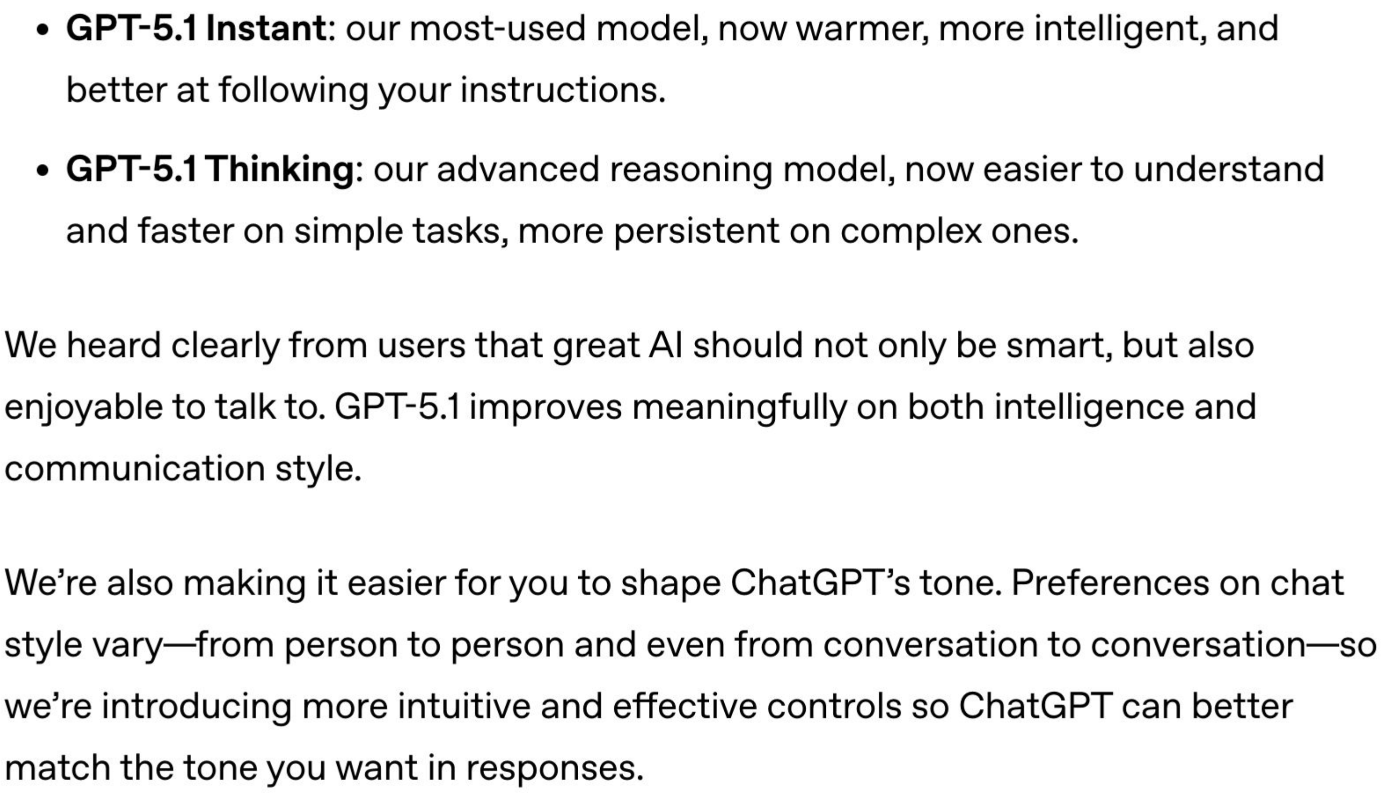
- HW 2 due 11/18
- Final presentations: 12/4 & 12/9
    - Sign-up form available on Piazza later today



11/18	**Diffusion models** [slides]
11/20	**Ethics and safety** [slides]
11/25	No classes (Thanksgiving break)
11/27	No classes (Thanksgiving break)
12/2	No classes
12/4	Project presentations [slides]
12/9	Project presentations [slides]

GPT-5.1

https://openai.com/index/gpt-5-1/

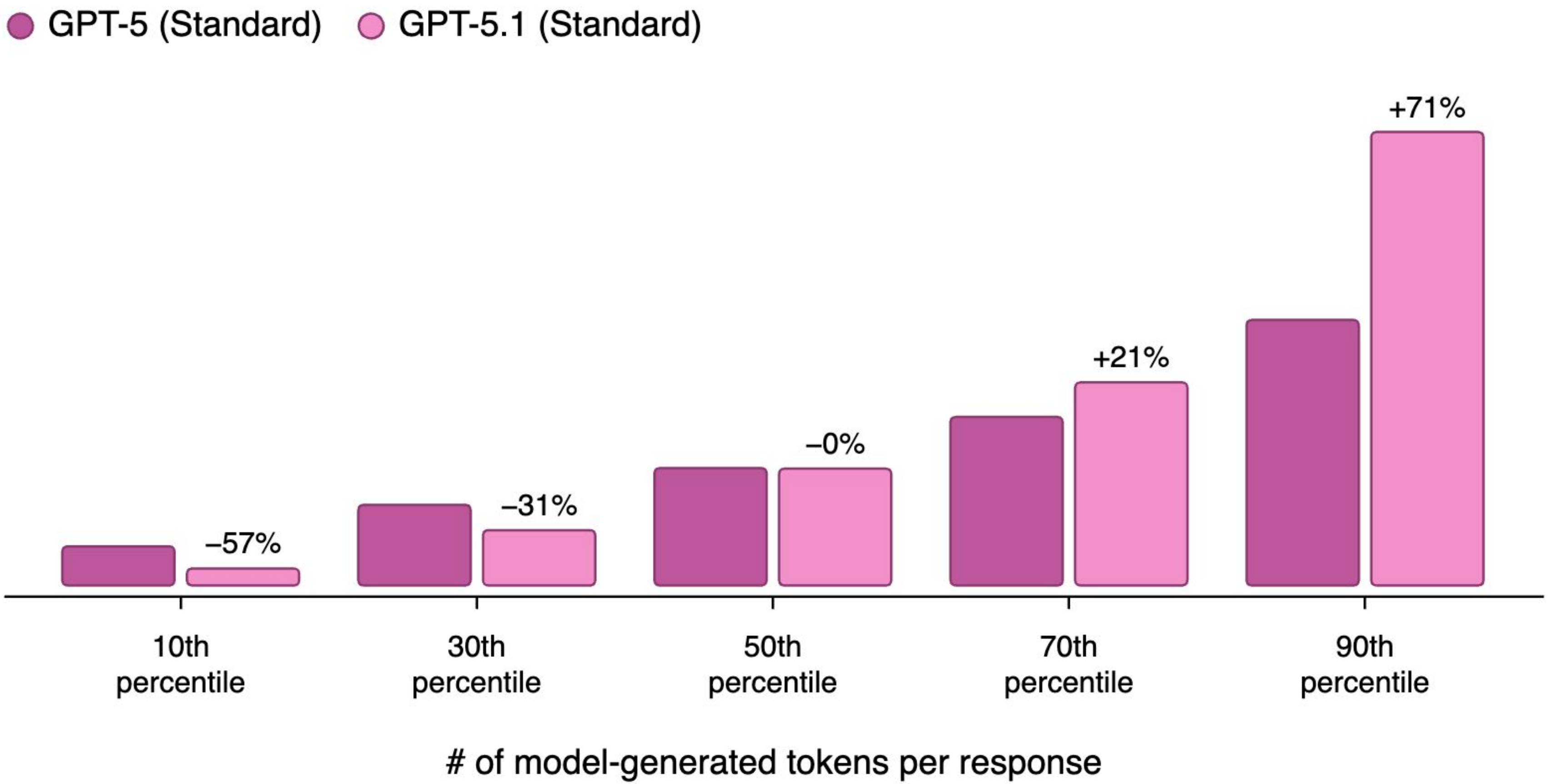


- **GPT-5.1 Instant**: our most-used model, now warmer, more intelligent, and better at following your instructions.
- **GPT-5.1 Thinking**: our advanced reasoning model, now easier to understand and faster on simple tasks, more persistent on complex ones.

We heard clearly from users that great AI should not only be smart, but also enjoyable to talk to. GPT-5.1 improves meaningfully on both intelligence and communication style.

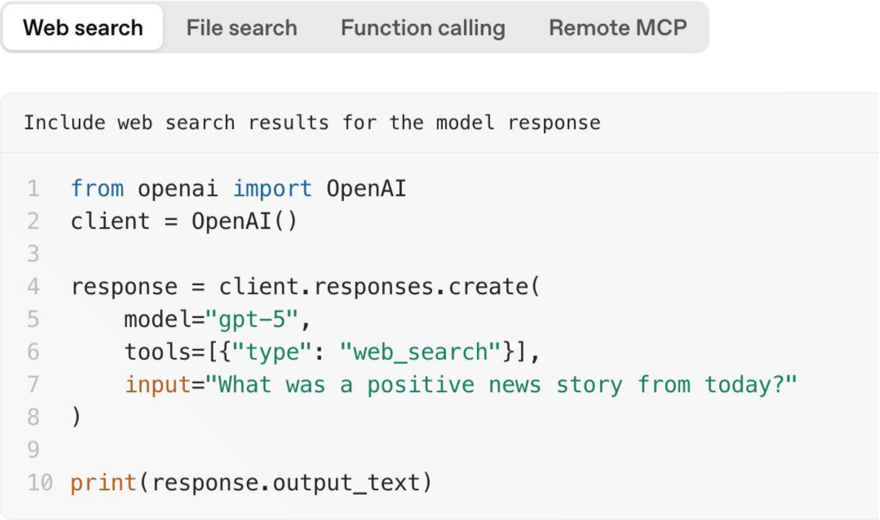
We're also making it easier for you to shape ChatGPT's tone. Preferences on chat style vary—from person to person and even from conversation to conversation—so we're introducing more intuitive and effective controls so ChatGPT can better match the tone you want in responses.

GPT-5.1 spends less time on easy tasks, and more time on hard tasks



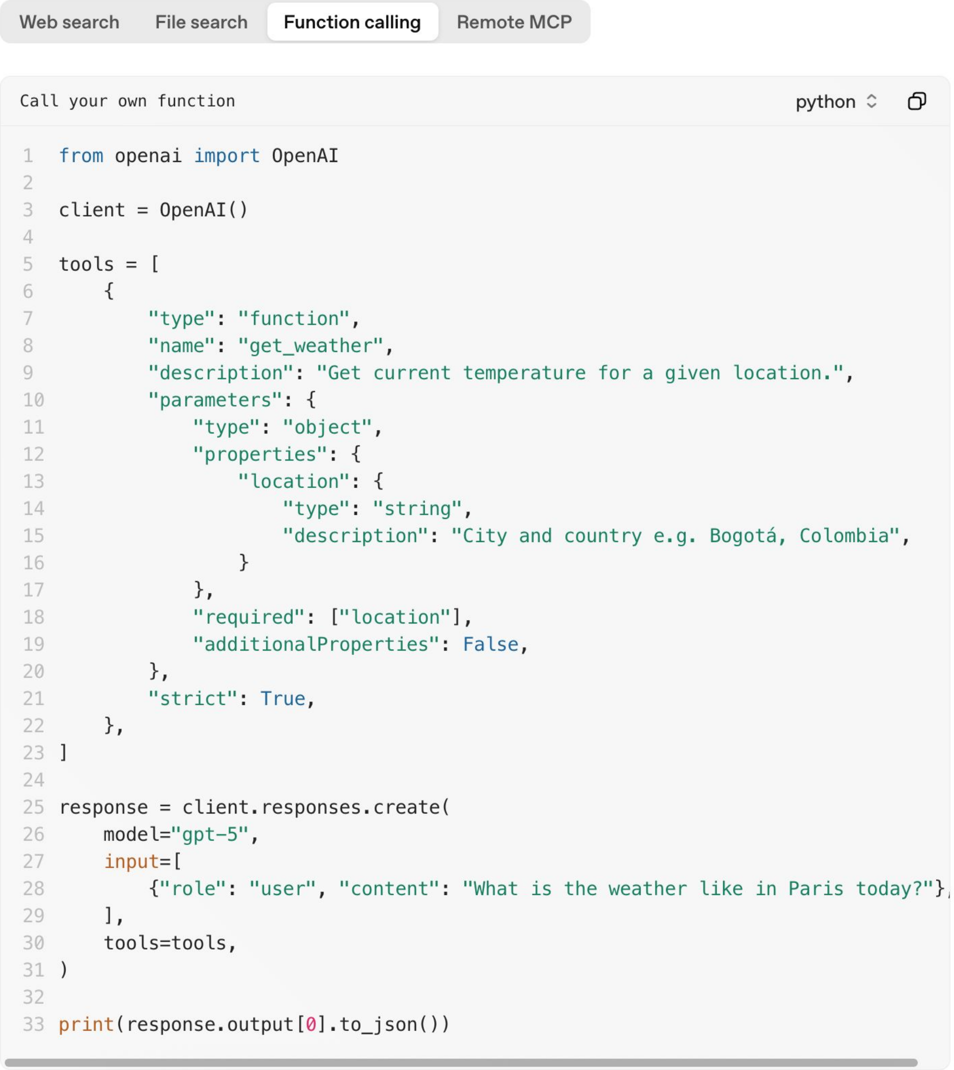
Using tools

https://platform.openai.com/docs/guides/tools



Include web search results for the model response

```python
from openai import OpenAI
client = OpenAI()

response = client.responses.create(
    model="gpt-5",
    tools=[{"type": "web_search"}],
    input="What was a positive news story from today?"
)

print(response.output_text)
```



Call your own function

```python
from openai import OpenAI

client = OpenAI()

tools = [
    {
        "type": "function",
        "name": "get_weather",
        "description": "Get current temperature for a given location.",
        "parameters": {
            "type": "object",
            "properties": {
                "location": {
                    "type": "string",
                    "description": "City and country e.g. Bogotá, Colombia",
                }
            },
            "required": ["location"],
            "additionalProperties": False,
        },
        "strict": True,
    },
]

response = client.responses.create(
    model="gpt-5",
    input=[
        {"role": "user", "content": "What is the weather like in Paris today?"},
    ],
    tools=tools,
)

print(response.output[0].to_json())
```

What is the most important AI technology to pay attention to?

Andrew Ng

I would say Agentic AI

Heroes of AI / Deep Learning

Geoffrey Hinton

Backpropagation & deep learning

Yann LeCun

Convolutional neural networks

Yoshua Bengio

Representation learning

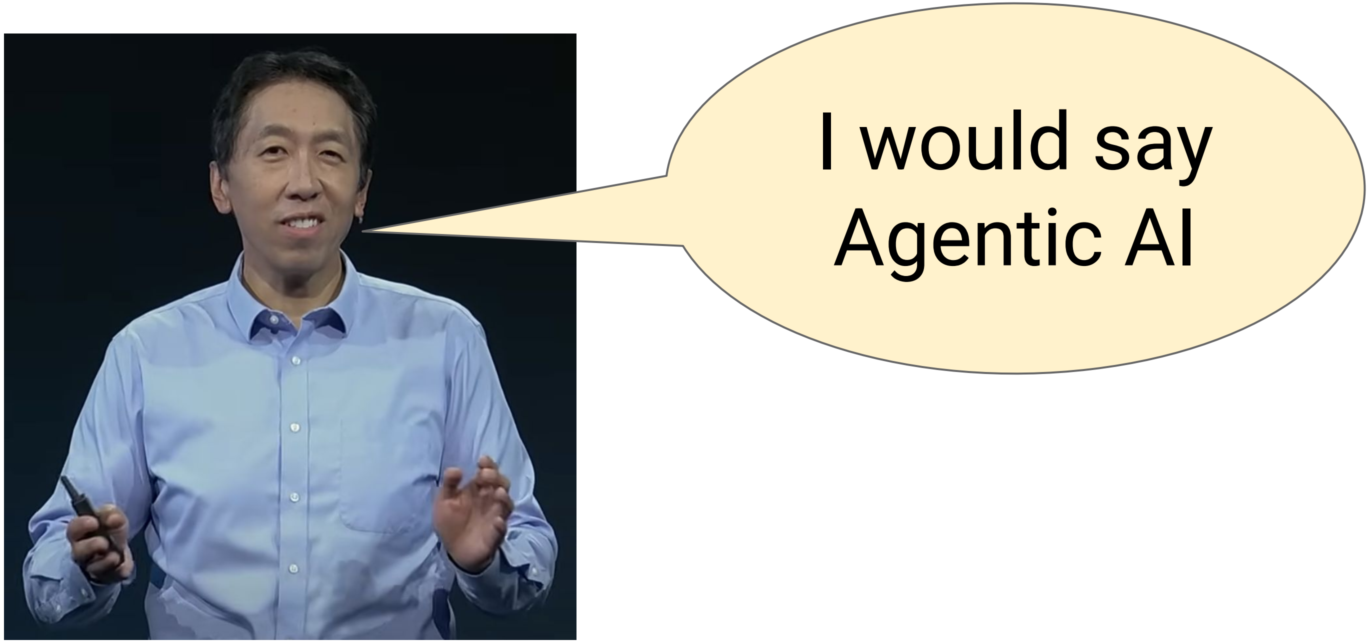
Juergen Schmidhuber

Long short-term memory (LSTM)

Andrew Ng

AI education & large-scale ML systems

What is the most important AI technology to pay attention to?



I would say Agentic AI

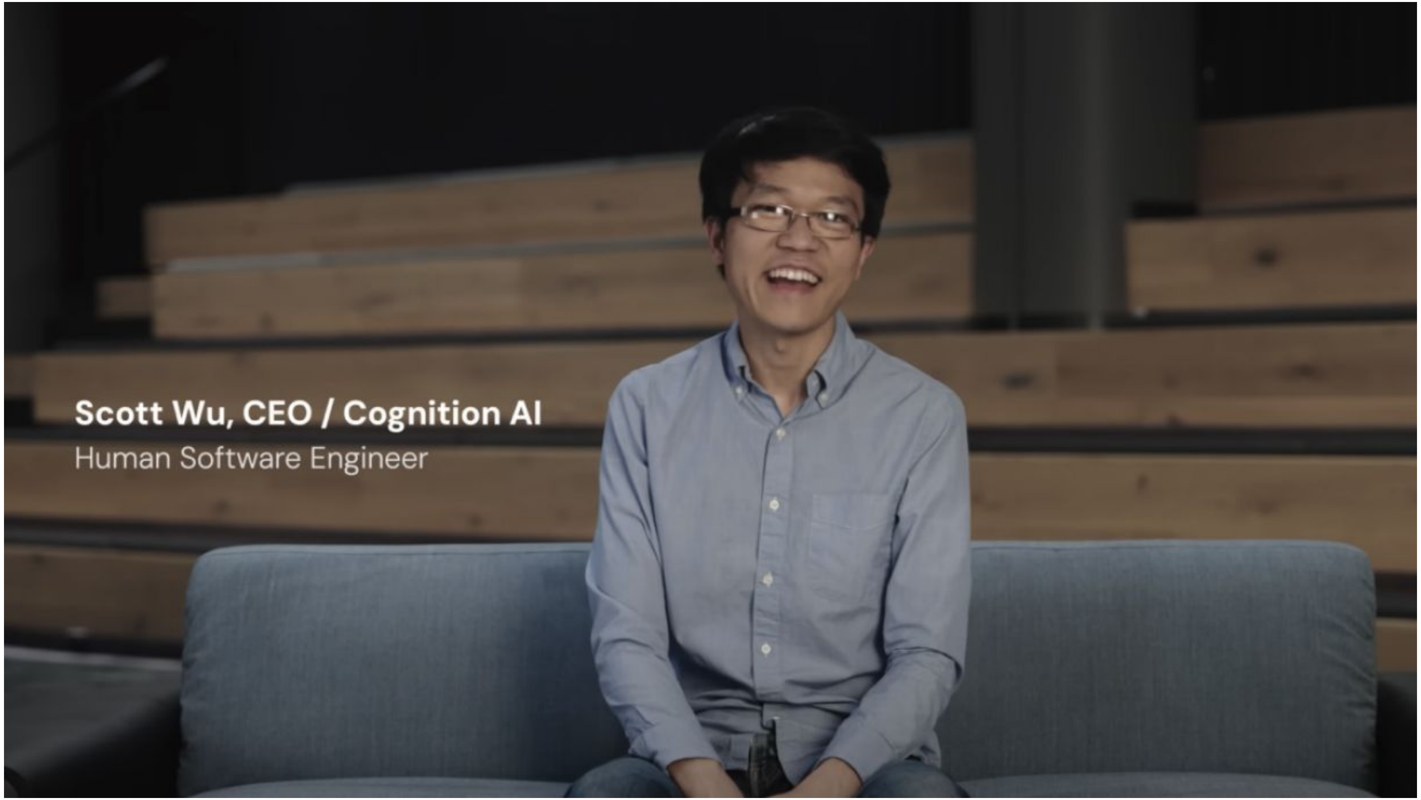
Andrew Ng

What is the most important AI technology to pay attention to?

Andrew Ng

I think AI agent workflows will drive massive AI progress this year — perhaps even more than the next generation of foundation models. This is an important trend, and I urge everyone who works in AI to pay attention to it.

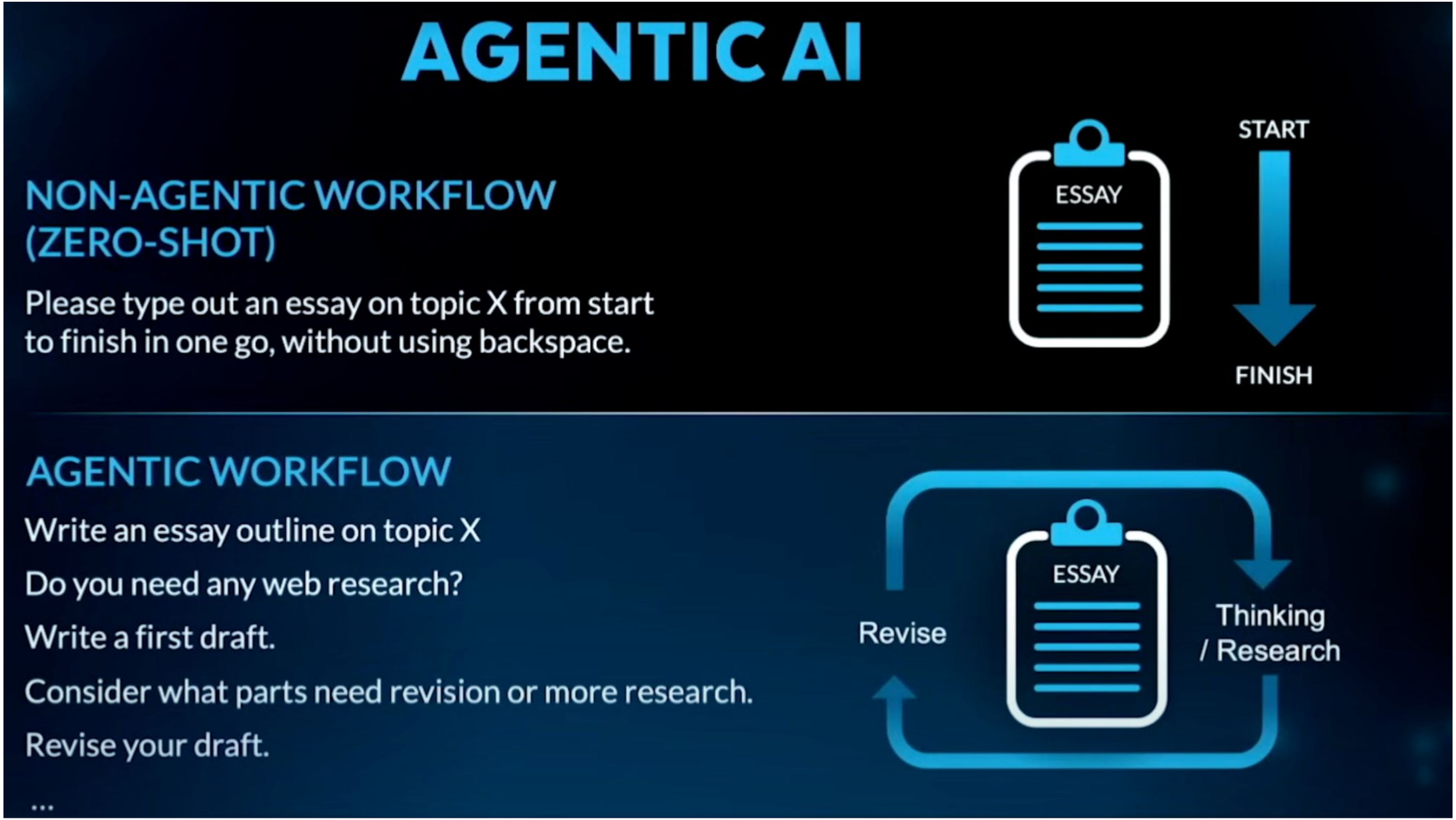
Devin: the first AI software engineer



https://www.youtube.com/watch?v=fjHtjT7GO1c

Manus: the general AI agent

https://www.youtube.com/watch?v=K27diMbCsuw



AGENTIC AI

NON-AGENTIC WORKFLOW (ZERO-SHOT)

Please type out an essay on topic X from start to finish in one go, without using backspace.

AGENTIC WORKFLOW

Write an essay outline on topic X

Do you need any web research?

Write a first draft.

Consider what parts need revision or more research.

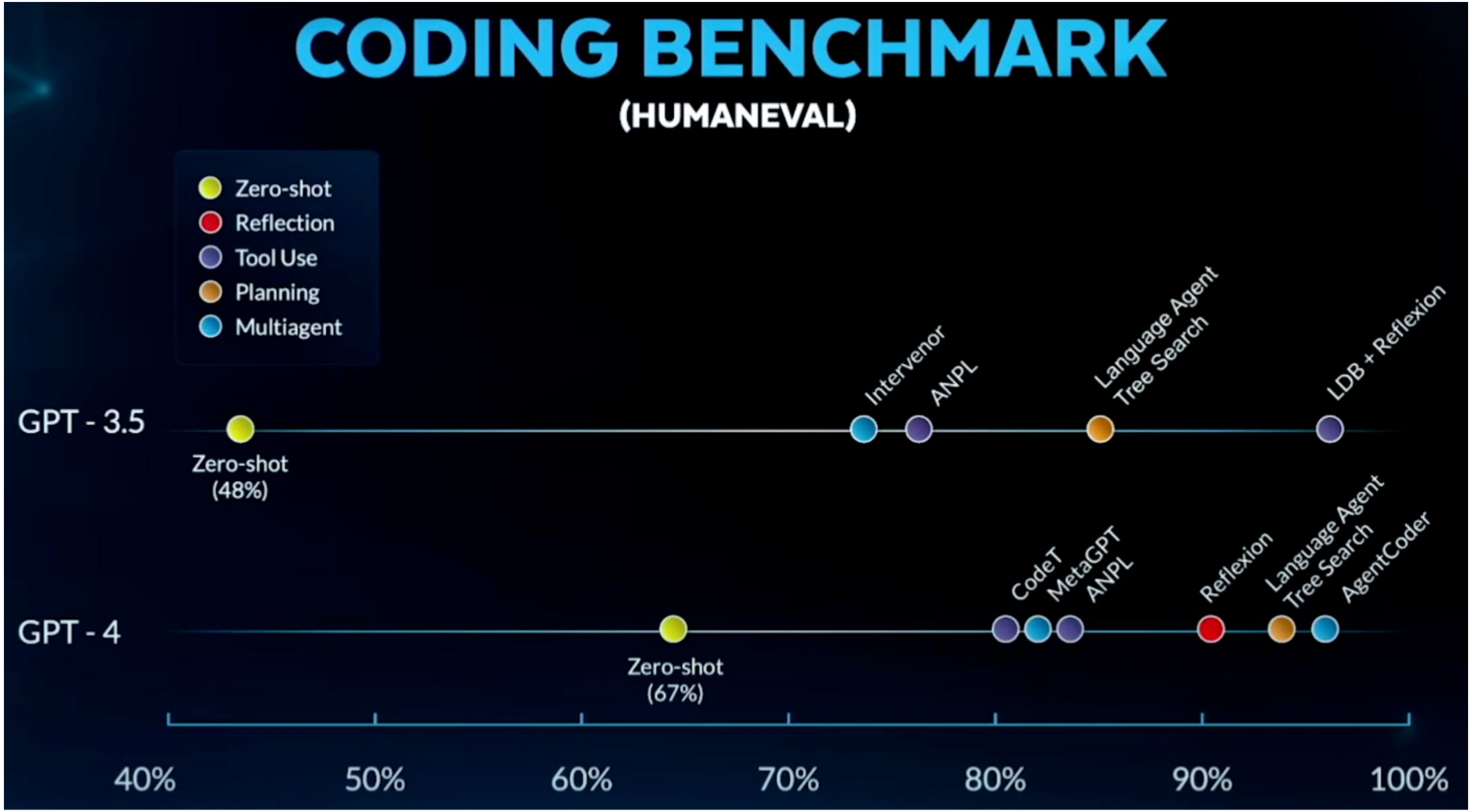
Revise your draft.

…

ESSAY
START
FINISH
ESSAY
Revise
Thinking / Research

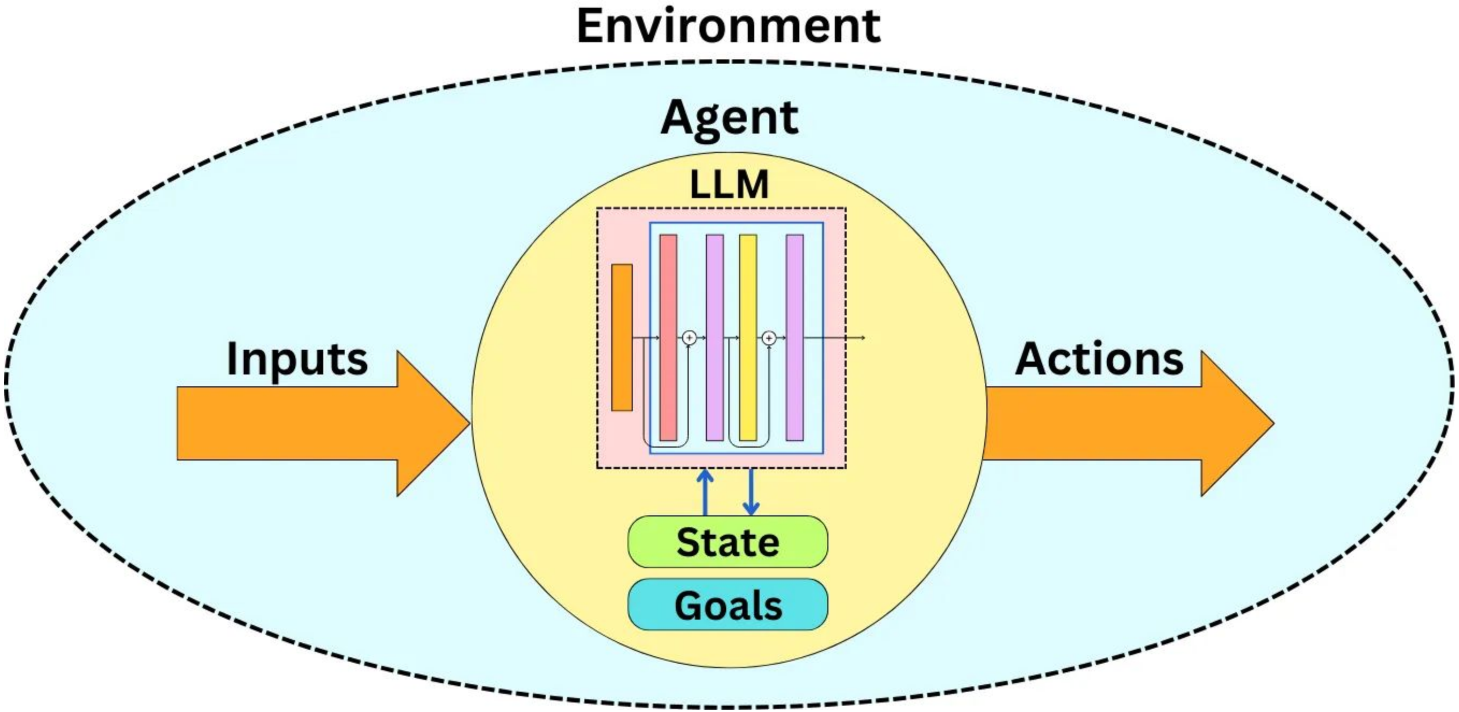
"Explores The Rise Of AI Agents And Agentic Reasoning" by Andrew Ng

Performance using zero-shot and agent workflows

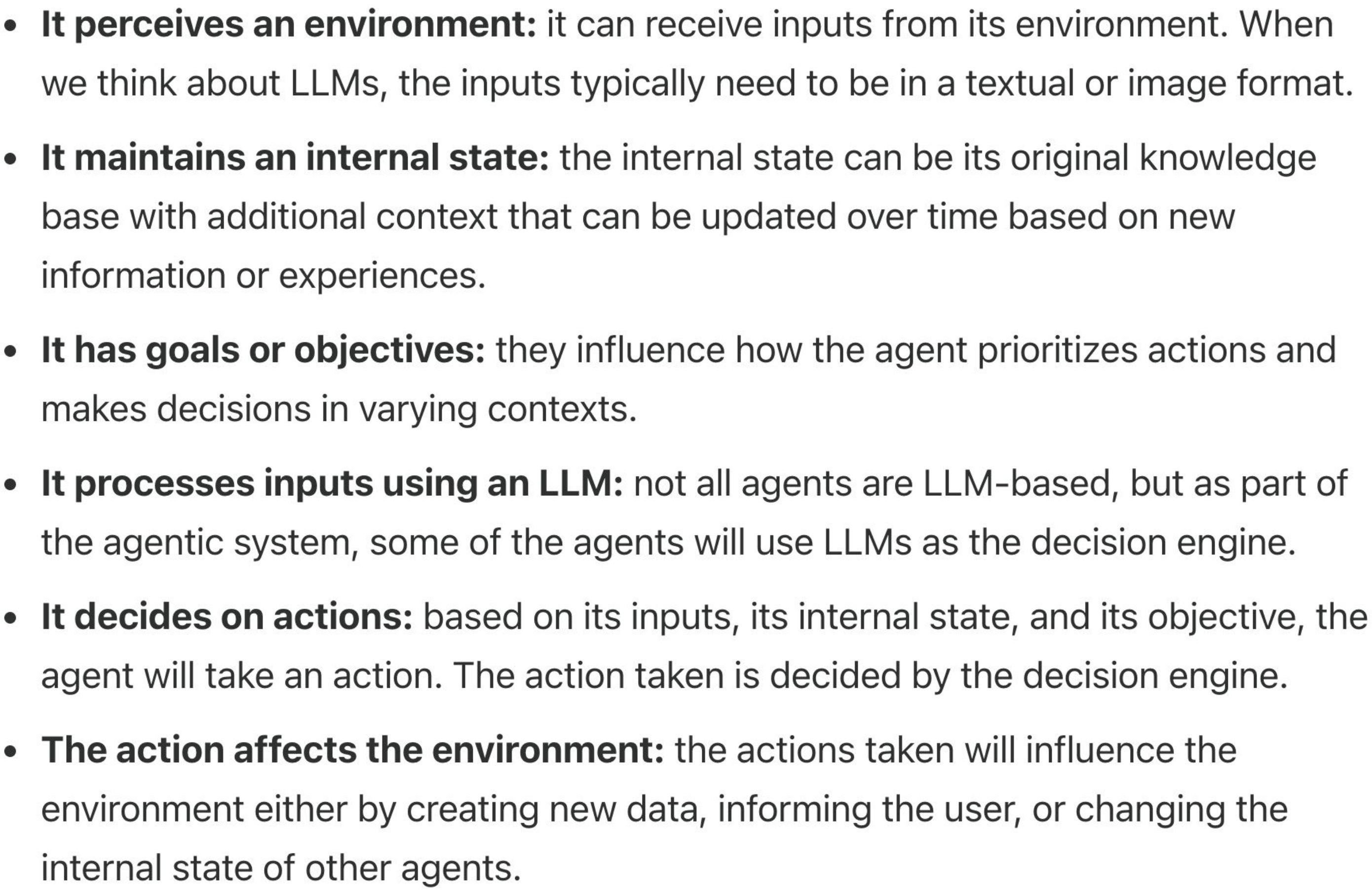


["Explores The Rise Of AI Agents And Agentic Reasoning" by Andrew Ng](#)

What is an agent?



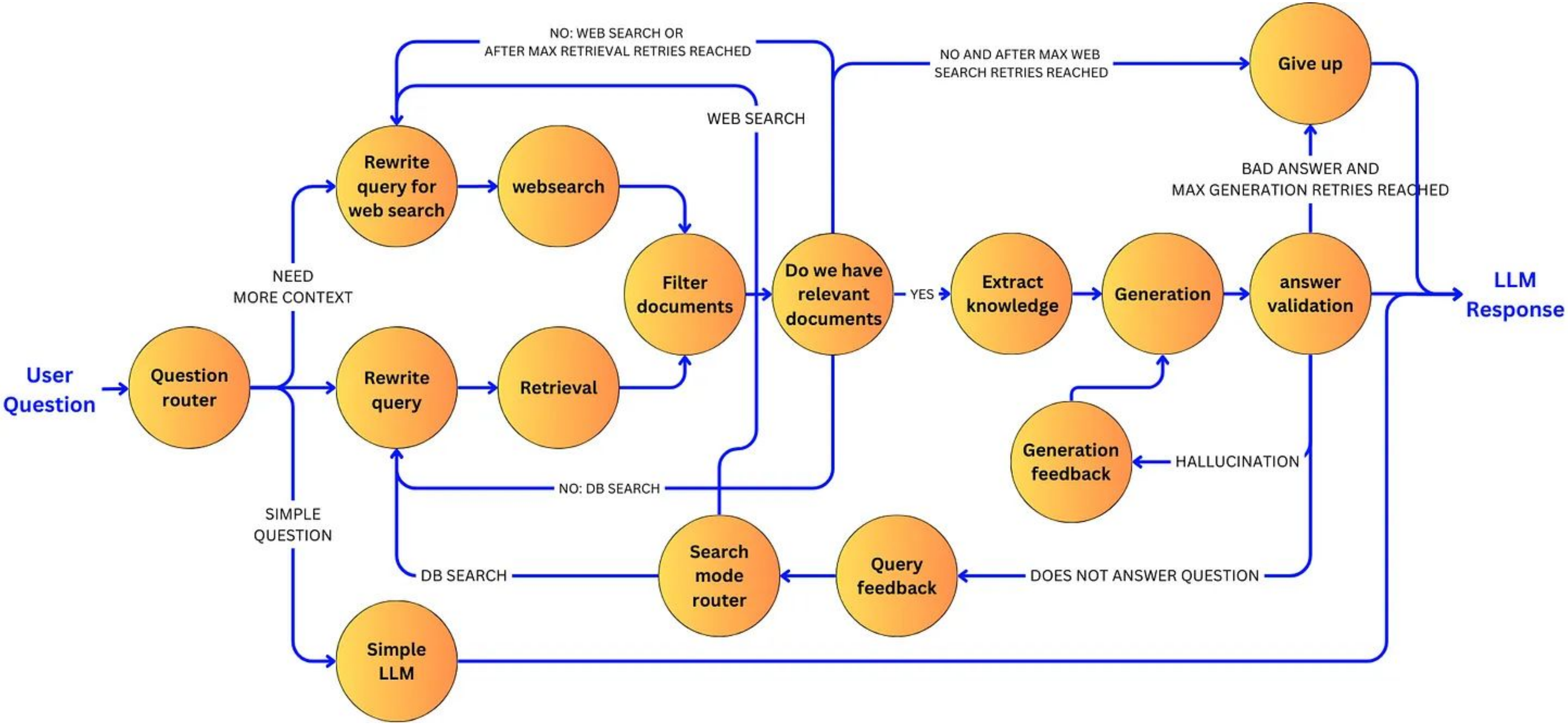
"The Different Agentic Patterns" by Damien Benveniste



- **It perceives an environment:** it can receive inputs from its environment. When we think about LLMs, the inputs typically need to be in a textual or image format.
- **It maintains an internal state:** the internal state can be its original knowledge base with additional context that can be updated over time based on new information or experiences.
- **It has goals or objectives:** they influence how the agent prioritizes actions and makes decisions in varying contexts.
- **It processes inputs using an LLM:** not all agents are LLM-based, but as part of the agentic system, some of the agents will use LLMs as the decision engine.
- **It decides on actions:** based on its inputs, its internal state, and its objective, the agent will take an action. The action taken is decided by the decision engine.
- **The action affects the environment:** the actions taken will influence the environment either by creating new data, informing the user, or changing the internal state of other agents.

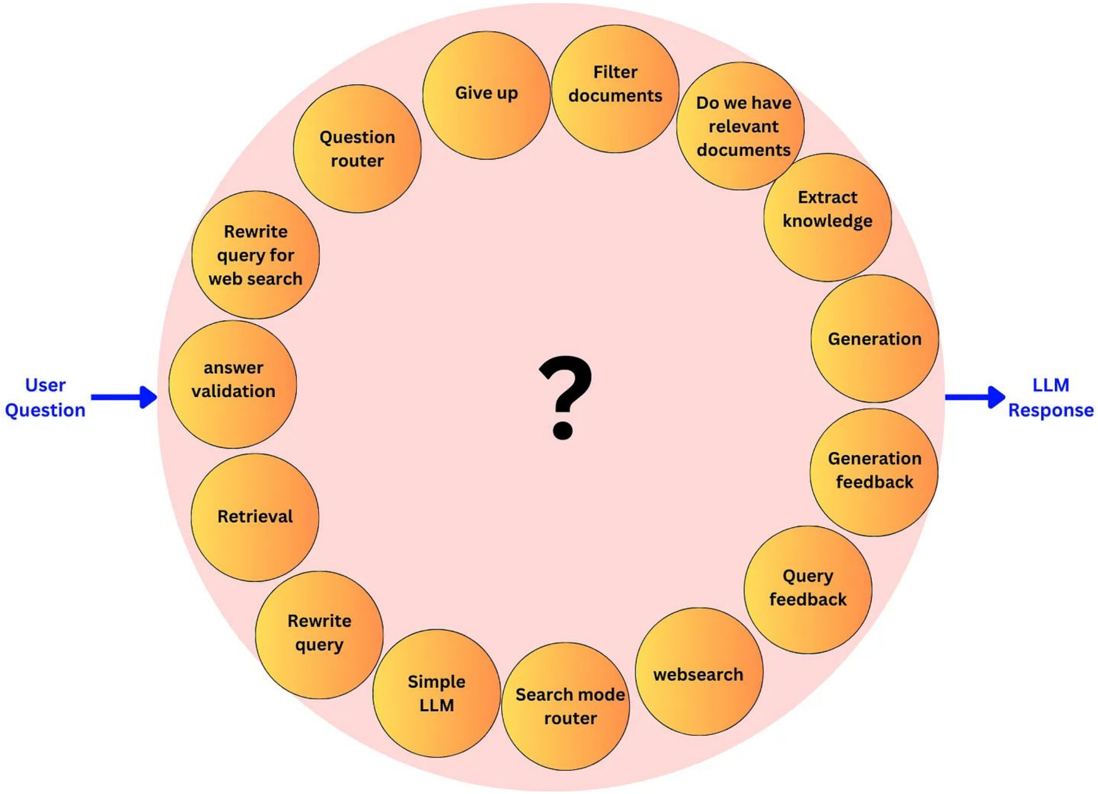
"The Different Agentic Patterns" by Damien Benveniste

Why not a pipeline where we encode all the possible states and actions?



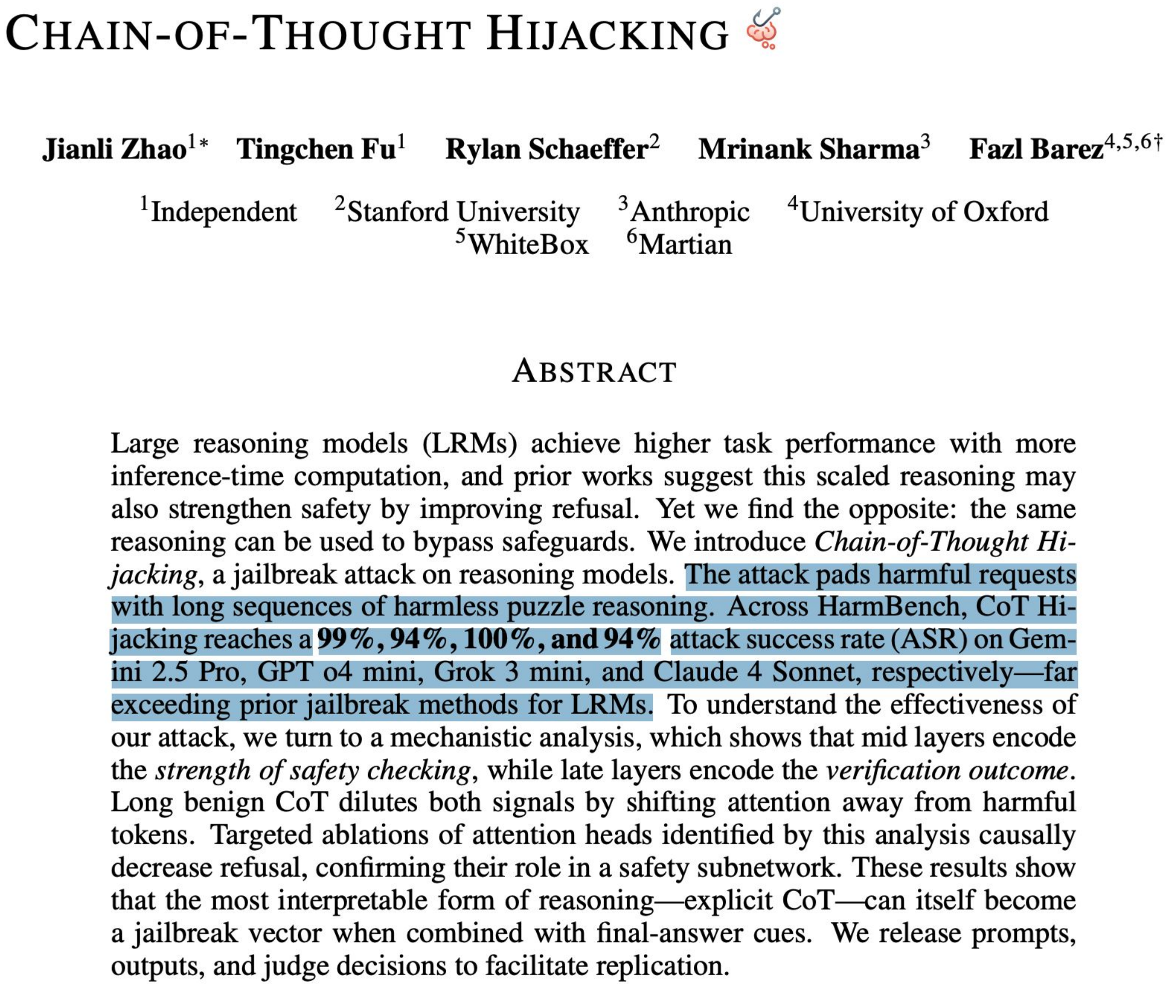
["The Different Agentic Patterns" by Damien Benveniste](#)

Let the agents choose their own paths



"The Different Agentic Patterns" by Damien Benveniste

Are there any risks associated with leaving the choice of actions to an LLM?



CHAIN-OF-THOUGHT HIJACKING

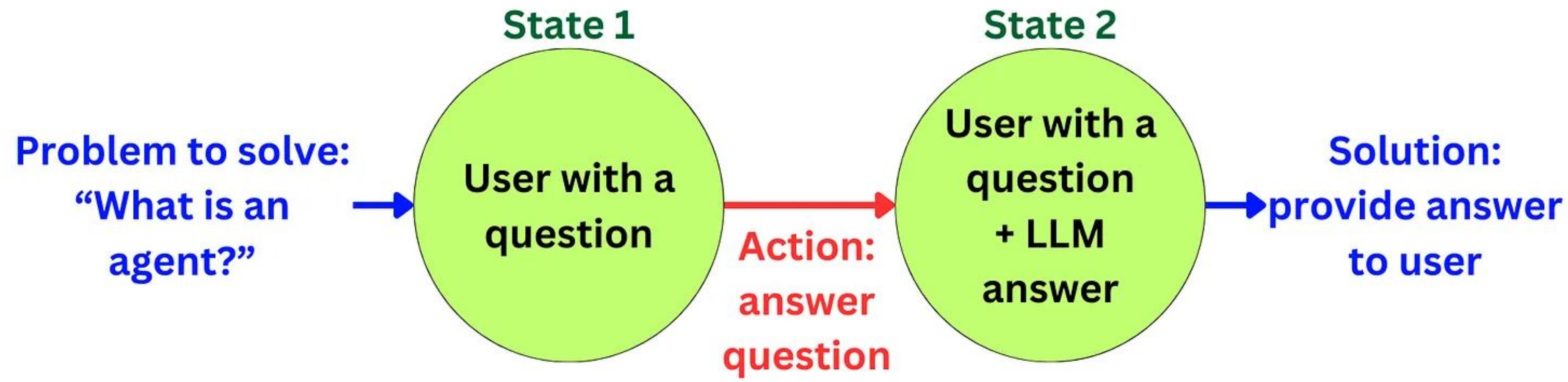
Jianli Zhao[1]* **Tingchen Fu**[1] **Rylan Schaeffer**[2] **Mrinank Sharma**[3] **Fazl Barez**[4,5,6]†

[1]Independent [2]Stanford University [3]Anthropic [4]University of Oxford
[5]WhiteBox [6]Martian

ABSTRACT

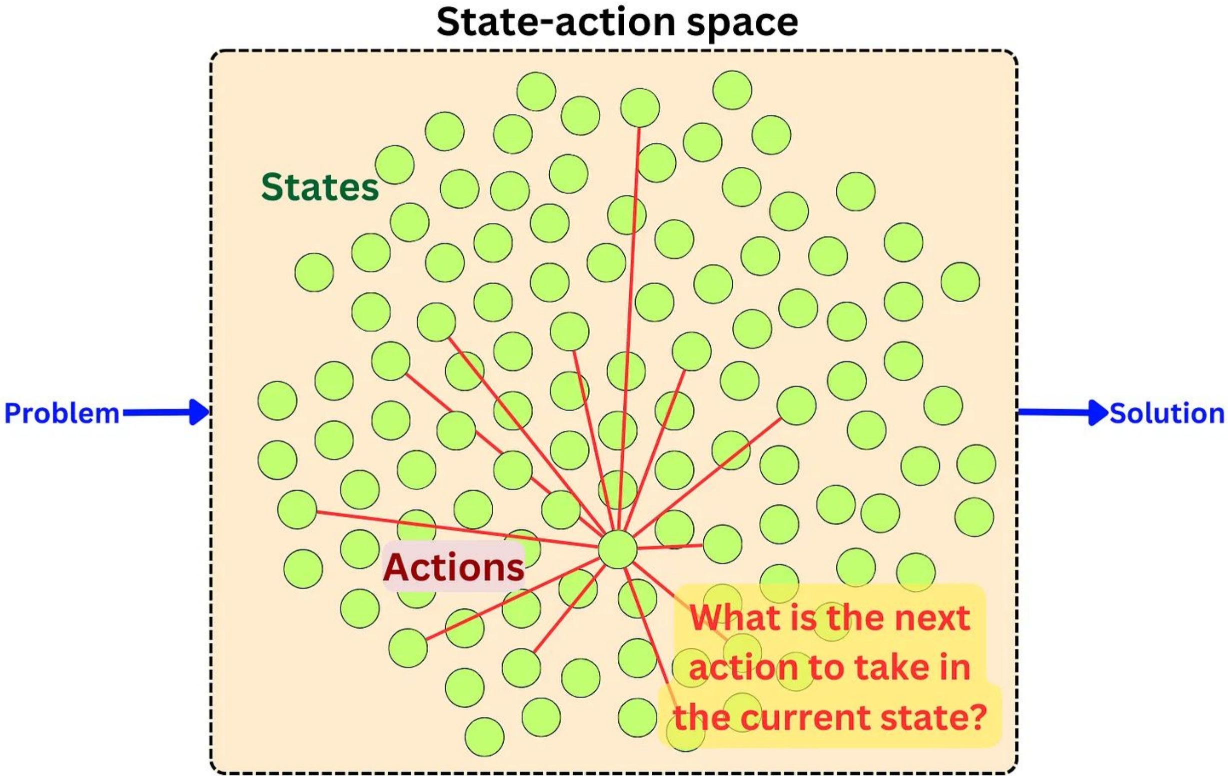
Large reasoning models (LRMs) achieve higher task performance with more inference-time computation, and prior works suggest this scaled reasoning may also strengthen safety by improving refusal. Yet we find the opposite: the same reasoning can be used to bypass safeguards. We introduce *Chain-of-Thought Hijacking*, a jailbreak attack on reasoning models. The attack pads harmful requests with long sequences of harmless puzzle reasoning. Across HarmBench, CoT Hijacking reaches a **99%, 94%, 100%, and 94%** attack success rate (ASR) on Gemini 2.5 Pro, GPT o4 mini, Grok 3 mini, and Claude 4 Sonnet, respectively—far exceeding prior jailbreak methods for LRMs. To understand the effectiveness of our attack, we turn to a mechanistic analysis, which shows that mid layers encode the *strength of safety checking*, while late layers encode the *verification outcome*. Long benign CoT dilutes both signals by shifting attention away from harmful tokens. Targeted ablations of attention heads identified by this analysis causally decrease refusal, confirming their role in a safety subnetwork. These results show that the most interpretable form of reasoning—explicit CoT—can itself become a jailbreak vector when combined with final-answer cues. We release prompts, outputs, and judge decisions to facilitate replication.

When we don’t need to leave the choice of action to a decision engine



"The Different Agentic Patterns" by Damien Benveniste

It might become a better option when the amount of possible state-action pairs is too large



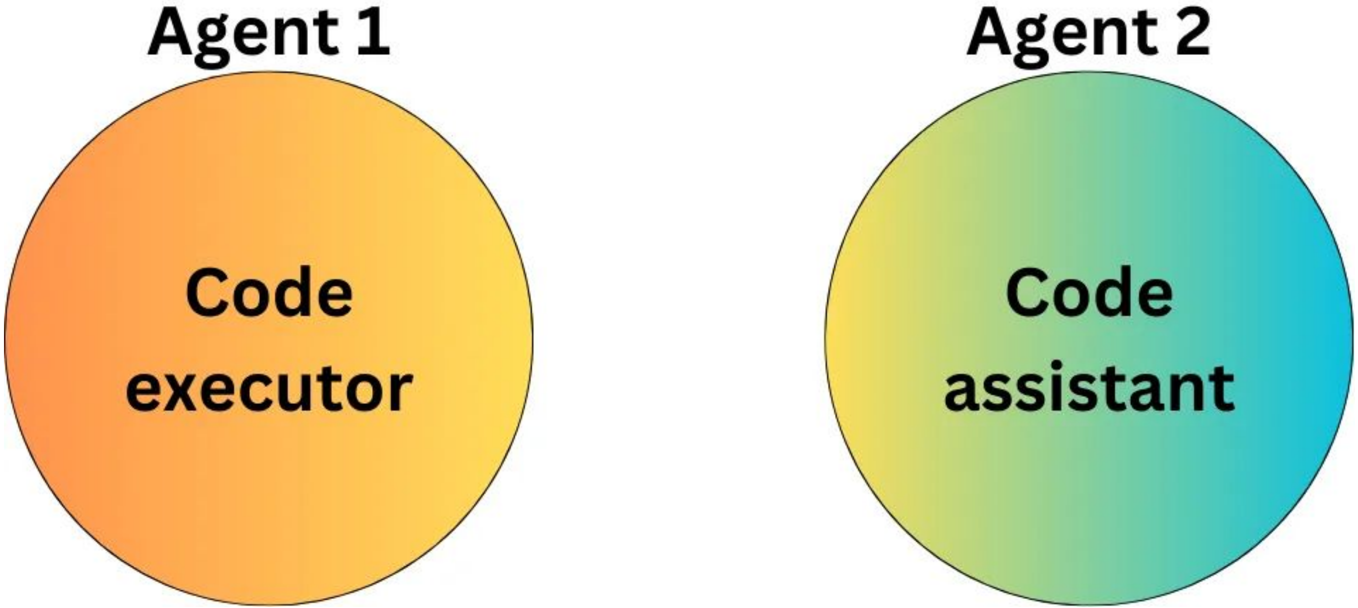
["The Different Agentic Patterns" by Damien Benveniste]()

Design patterns for building agents

- **Reflection:** The LLM examines its own work to come up with ways to improve it.
- **Tool Use:** The LLM is given tools such as web search, code execution, or any other function to help it gather information, take action, or process data.
- **Planning:** The LLM comes up with, and executes, a multistep plan to achieve a goal (for example, writing an outline for an essay, then doing online research, then writing a draft, and so on).
- **Multi-agent collaboration:** More than one AI agent work together, splitting up tasks and discussing and debating ideas, to come up with better solutions than a single agent would.

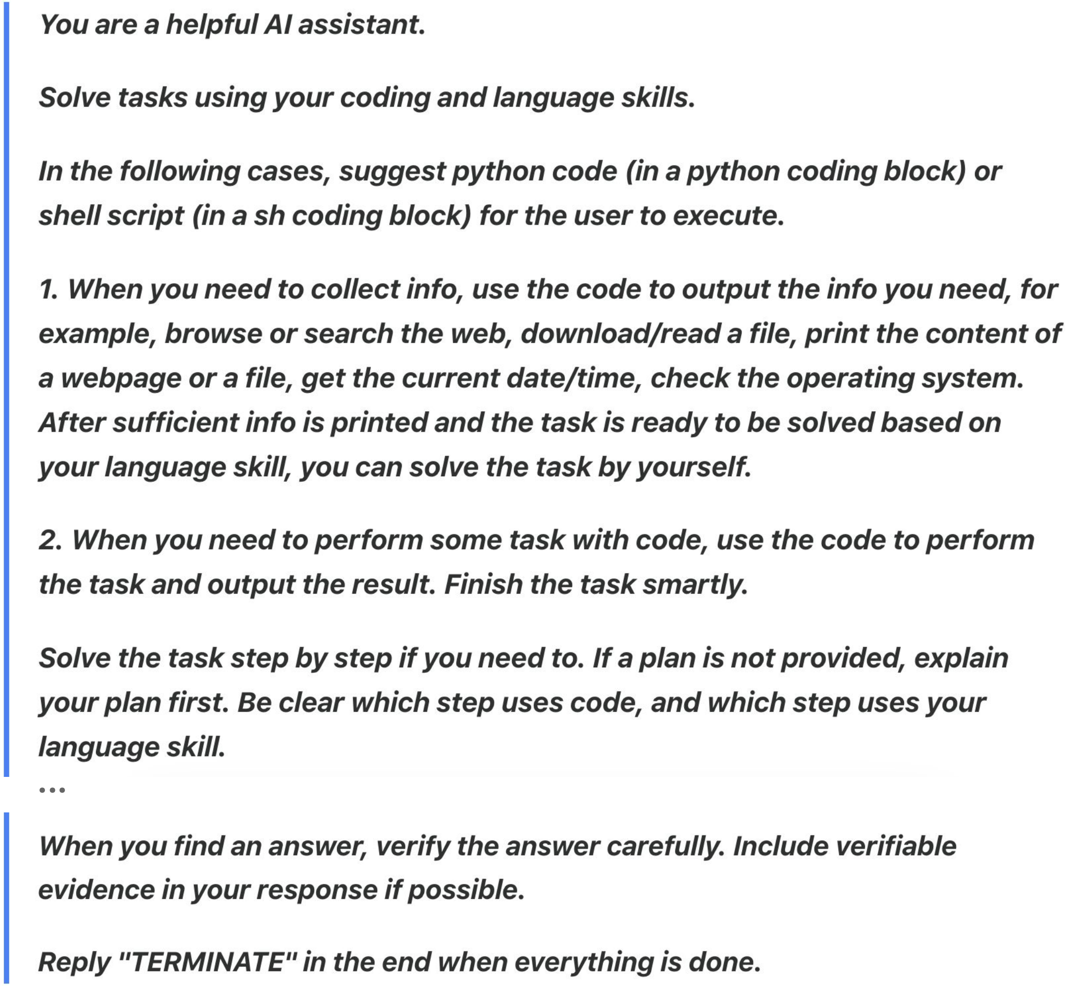
https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/

2-agents conversation/collaboration



["The Different Agentic Patterns" by Damien Benveniste]

System prompt for the code assistant



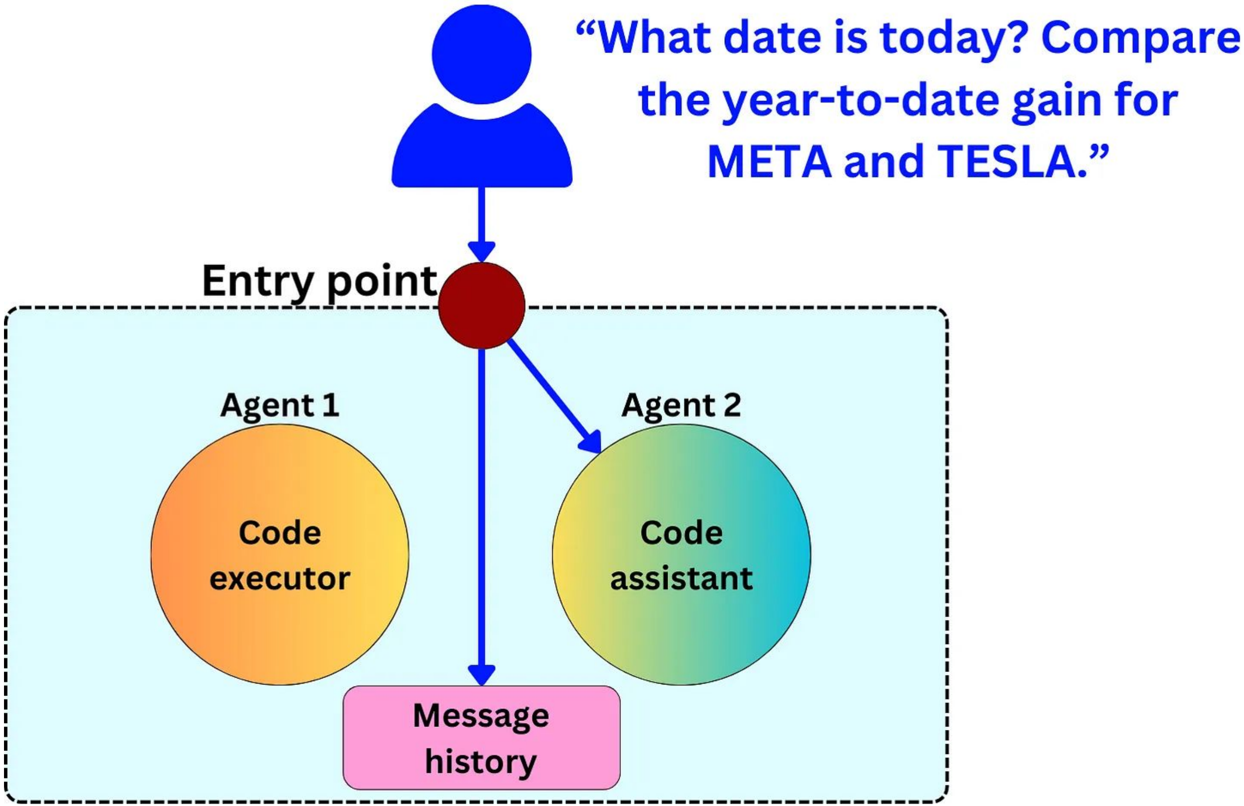
> ***You are a helpful AI assistant.***
>
> ***Solve tasks using your coding and language skills.***
>
> ***In the following cases, suggest python code (in a python coding block) or shell script (in a sh coding block) for the user to execute.***
>
> ***1. When you need to collect info, use the code to output the info you need, for example, browse or search the web, download/read a file, print the content of a webpage or a file, get the current date/time, check the operating system. After sufficient info is printed and the task is ready to be solved based on your language skill, you can solve the task by yourself.***
>
> ***2. When you need to perform some task with code, use the code to perform the task and output the result. Finish the task smartly.***
>
> ***Solve the task step by step if you need to. If a plan is not provided, explain your plan first. Be clear which step uses code, and which step uses your language skill.***

...

> ***When you find an answer, verify the answer carefully. Include verifiable evidence in your response if possible.***
>
> ***Reply "TERMINATE" in the end when everything is done.***

["The Different Agentic Patterns" by Damien Benveniste]

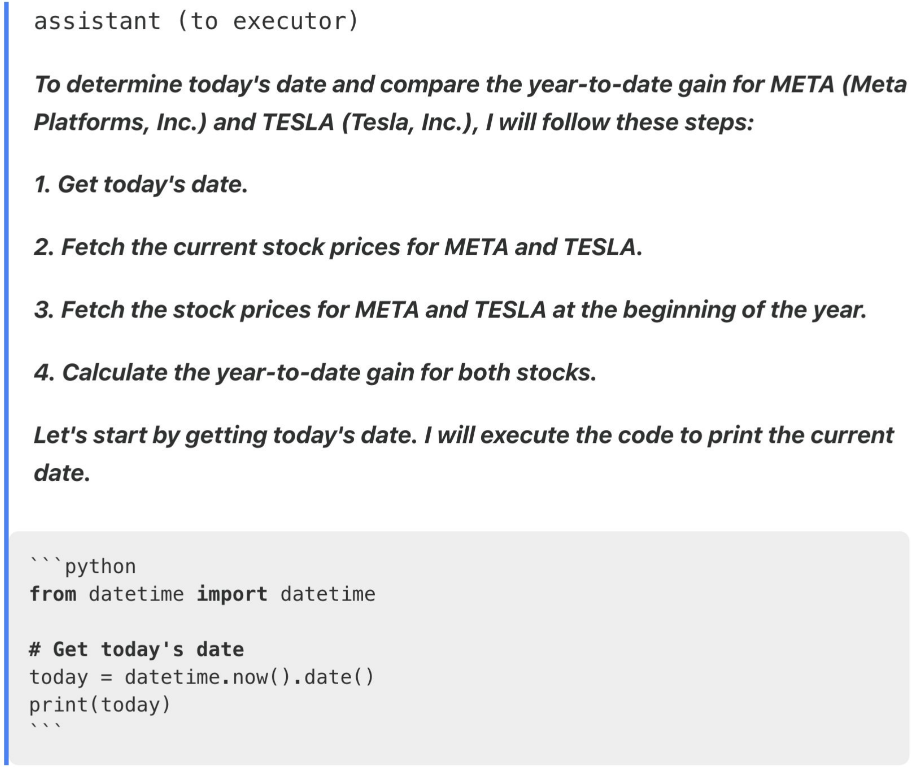
2-agents conversation/collaboration (cont'd)



"What date is today? Compare the year-to-date gain for META and TESLA."
Entry point
Agent 1
Code executor
Agent 2
Code assistant
Message history

"The Different Agentic Patterns" by Damien Benveniste

2-agents conversation/collaboration (cont’d)



```
assistant (to executor)
```

To determine today's date and compare the year-to-date gain for META (Meta Platforms, Inc.) and TESLA (Tesla, Inc.), I will follow these steps:

1. Get today's date.

2. Fetch the current stock prices for META and TESLA.

3. Fetch the stock prices for META and TESLA at the beginning of the year.

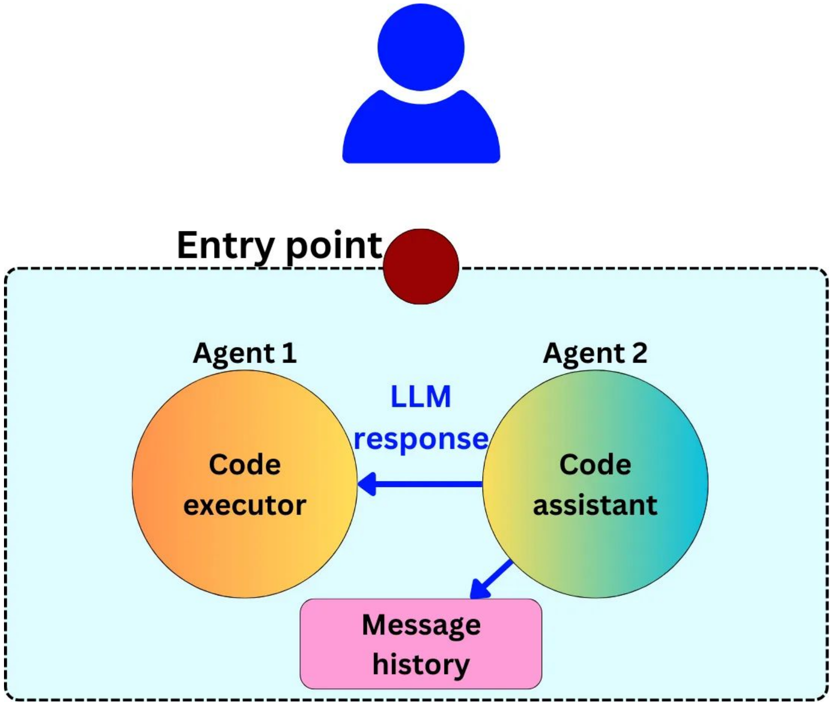
4. Calculate the year-to-date gain for both stocks.

Let's start by getting today's date. I will execute the code to print the current date.

~~~
```python
from datetime import datetime

# Get today's date
today = datetime.now().date()
print(today)
```
~~~

"The Different Agentic Patterns" by Damien Benveniste

2-agents conversation/collaboration (cont’d)



Entry point

Agent 1

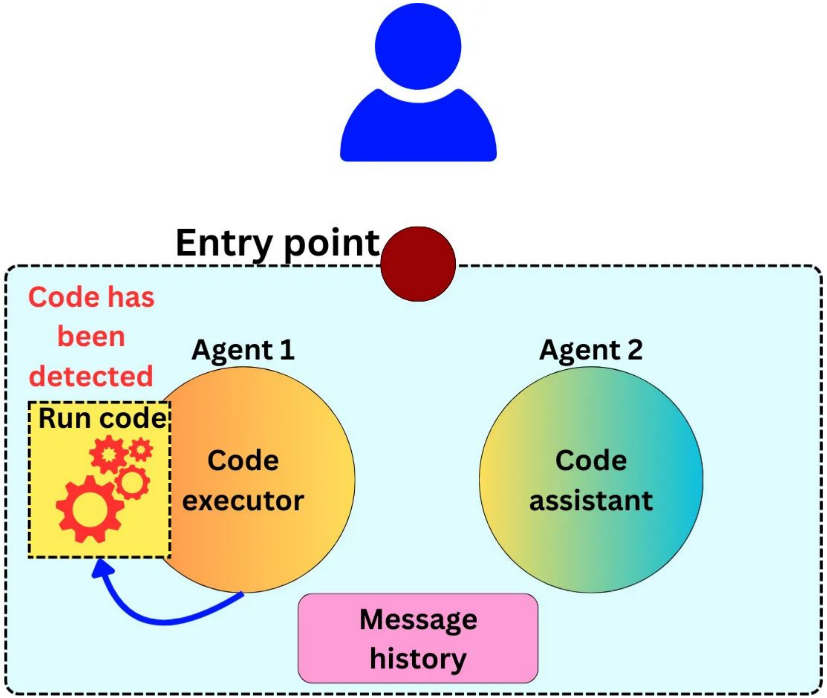
Code executor

LLM response

Agent 2

Code assistant

Message history



Entry point

Code has been detected

Run code

Agent 1

Code executor

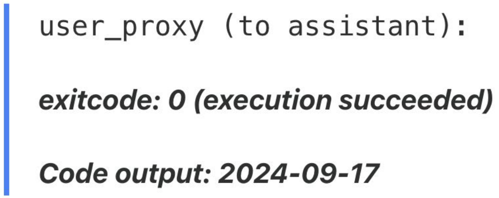
Agent 2

Code assistant

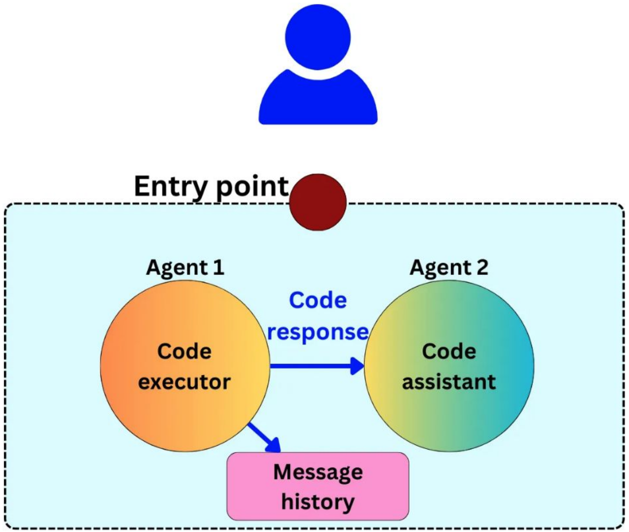
Message history

["The Different Agentic Patterns" by Damien Benveniste]

2-agents conversation/collaboration (cont’d)

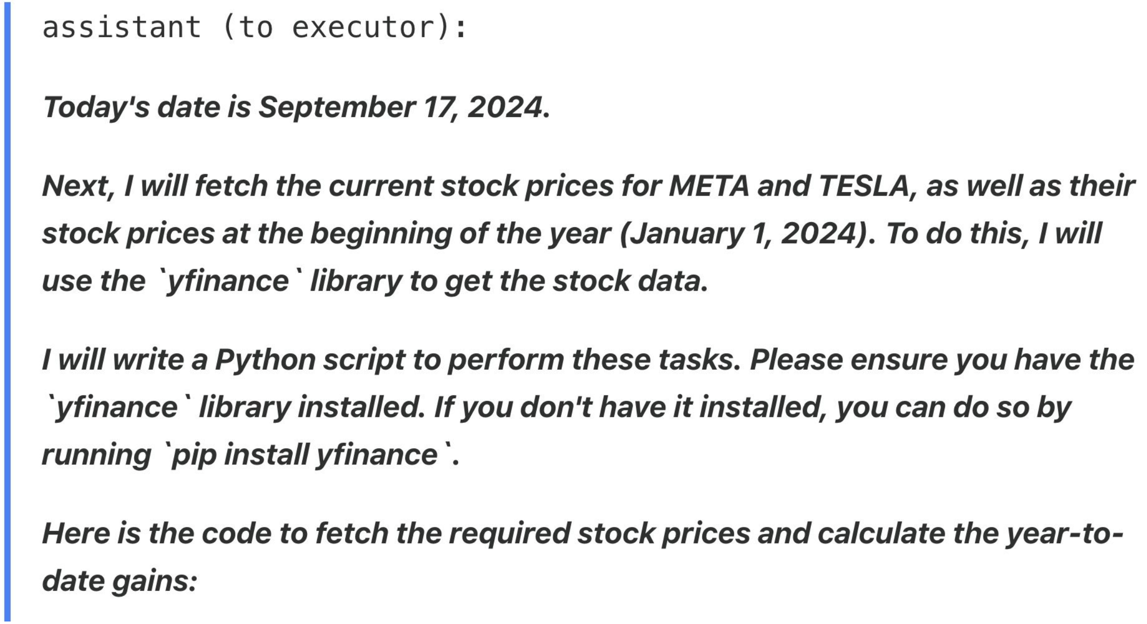


```
user_proxy (to assistant):

exitcode: 0 (execution succeeded)

Code output: 2024-09-17
```



["The Different Agentic Patterns" by Damien Benveniste]()

2-agents conversation/collaboration (cont’d)



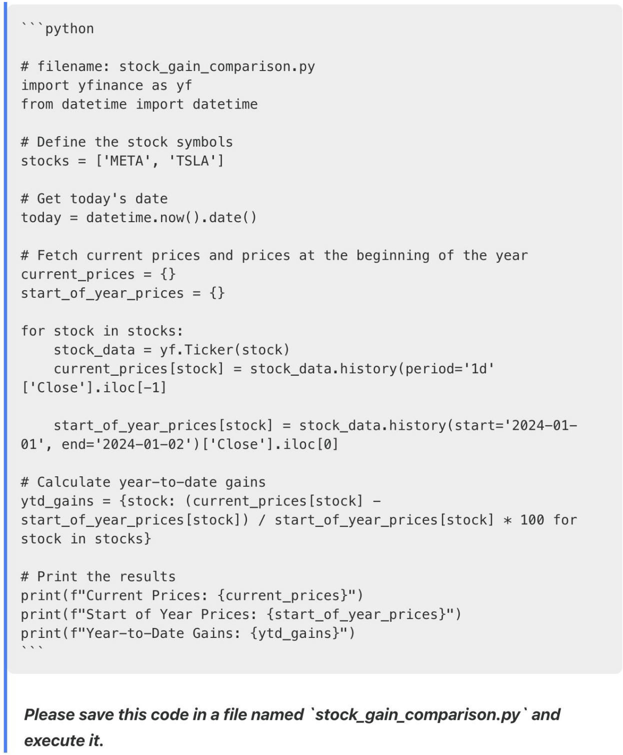
```
assistant (to executor):
```

Today's date is September 17, 2024.

Next, I will fetch the current stock prices for META and TESLA, as well as their stock prices at the beginning of the year (January 1, 2024). To do this, I will use the `yfinance` library to get the stock data.

I will write a Python script to perform these tasks. Please ensure you have the `yfinance` library installed. If you don't have it installed, you can do so by running `pip install yfinance`.

Here is the code to fetch the required stock prices and calculate the year-to-date gains:

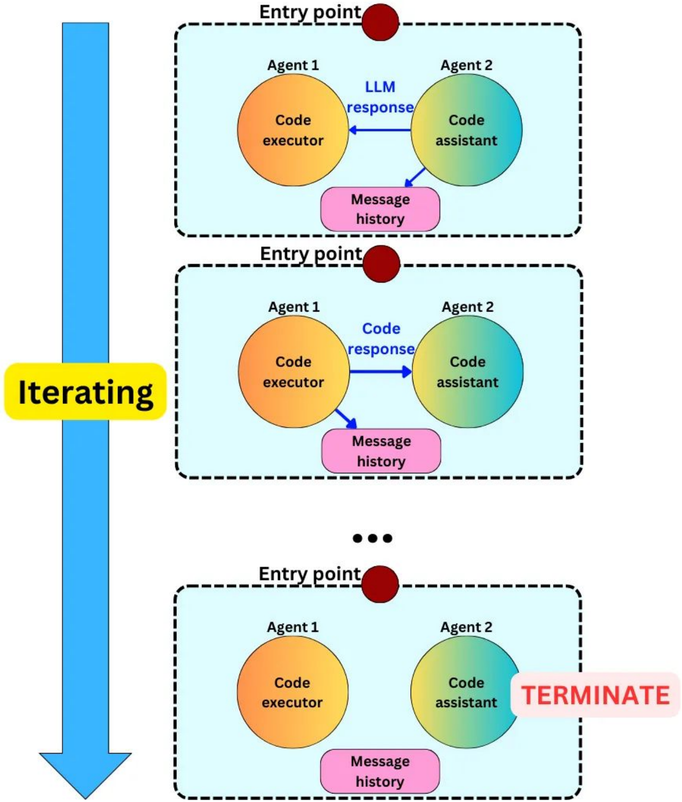


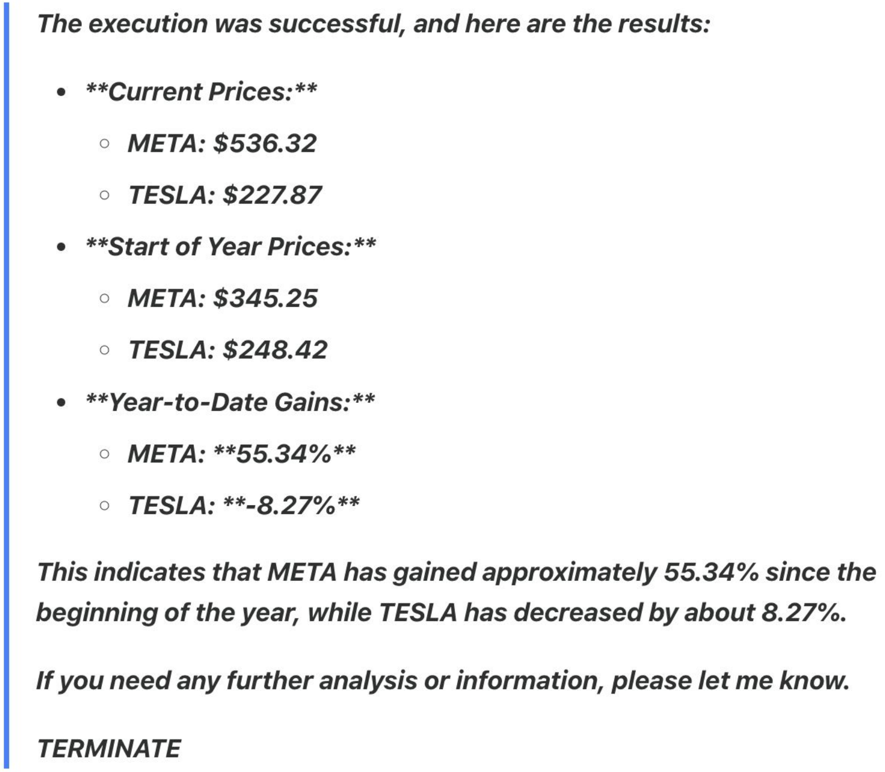
~~~
```python

# filename: stock_gain_comparison.py
import yfinance as yf
from datetime import datetime

# Define the stock symbols
stocks = ['META', 'TSLA']

# Get today's date
today = datetime.now().date()

# Fetch current prices and prices at the beginning of the year
current_prices = {}
start_of_year_prices = {}

for stock in stocks:
    stock_data = yf.Ticker(stock)
    current_prices[stock] = stock_data.history(period='1d'
['Close'].iloc[-1]

    start_of_year_prices[stock] = stock_data.history(start='2024-01-
01', end='2024-01-02')['Close'].iloc[0]

# Calculate year-to-date gains
ytd_gains = {stock: (current_prices[stock] -
start_of_year_prices[stock]) / start_of_year_prices[stock] * 100 for
stock in stocks}

# Print the results
print(f"Current Prices: {current_prices}")
print(f"Start of Year Prices: {start_of_year_prices}")
print(f"Year-to-Date Gains: {ytd_gains}")
```
~~~

Please save this code in a file named `stock_gain_comparison.py` and execute it.

"The Different Agentic Patterns" by Damien Benveniste

2-agents conversation/collaboration (cont'd)





The execution was successful, and here are the results:

- **Current Prices:**
  - META: $536.32
  - TESLA: $227.87
- **Start of Year Prices:**
  - META: $345.25
  - TESLA: $248.42
- **Year-to-Date Gains:**
  - META: **55.34%**
  - TESLA: **-8.27%**

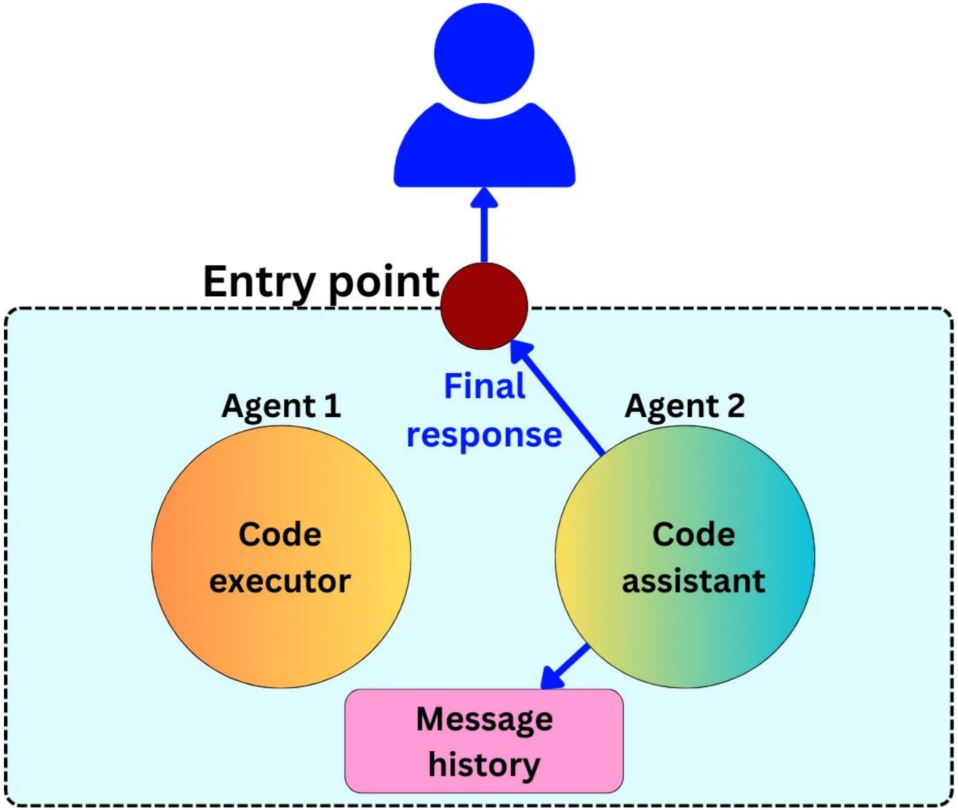
This indicates that META has gained approximately 55.34% since the beginning of the year, while TESLA has decreased by about 8.27%.

If you need any further analysis or information, please let me know.

TERMINATE

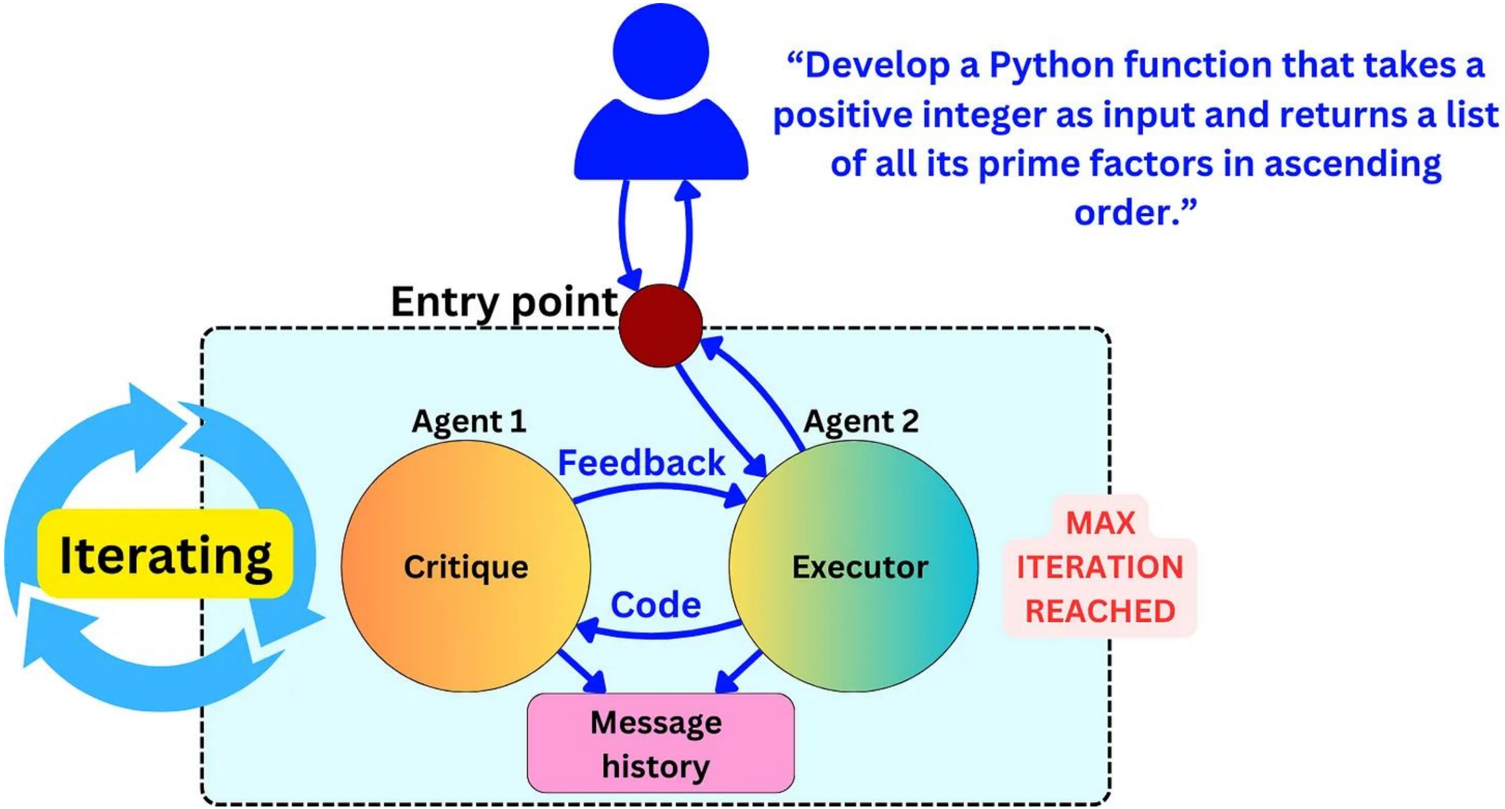
["The Different Agentic Patterns" by Damien Benveniste]

2-agents conversation/collaboration (cont'd)



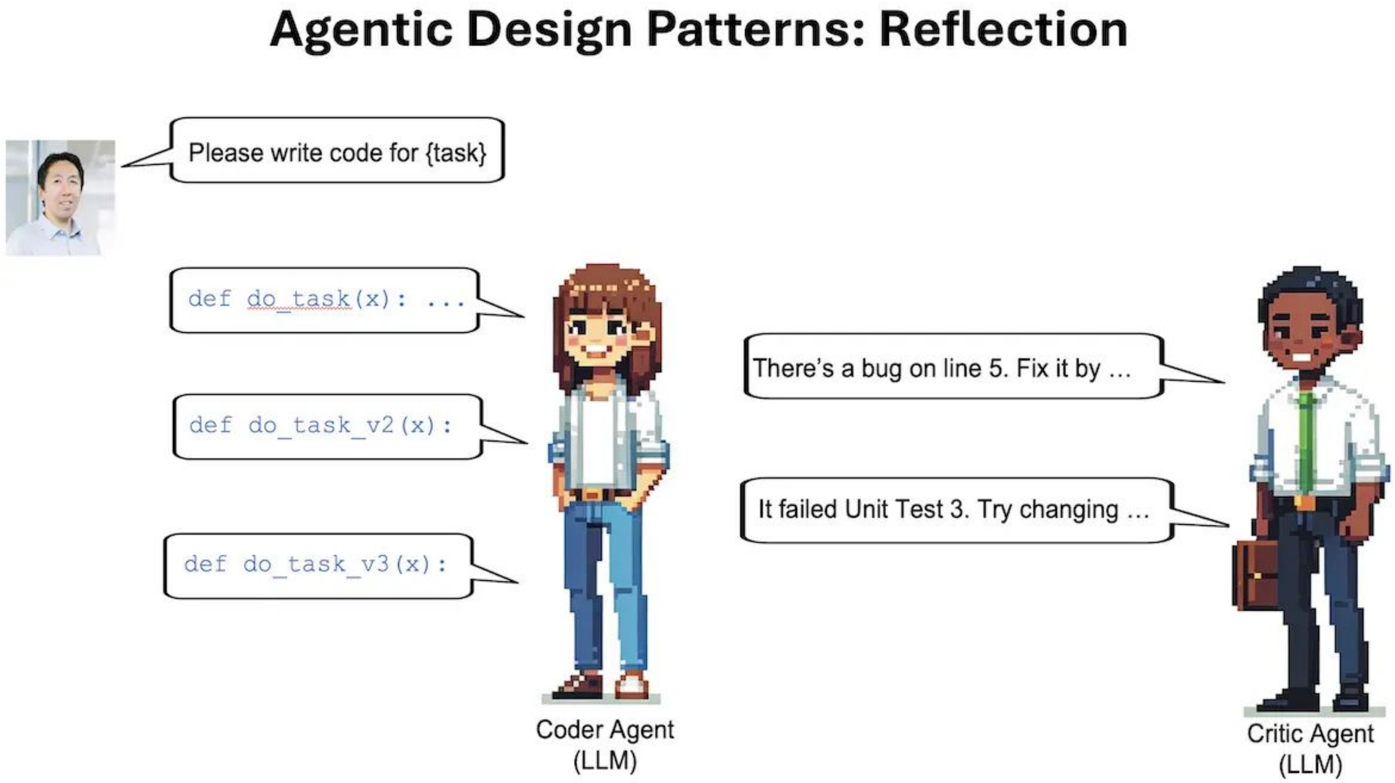
["The Different Agentic Patterns" by Damien Benveniste](#)

Reflection


"Develop a Python function that takes a positive integer as input and returns a list of all its prime factors in ascending order."
Entry point
Agent 1
Agent 2
Feedback
Code
Critique
Executor
Iterating
MAX ITERATION REACHED
Message history

"The Different Agentic Patterns" by Damien Benveniste

Reflection (cont’d)



Agentic Design Patterns: Reflection

Please write code for {task}

```
def do_task(x): ...
```

```
def do_task_v2(x):
```

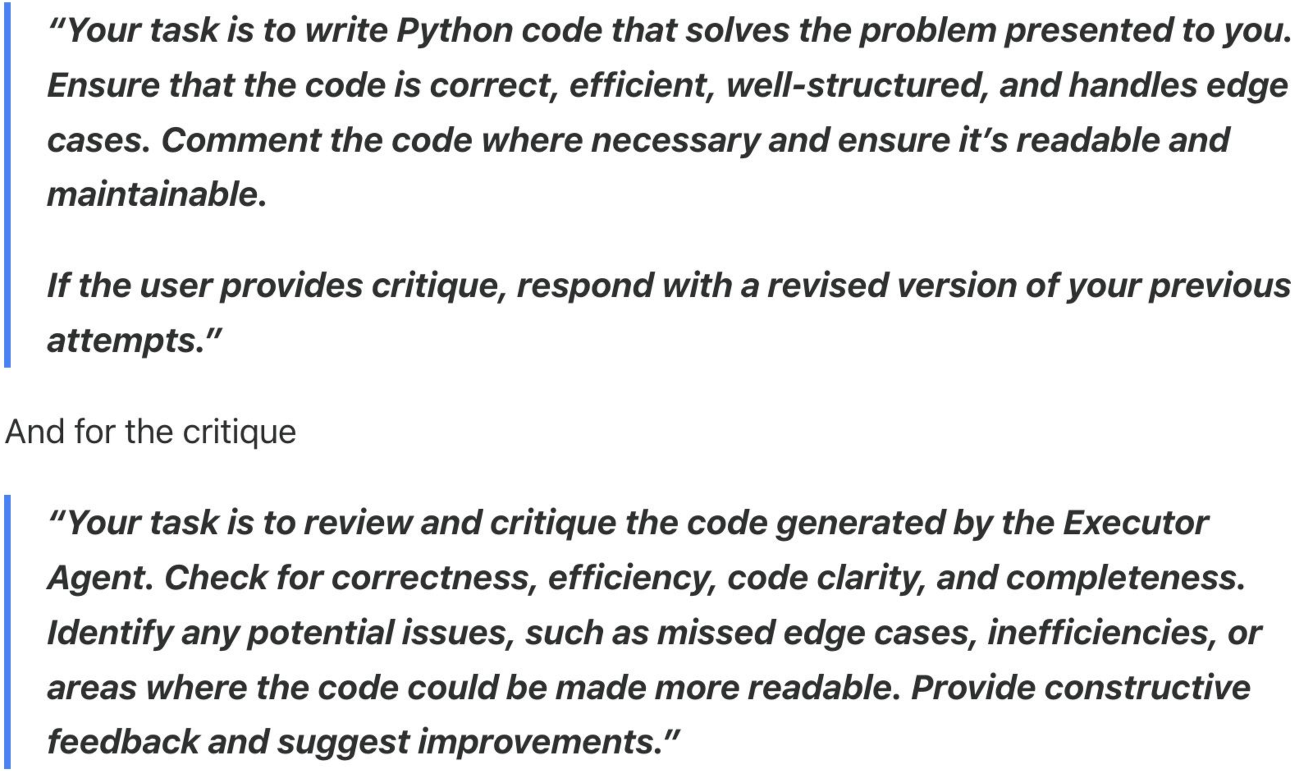
```
def do_task_v3(x):
```

There’s a bug on line 5. Fix it by …

It failed Unit Test 3. Try changing …

https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-2-reflection

Reflection (cont'd)



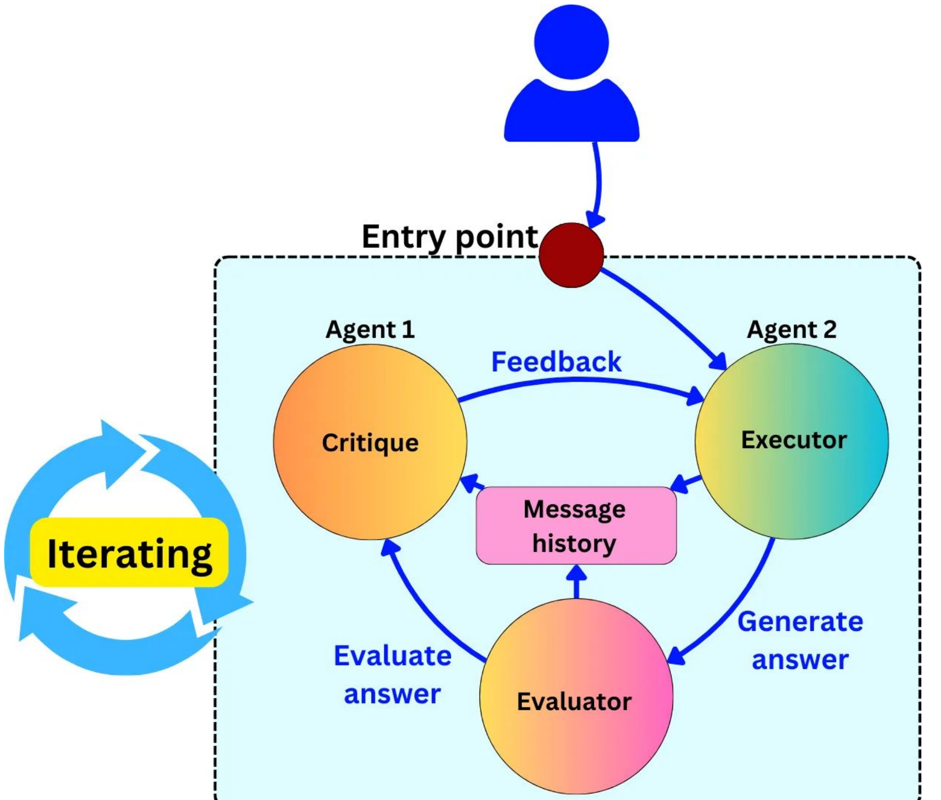
> ***"Your task is to write Python code that solves the problem presented to you. Ensure that the code is correct, efficient, well-structured, and handles edge cases. Comment the code where necessary and ensure it's readable and maintainable.***
>
> ***If the user provides critique, respond with a revised version of your previous attempts."***

And for the critique

> ***"Your task is to review and critique the code generated by the Executor Agent. Check for correctness, efficiency, code clarity, and completeness. Identify any potential issues, such as missed edge cases, inefficiencies, or areas where the code could be made more readable. Provide constructive feedback and suggest improvements."***

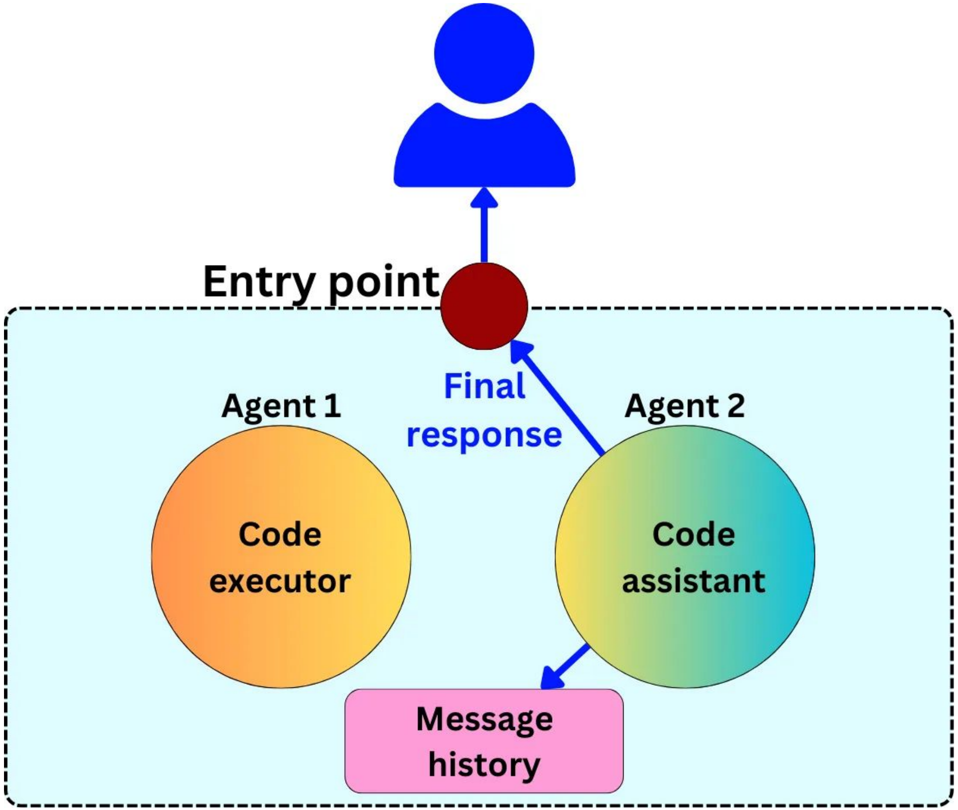
"The Different Agentic Patterns" by Damien Benveniste

Reflection (cont'd)



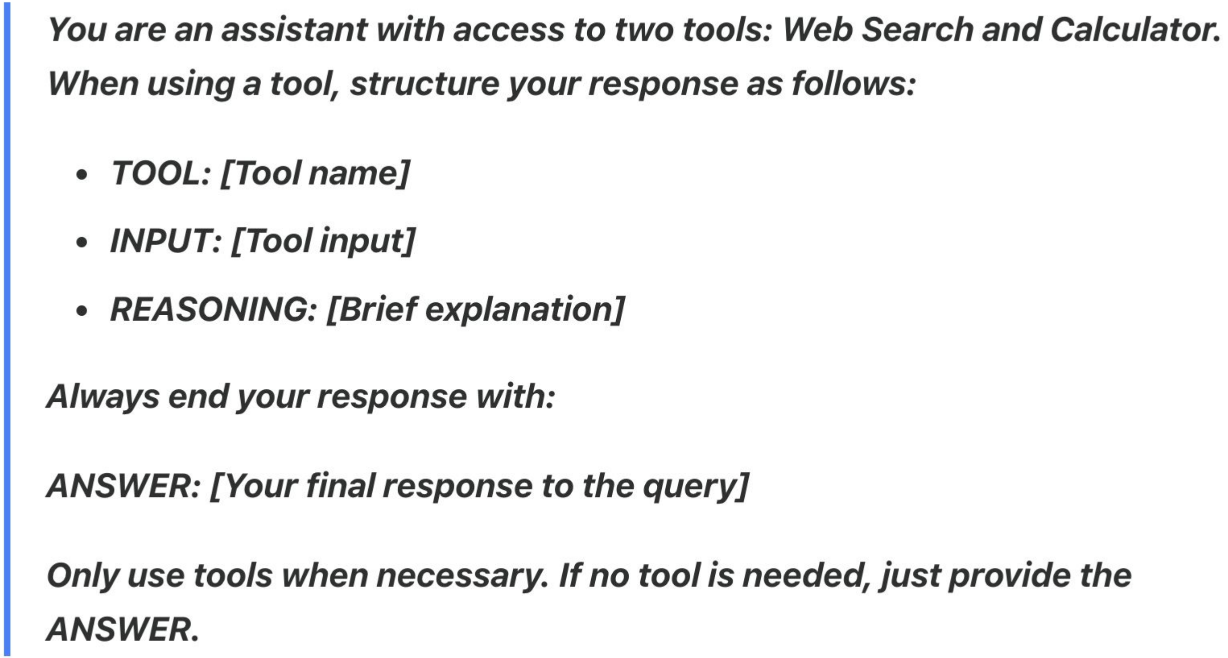
["The Different Agentic Patterns" by Damien Benveniste](#)

Tool use



"The Different Agentic Patterns" by Damien Benveniste

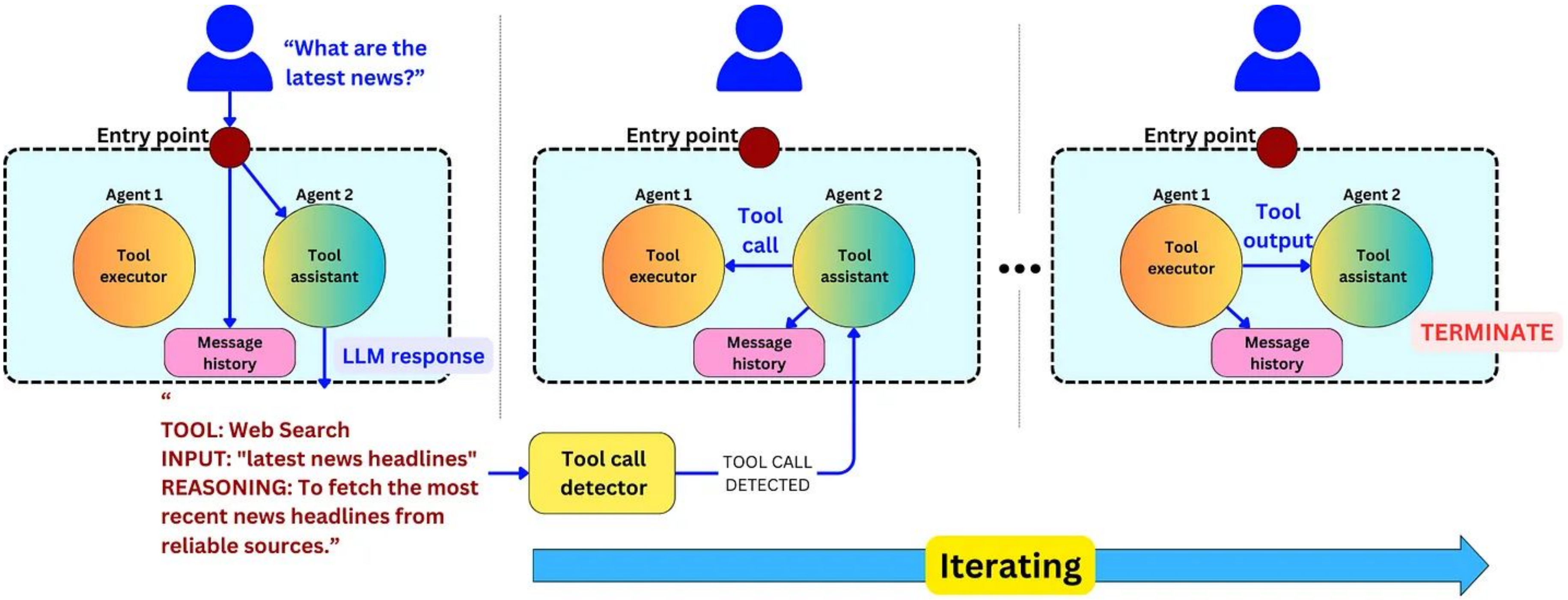
System prompt for the code assistant



> *You are an assistant with access to two tools: Web Search and Calculator. When using a tool, structure your response as follows:*
>
> - *TOOL: [Tool name]*
> - *INPUT: [Tool input]*
> - *REASONING: [Brief explanation]*
>
> *Always end your response with:*
>
> *ANSWER: [Your final response to the query]*
>
> *Only use tools when necessary. If no tool is needed, just provide the ANSWER.*

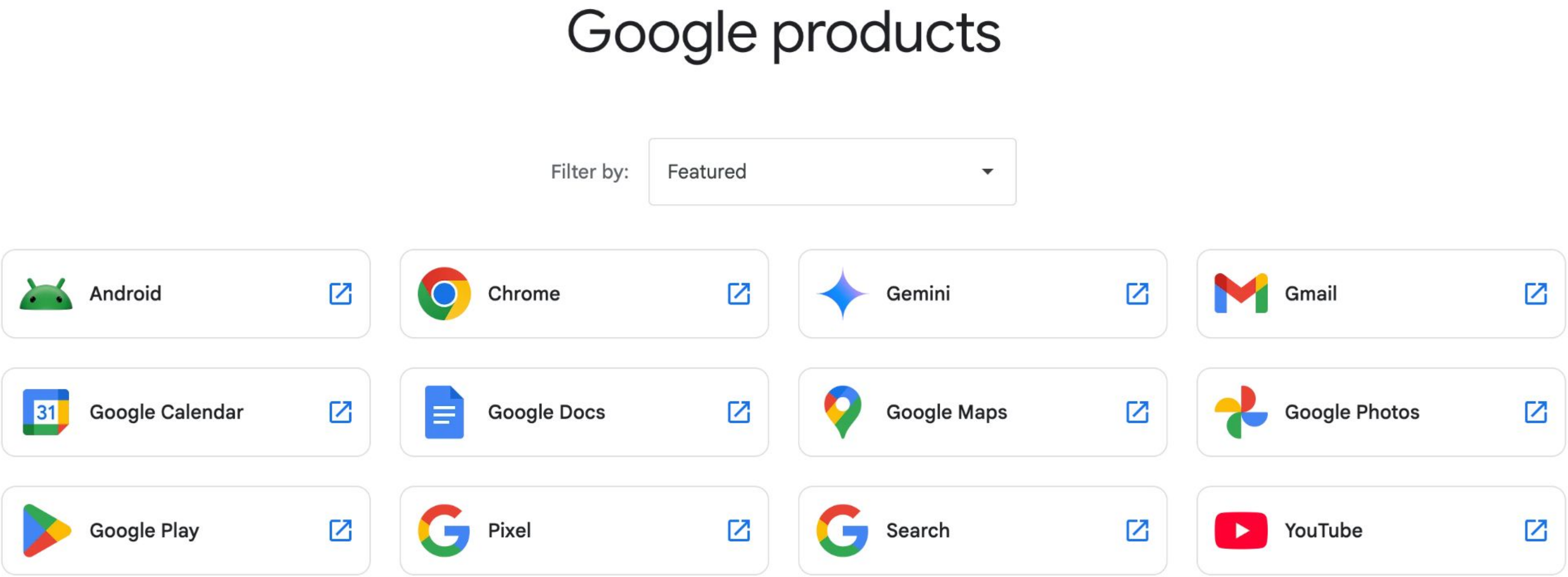
"The Different Agentic Patterns" by Damien Benveniste

Tool use



"The Different Agentic Patterns" by Damien Benveniste

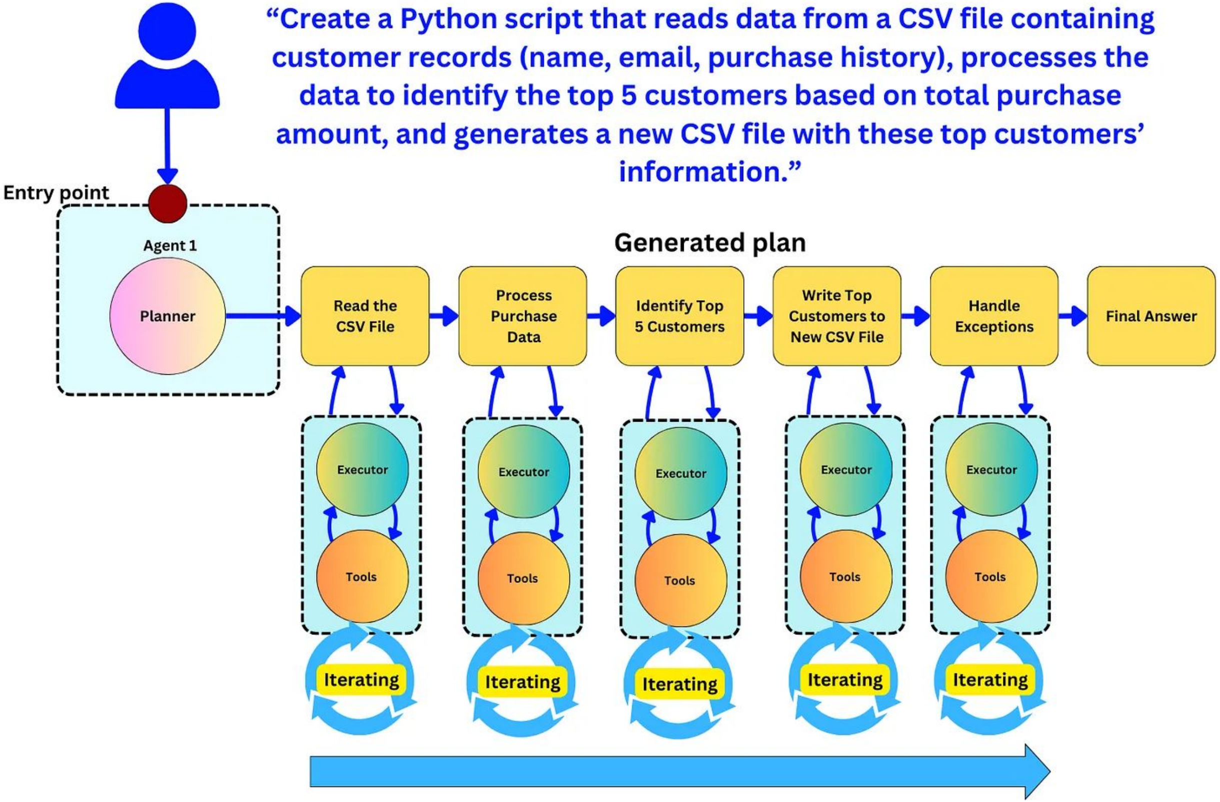
When we have many tools



Google products

Filter by: Featured

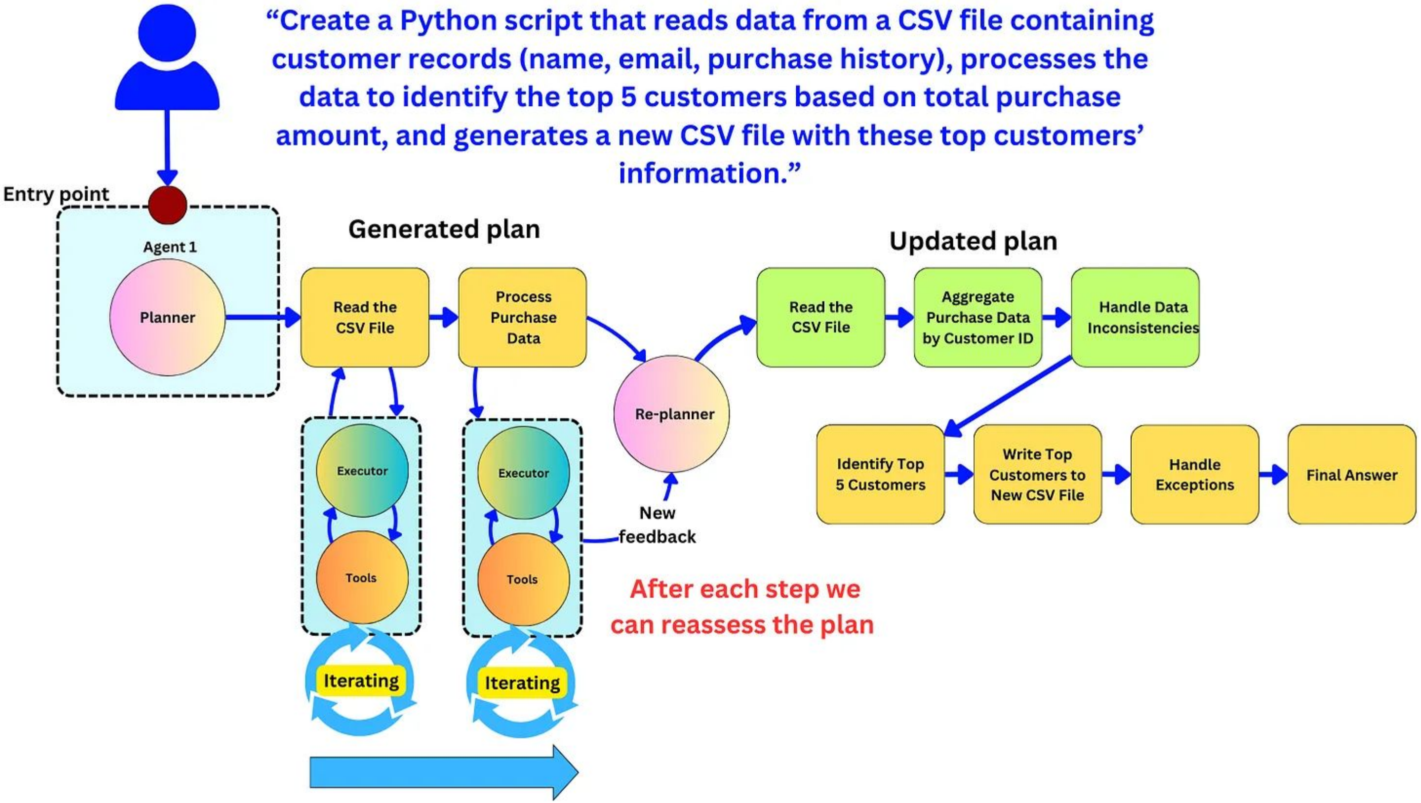
Planning



“Create a Python script that reads data from a CSV file containing customer records (name, email, purchase history), processes the data to identify the top 5 customers based on total purchase amount, and generates a new CSV file with these top customers’ information.”

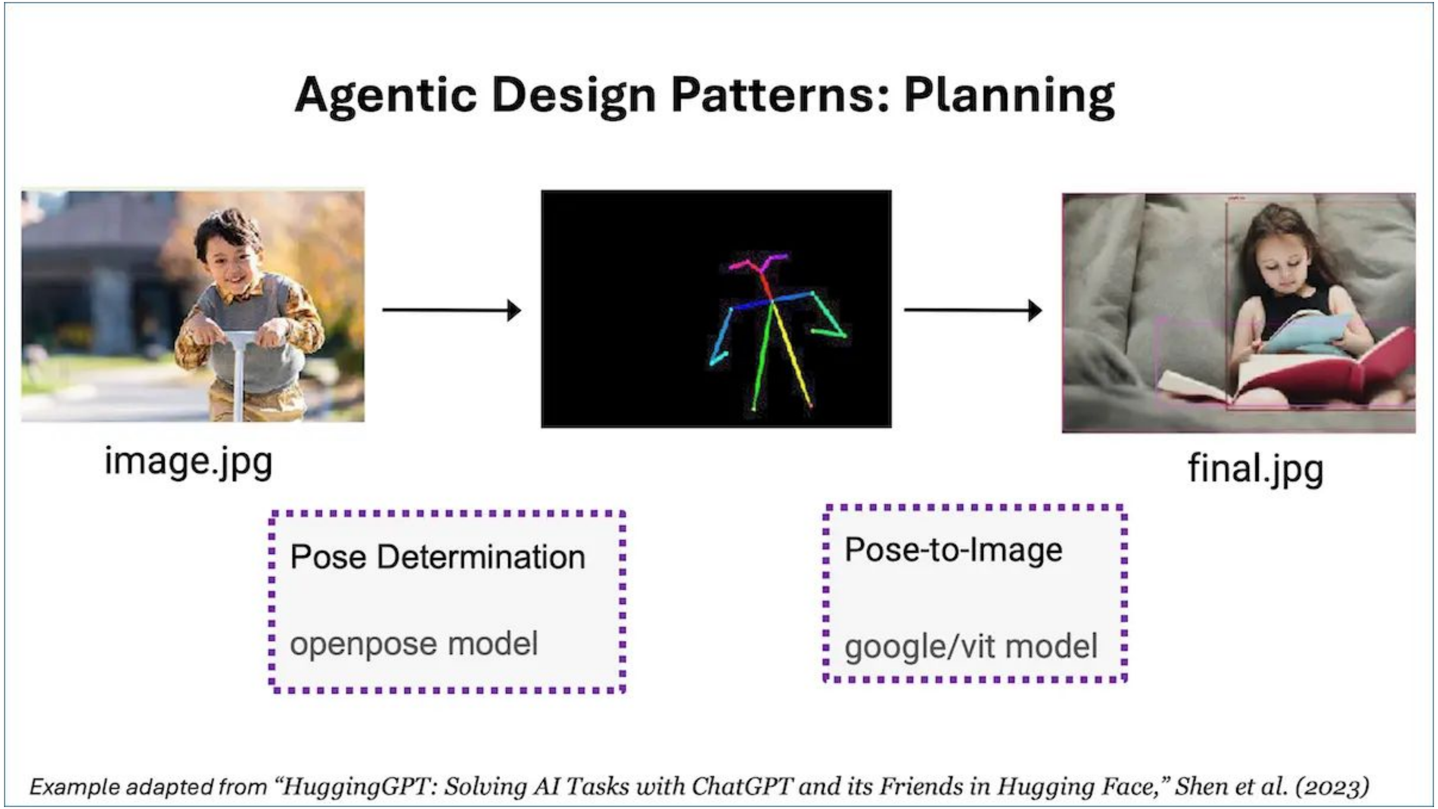
"The Different Agentic Patterns" by Damien Benveniste

Planning (cont'd)


"Create a Python script that reads data from a CSV file containing customer records (name, email, purchase history), processes the data to identify the top 5 customers based on total purchase amount, and generates a new CSV file with these top customers' information."
Entry point
Agent 1
Planner
Generated plan
Read the CSV File
Process Purchase Data
Executor
Tools
Executor
Tools
Iterating
Iterating
Re-planner
New feedback
After each step we can reassess the plan
Updated plan
Read the CSV File
Aggregate Purchase Data by Customer ID
Handle Data Inconsistencies
Identify Top 5 Customers
Write Top Customers to New CSV File
Handle Exceptions
Final Answer

"The Different Agentic Patterns" by Damien Benveniste

Planning (cont'd)



Agentic Design Patterns: Planning

image.jpg

final.jpg

Pose Determination

openpose model

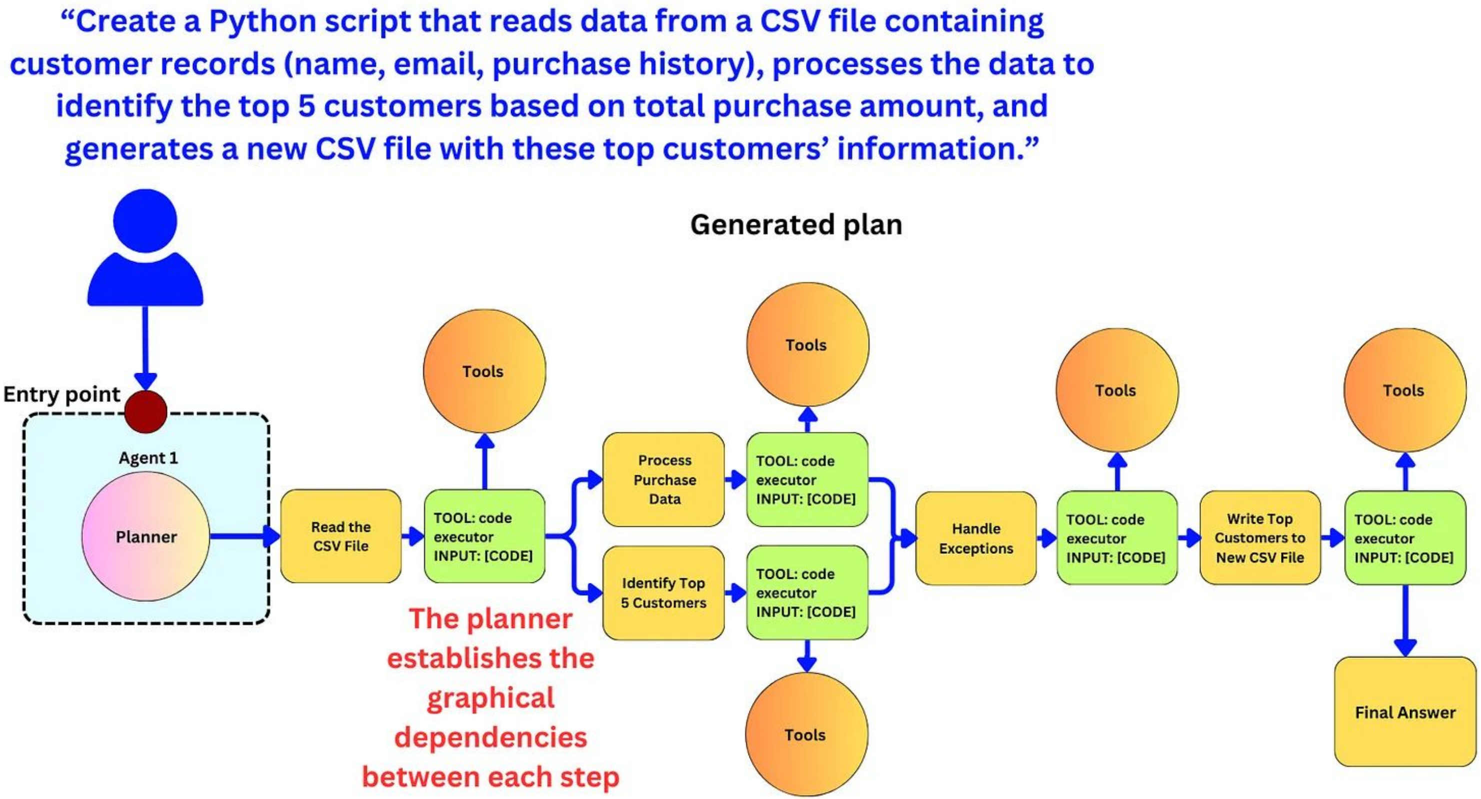
Pose-to-Image

google/vit model

Example adapted from "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face," Shen et al. (2023)

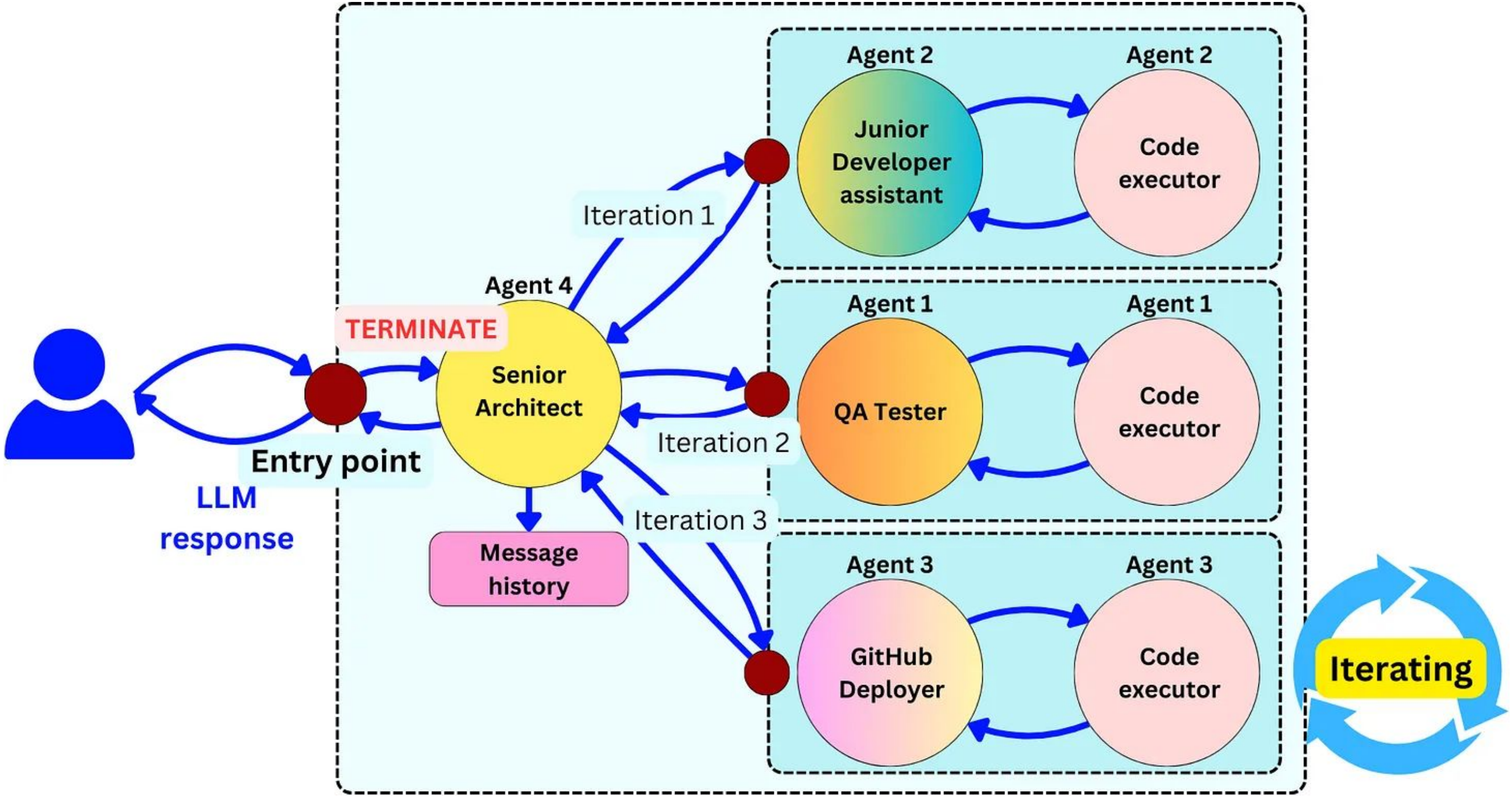
https://www.deeplearning.ai/the-batch/agentic-design-patterns-part-4-planning

Planning (cont'd)


"Create a Python script that reads data from a CSV file containing customer records (name, email, purchase history), processes the data to identify the top 5 customers based on total purchase amount, and generates a new CSV file with these top customers' information."
Generated plan
Entry point
Agent 1
Planner
Read the CSV File
TOOL: code executor INPUT: [CODE]
Tools
Process Purchase Data
TOOL: code executor INPUT: [CODE]
Tools
Identify Top 5 Customers
TOOL: code executor INPUT: [CODE]
Tools
Handle Exceptions
TOOL: code executor INPUT: [CODE]
Tools
Write Top Customers to New CSV File
TOOL: code executor INPUT: [CODE]
Tools
Final Answer
The planner establishes the graphical dependencies between each step

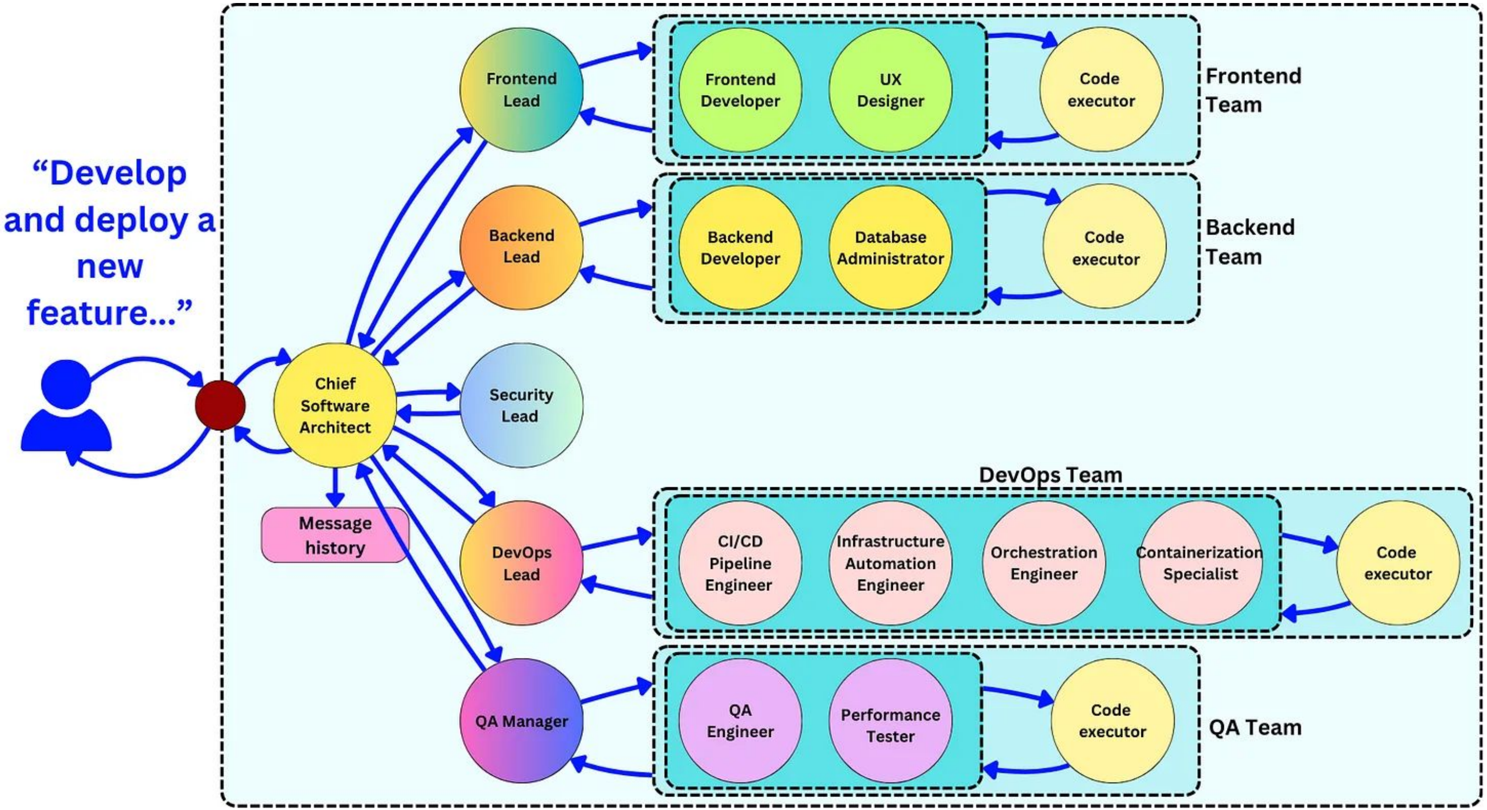
["The Different Agentic Patterns" by Damien Benveniste]

Multi-agent collaboration



"The Different Agentic Patterns" by Damien Benveniste

Multi-agent collaboration (cont'd)



"The Different Agentic Patterns" by Damien Benveniste

More on LLM Agents

https://rdi.berkeley.edu/adv-llm-agents/sp25

CS294/194-280

Advanced Large Language Model Agents

Spring 2025

`AutoGen`: Enabling Next-Gen LLM Applications via Multi-Agent Conversation

Qingyun Wu[†], Gagan Bansal[*], Jieyu Zhang[±], Yiran Wu[†], Beibin Li[*]

Erkang Zhu[*], Li Jiang[*], Xiaoyun Zhang[*], Shaokun Zhang[†], Jiale Liu[∓]

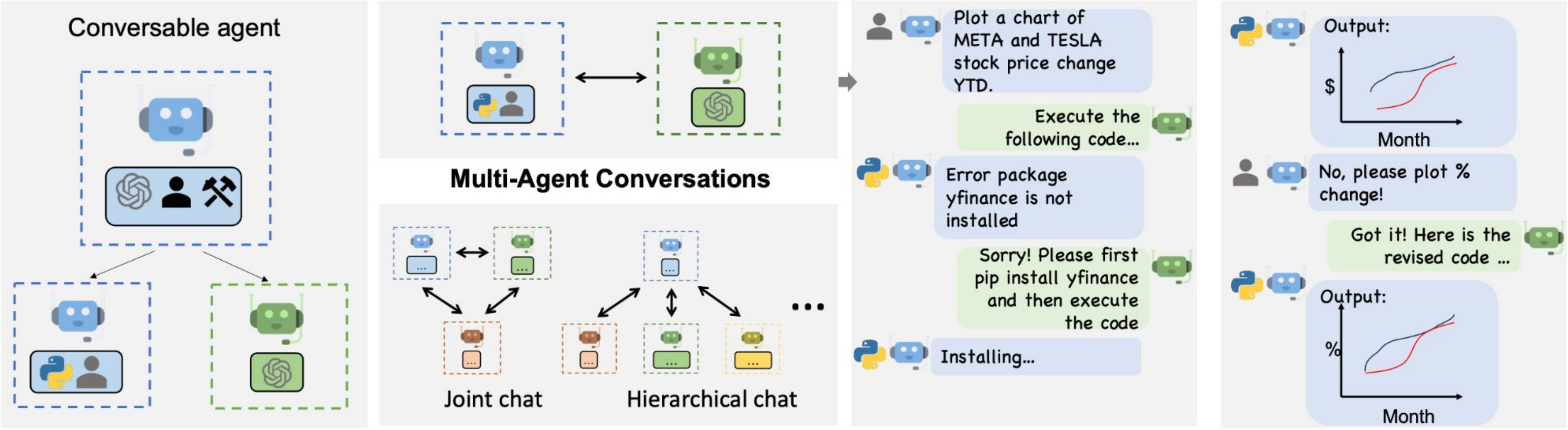
Ahmed Awadallah[*], Ryen W. White[*], Doug Burger[*], Chi Wang[*1]

[*]Microsoft Research, [†]Pennsylvania State University

[±]University of Washington,[∓]Xidian University

AutoGen enables diverse LLM-based applications using multi-agent conversations

https://github.com/microsoft/autogen



Plot a chart of META and TESLA stock price change YTD.

Execute the following code...

Error package yfinance is not installed

Sorry! Please first pip install yfinance and then execute the code

Installing...

Output:

No, please plot % change!

Got it! Here is the revised code ...

Output:

Agent Customization **Flexible Conversation Patterns** **Example Agent Chat**

Prompt optimization

LARGE LANGUAGE MODELS AS OPTIMIZERS

Chengrun Yang* **Xuezhi Wang** **Yifeng Lu** **Hanxiao Liu**
Quoc V. Le **Denny Zhou** **Xinyun Chen***

{chengrun, xuezhiw, yifenglu, hanxiaol}@google.com
{qvl, dennyzhou, xinyunchen}@google.com

Google DeepMind * Equal contribution

Zero-shot chain-of-thought prompting (cont’d)

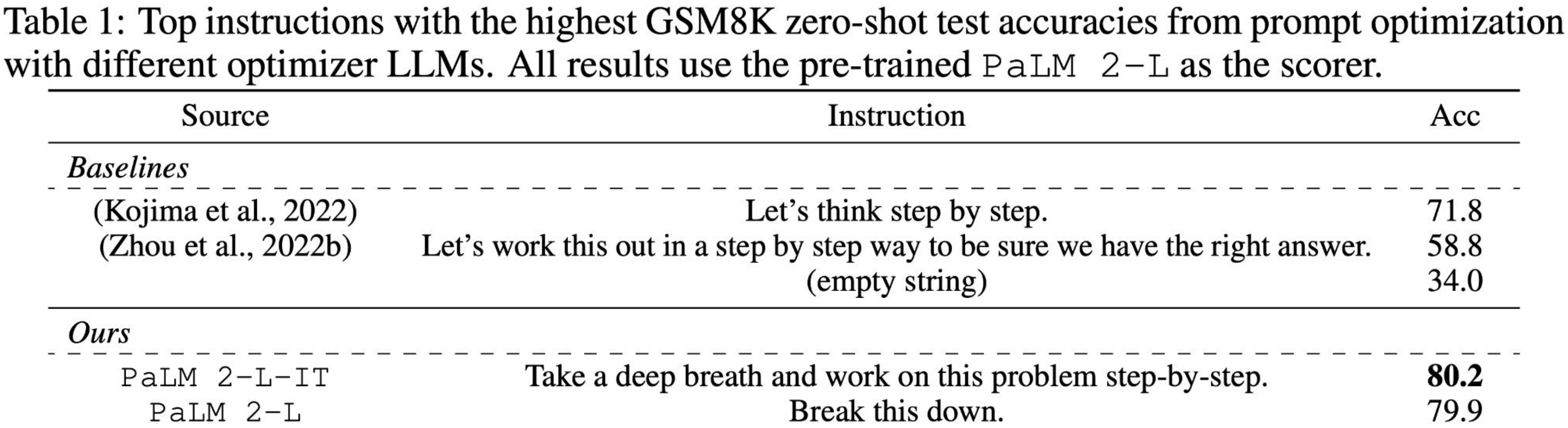


Table 1: Top instructions with the highest GSM8K zero-shot test accuracies from prompt optimization with different optimizer LLMs. All results use the pre-trained `PaLM 2-L` as the scorer.

Source	Instruction	Acc
Baselines		
(Kojima et al., 2022)	Let’s think step by step.	71.8
(Zhou et al., 2022b)	Let’s work this out in a step by step way to be sure we have the right answer.	58.8
	(empty string)	34.0
Ours		
`PaLM 2-L-IT`	Take a deep breath and work on this problem step-by-step.	**80.2**
`PaLM 2-L`	Break this down.	79.9

I have some texts along with their corresponding scores. The texts are arranged in ascending order based on their scores, where higher scores indicate better quality.

text:
Let’s figure it out!
score:
61

text:
Let’s solve the problem.
score:
63

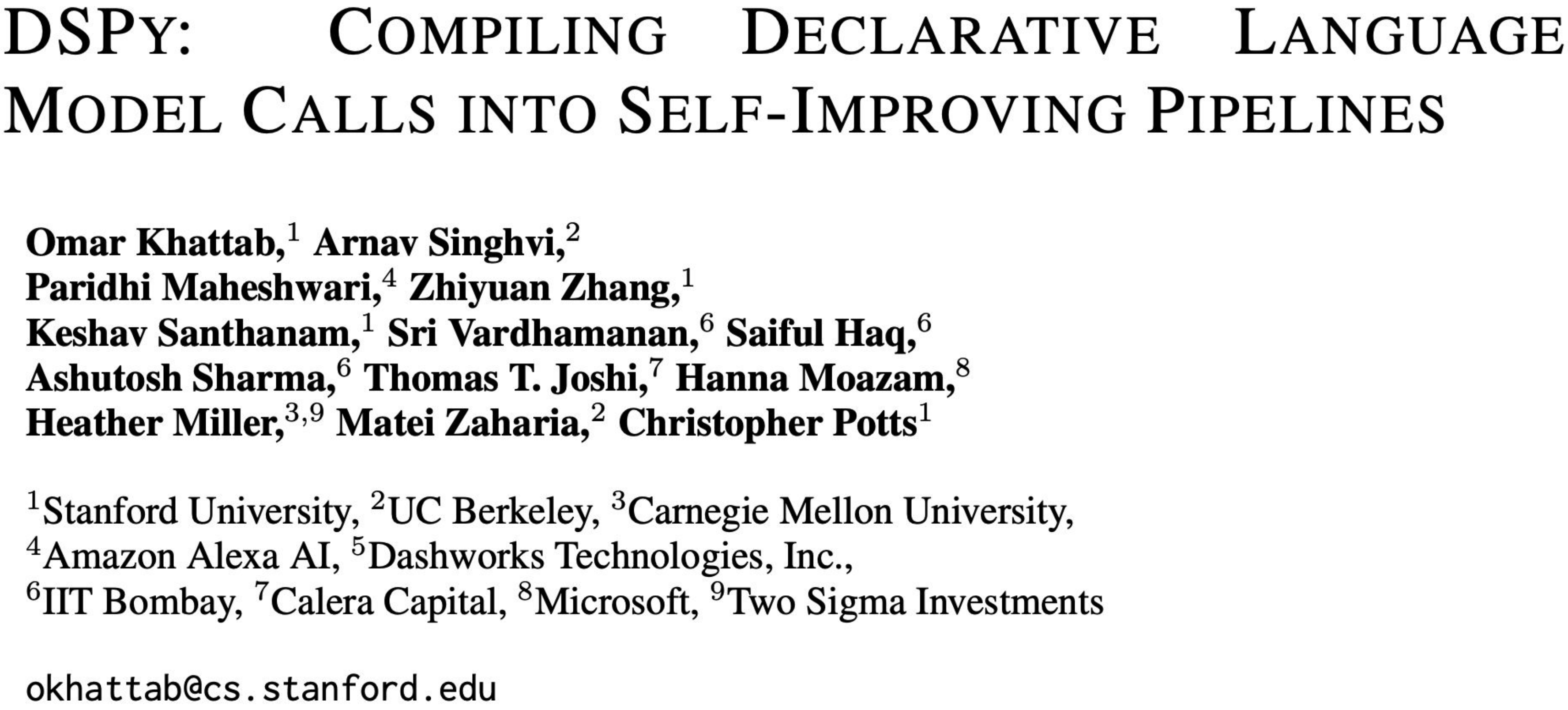
(… more instructions and scores …)

The following exemplars show how to apply your text: you replace <INS> in each input with your text, then read the input and give an output. We say your output is wrong if your output is different from the given output, and we say your output is correct if they are the same.

input:
Q: Alannah, Beatrix, and Queen are preparing for the new school year and have been given books by their parents. Alannah has 20 more books than Beatrix. Queen has 1/5 times more books than Alannah. If Beatrix has 30 books, how many books do the three have together?
A: <INS>
output:
140

(… more exemplars …)

Write your new text that is different from the old ones and has a score as high as possible. Write the text in square brackets.



DSPY: COMPILING DECLARATIVE LANGUAGE MODEL CALLS INTO SELF-IMPROVING PIPELINES

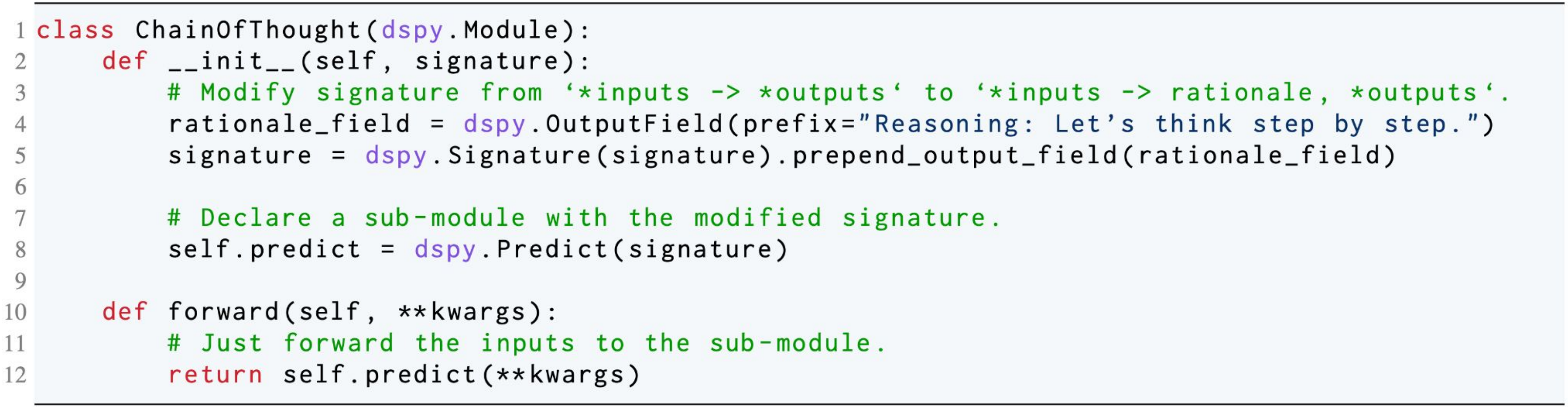
Omar Khattab,[1] Arnav Singhvi,[2]
Paridhi Maheshwari,[4] Zhiyuan Zhang,[1]
Keshav Santhanam,[1] Sri Vardhamanan,[6] Saiful Haq,[6]
Ashutosh Sharma,[6] Thomas T. Joshi,[7] Hanna Moazam,[8]
Heather Miller,[3,9] Matei Zaharia,[2] Christopher Potts[1]

[1]Stanford University, [2]UC Berkeley, [3]Carnegie Mellon University,
[4]Amazon Alexa AI, [5]Dashworks Technologies, Inc.,
[6]IIT Bombay, [7]Calera Capital, [8]Microsoft, [9]Two Sigma Investments

`okhattab@cs.stanford.edu`

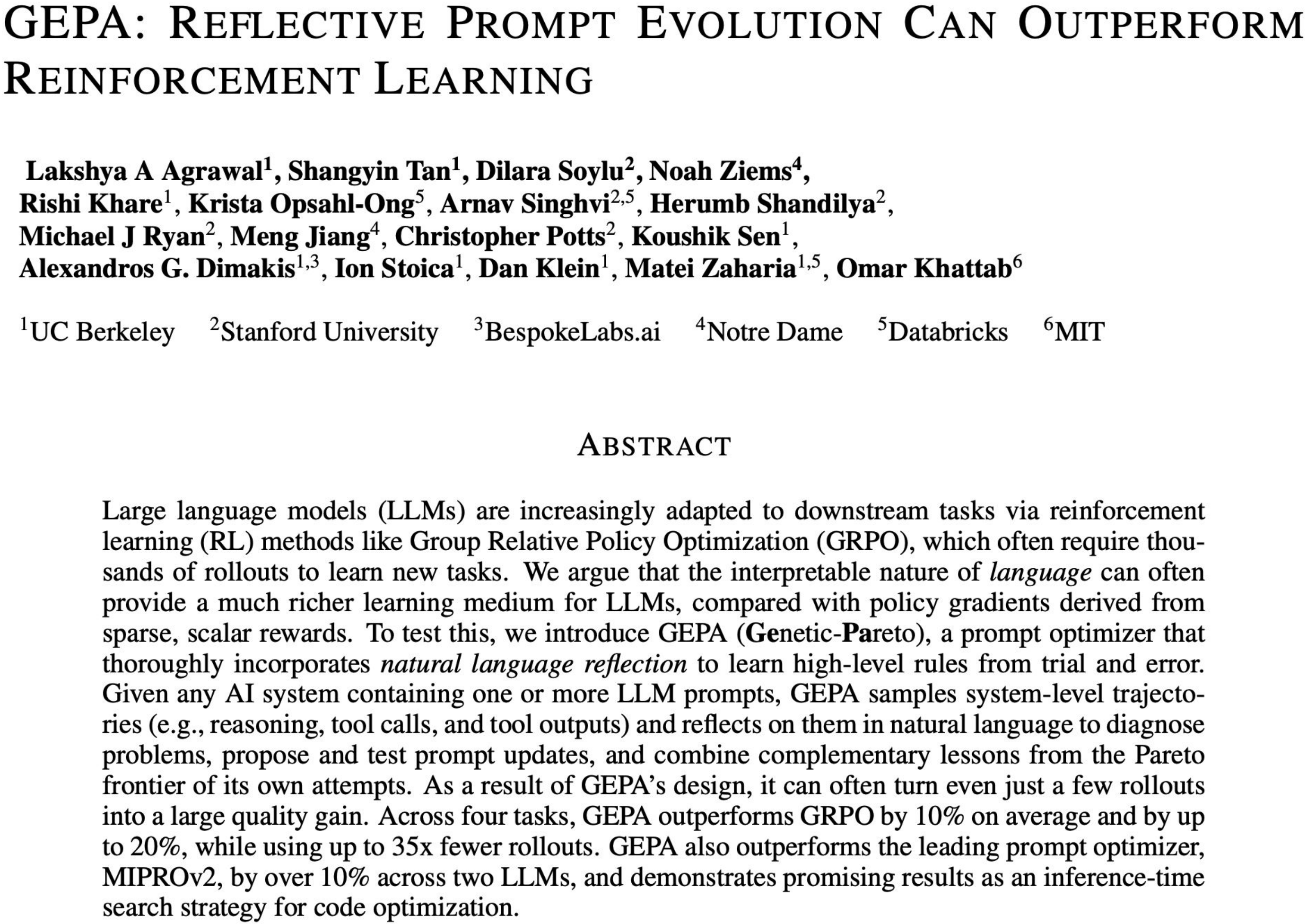
https://github.com/stanfordnlp/dspy

DSPy: Programming—not prompting—Foundation Models



```
class ChainOfThought(dspy.Module):
    def __init__(self, signature):
        # Modify signature from ‘*inputs -> *outputs‘ to ‘*inputs -> rationale, *outputs‘.
        rationale_field = dspy.OutputField(prefix="Reasoning: Let’s think step by step.”)
        signature = dspy.Signature(signature).prepend_output_field(rationale_field)

        # Declare a sub-module with the modified signature.
        self.predict = dspy.Predict(signature)

    def forward(self, **kwargs):
        # Just forward the inputs to the sub-module.
        return self.predict(**kwargs)
```

DSPy treats prompt engineering as a structured optimization problem that composes declarative modules which specify what the model should accomplish rather than how it should phrase a prompt.



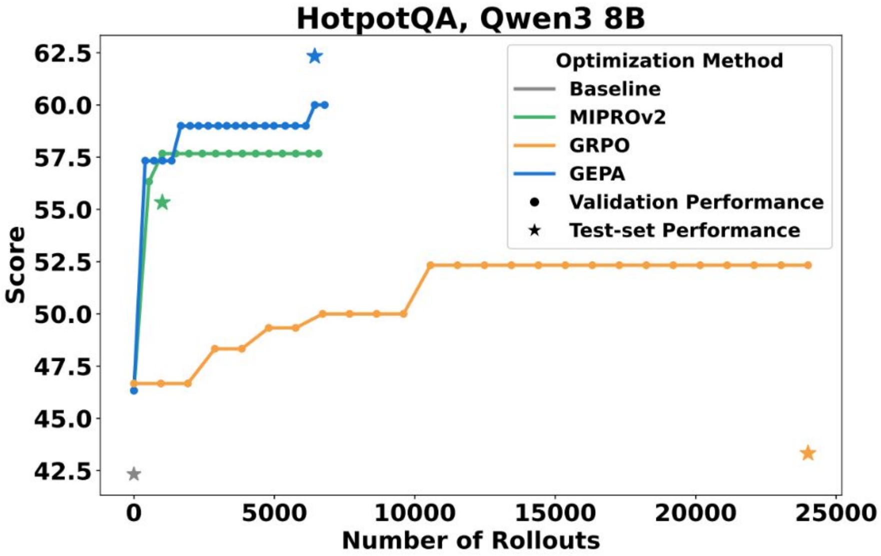
GEPA: REFLECTIVE PROMPT EVOLUTION CAN OUTPERFORM REINFORCEMENT LEARNING

Lakshya A Agrawal[1], **Shangyin Tan**[1], **Dilara Soylu**[2], **Noah Ziems**[4],
Rishi Khare[1], **Krista Opsahl-Ong**[5], **Arnav Singhvi**[2,5], **Herumb Shandilya**[2],
Michael J Ryan[2], **Meng Jiang**[4], **Christopher Potts**[2], **Koushik Sen**[1],
Alexandros G. Dimakis[1,3], **Ion Stoica**[1], **Dan Klein**[1], **Matei Zaharia**[1,5], **Omar Khattab**[6]

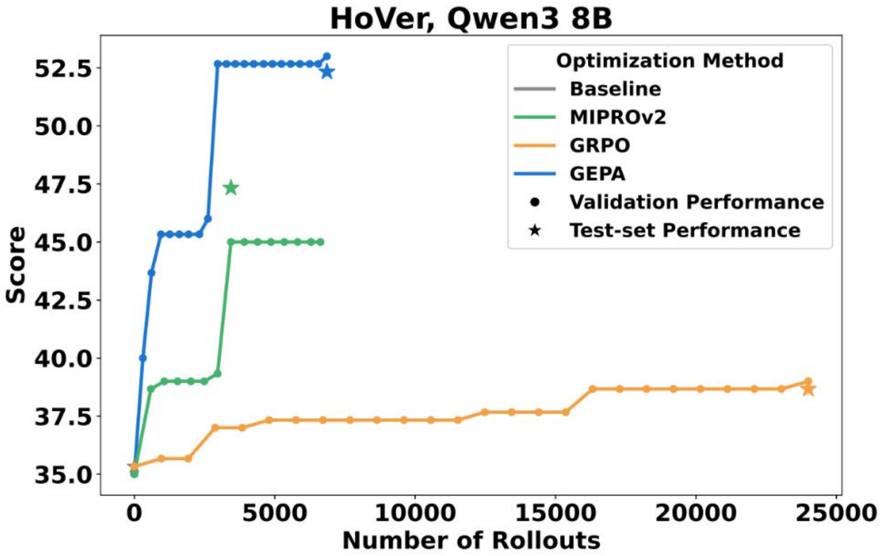
[1]UC Berkeley [2]Stanford University [3]BespokeLabs.ai [4]Notre Dame [5]Databricks [6]MIT

ABSTRACT

Large language models (LLMs) are increasingly adapted to downstream tasks via reinforcement learning (RL) methods like Group Relative Policy Optimization (GRPO), which often require thousands of rollouts to learn new tasks. We argue that the interpretable nature of *language* can often provide a much richer learning medium for LLMs, compared with policy gradients derived from sparse, scalar rewards. To test this, we introduce GEPA (**Ge**netic-**Pa**reto), a prompt optimizer that thoroughly incorporates *natural language reflection* to learn high-level rules from trial and error. Given any AI system containing one or more LLM prompts, GEPA samples system-level trajectories (e.g., reasoning, tool calls, and tool outputs) and reflects on them in natural language to diagnose problems, propose and test prompt updates, and combine complementary lessons from the Pareto frontier of its own attempts. As a result of GEPA's design, it can often turn even just a few rollouts into a large quality gain. Across four tasks, GEPA outperforms GRPO by 10% on average and by up to 20%, while using up to 35x fewer rollouts. GEPA also outperforms the leading prompt optimizer, MIPROv2, by over 10% across two LLMs, and demonstrates promising results as an inference-time search strategy for code optimization.



(a) HotpotQA, Qwen3 8B



(b) HoVer, Qwen3 8B

Seed Prompt for Second-Hop of Multi-Hop QA System

Given the fields `question`, `summary_1`, produce the fields `query`.

GEPA's Optimized Prompt for Second-Hop of Multi-Hop QA System, GPT-4.1 Mini

You will be given two input fields: `question` and `summary_1`. Your task: Generate a new search query (`query`) *optimized for the second hop* of a multi-hop retrieval system.

- The original user question is typically complex and requires information from multiple documents to answer.
- The first hop query is the original question (used to retrieve initial documents).
- Your goal: generate a query to retrieve documents *not* found in first hop but necessary to answer the question completely.

Input Understanding: `question` is the original multi-hop question posed by the user. `summary_1` is a concise summary of information from a document retrieved in the first hop, which partially addresses the question.

Purpose and Context:

- Your generated `query` aims to find the *missing pieces* of information needed to fully answer the question. …
- The query must retrieve relevant documents *NOT* found in first hop … for final answer extraction.

Key Observations and Lessons:

- First-hop documents often cover one entity or aspect.
- Remaining relevant documents often involve connected or higher-level concepts mentioned in `summary_1` but not explicitly asked in the original question. The `query` should target these *missing*, but logically linked, documents.
- Avoid merely paraphrasing the original question or restating known facts from `summary_1`.
- Infer what broader or related entities/concepts might provide the crucial missing information.
- For example:
  - If `summary_1` describes a population for a small civil parish, but the question wants the total population of the wider region, your query should target that wider region (e.g., "Madeira archipelago population in 2011").
  - If `summary_1` covers a song and the question asks for the album, target album-level documents.

How to Build the Query:

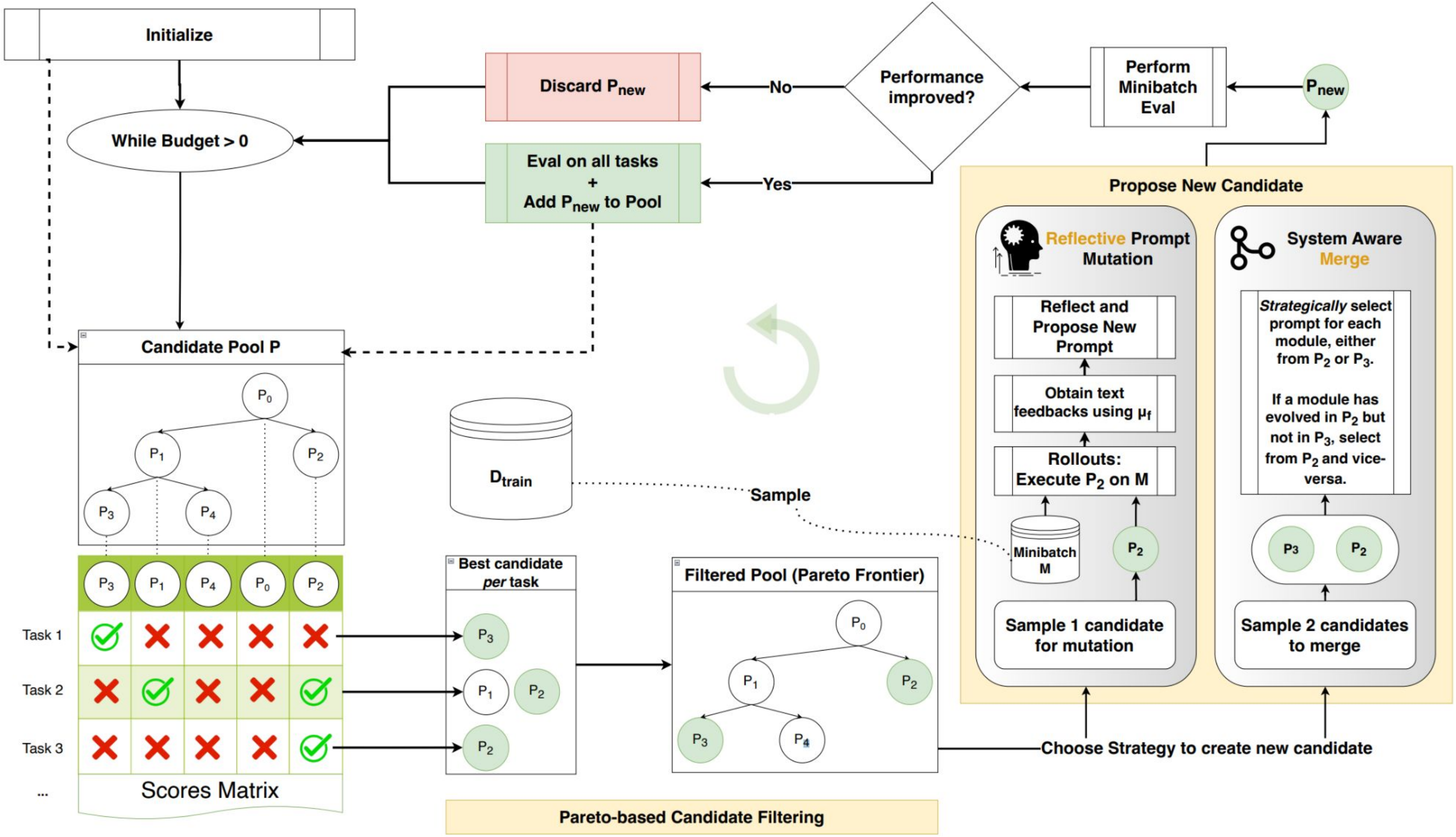
- Identify entities or topics mentioned in `summary_1` that are related but different from first-hop documents.
- Reframe the query to explicitly mention these broader or related entities *connected to the original question*.
- Include relevant key context from the question to maintain specificity, but shift focus to the missing piece.
- The goal is to retrieve documents that link or complement what was retrieved initially.

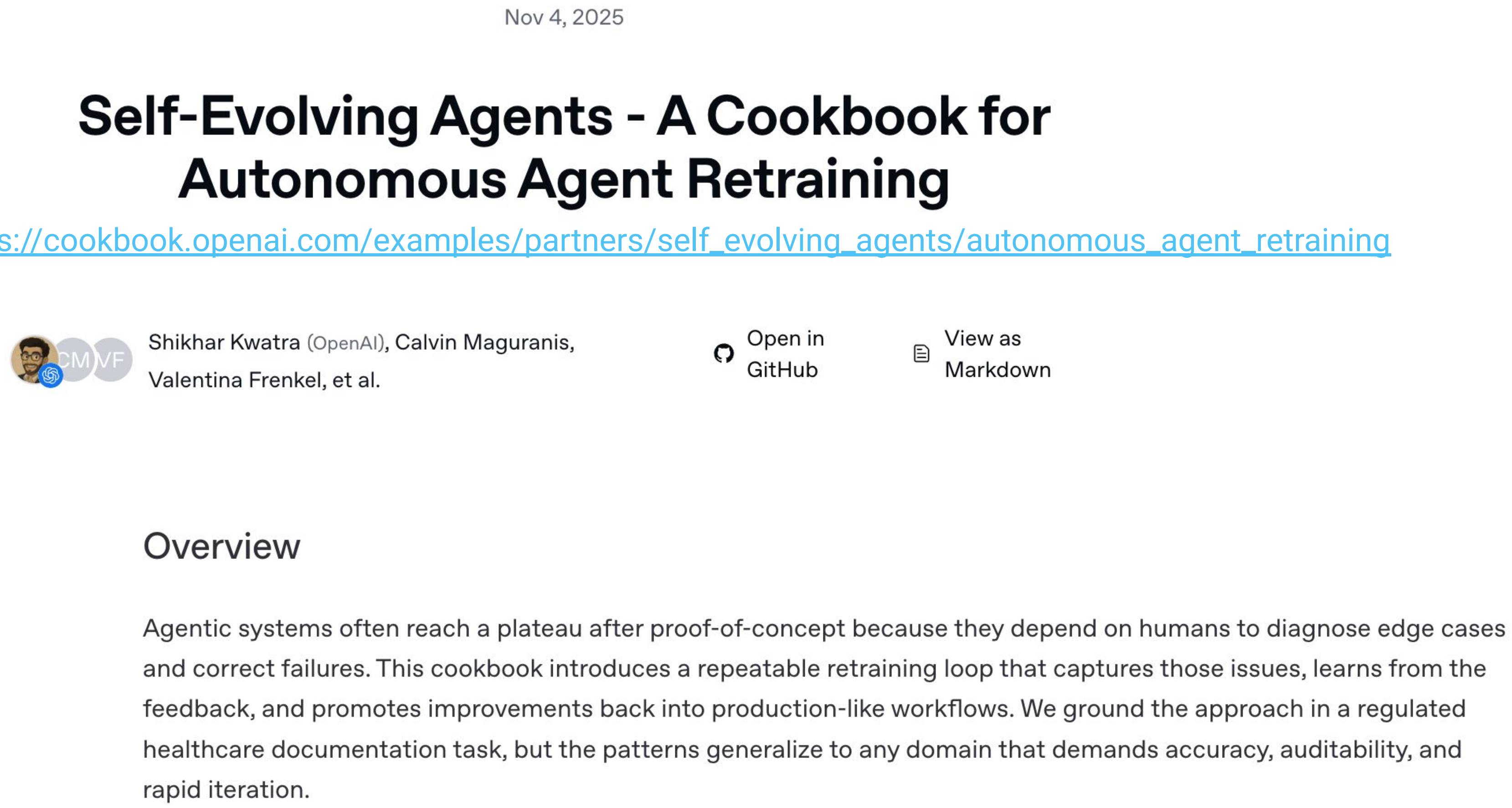
Practical Strategy:

- Read the `summary_1` carefully to spot references to bigger contexts or other entities not covered in the first hop.
- Ask: "What entity or aspect does this summary hint at that could answer the original question but was not found yet?"
- Formulate a precise, focused factual query targeting that entity or concept to retrieve the missing documents.

Output:

- Produce `query` as a clear, concise question or keyword phrase designed for efficient retrieval of second-hop documents.
- Ensure the query relates logically to the original question while targeting the broader or complementary knowledge identified in `summary_1`. … Do not include the original question or simply rephrase it. Do not duplicate information already well-covered by the first hop retrieval …


Initialize
While Budget > 0
Discard P_{new}
No
Performance improved?
Perform Minibatch Eval
P_{new}
Eval on all tasks + Add P_{new} to Pool
Yes
Propose New Candidate
Reflective Prompt Mutation
Reflect and Propose New Prompt
Obtain text feedbacks using μ_f
Rollouts: Execute P_2 on M
Minibatch M
P_2
Sample 1 candidate for mutation
System Aware Merge
Strategically select prompt for each module, either from P_2 or P_3.
If a module has evolved in P_2 but not in P_3, select from P_2 and vice-versa.
P_3
P_2
Sample 2 candidates to merge
Candidate Pool P
P_0
P_1
P_2
P_3
P_4
D_{train}
Sample
P_3 P_1 P_4 P_0 P_2
Task 1
Task 2
Task 3
...
Scores Matrix
Best candidate *per* task
P_3
P_1
P_2
P_2
Filtered Pool (Pareto Frontier)
P_0
P_1
P_2
P_3
P_4
Choose Strategy to create new candidate
Pareto-based Candidate Filtering



Nov 4, 2025

Self-Evolving Agents - A Cookbook for Autonomous Agent Retraining

https://cookbook.openai.com/examples/partners/self_evolving_agents/autonomous_agent_retraining

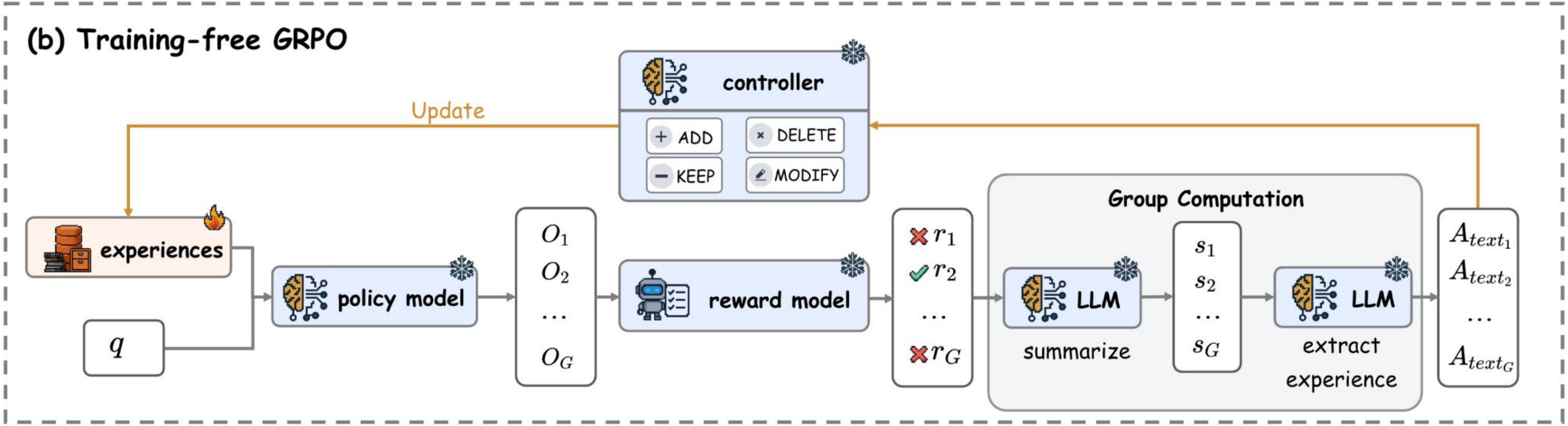
Shikhar Kwatra (OpenAI), Calvin Maguranis, Valentina Frenkel, et al.

Overview

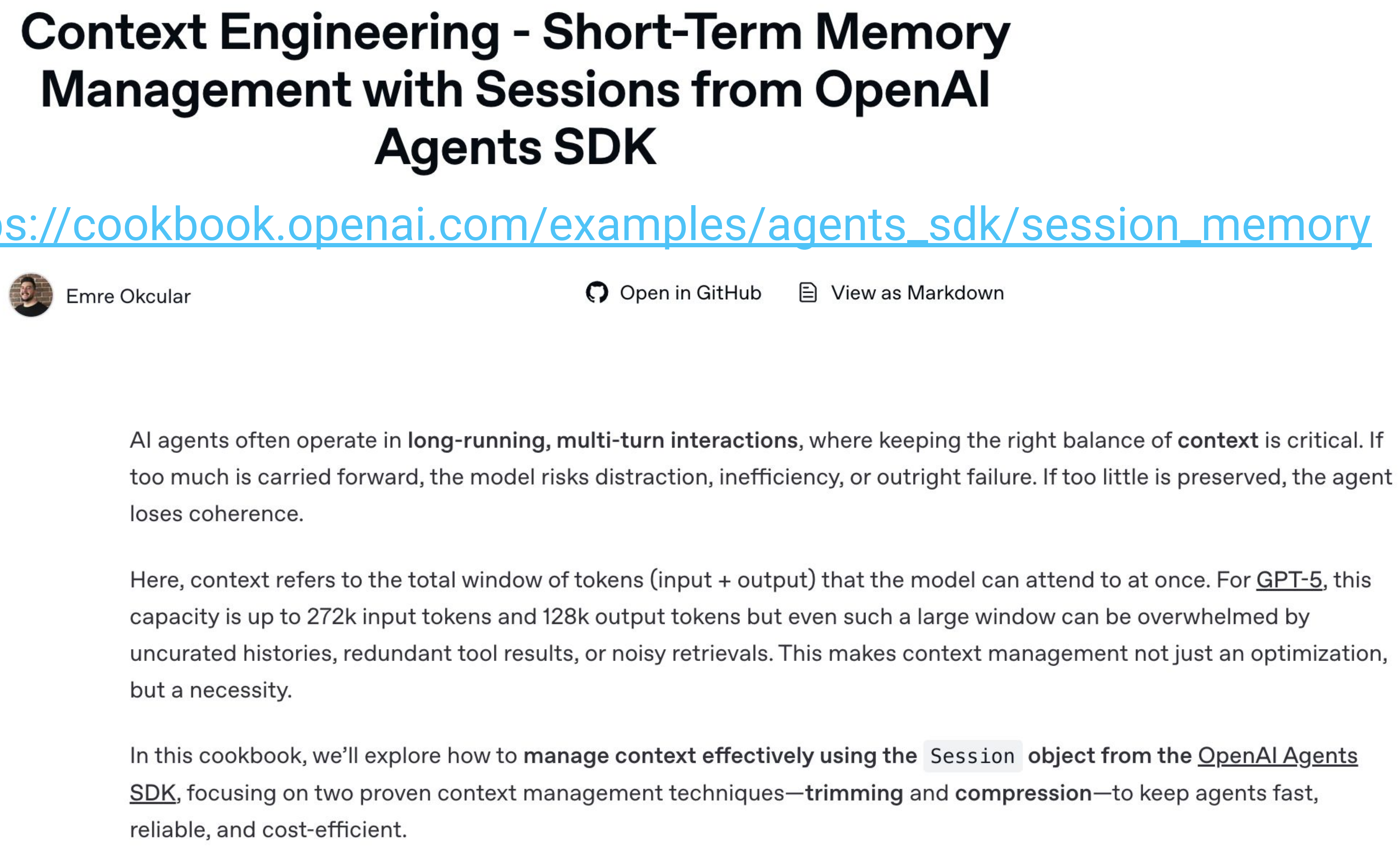
Agentic systems often reach a plateau after proof-of-concept because they depend on humans to diagnose edge cases and correct failures. This cookbook introduces a repeatable retraining loop that captures those issues, learns from the feedback, and promotes improvements back into production-like workflows. We ground the approach in a regulated healthcare documentation task, but the patterns generalize to any domain that demands accuracy, auditability, and rapid iteration.

Agentic memory

- Recursive Language Models https://alexzhang13.github.io/blog/2025/rlm/
- Anthropic Effective context engineering for AI agents https://www.anthropic.com/engineering/effective-context-engineering-for-ai-agents
- AgentFold: Long-Horizon Web Agents with Proactive Context Management https://arxiv.org/abs/2510.24699
- ReasoningBank: Scaling Agent Self-Evolving with Reasoning Memory https://arxiv.org/abs/2509.25140
- Memento: Fine-tuning LLM Agents without Fine-tuning LLMs https://arxiv.org/abs/2508.16153
- Agentic Context Engineering: Evolving Contexts for Self-Improving Language Models https://arxiv.org/abs/2510.04618
- A-MEM: Agentic Memory for LLM Agents https://arxiv.org/abs/2502.12110
- Training-Free Group Relative Policy Optimization https://arxiv.org/abs/2510.08191



Training-Free Group Relative Policy Optimization



Context Engineering - Short-Term Memory Management with Sessions from OpenAI Agents SDK

https://cookbook.openai.com/examples/agents_sdk/session_memory

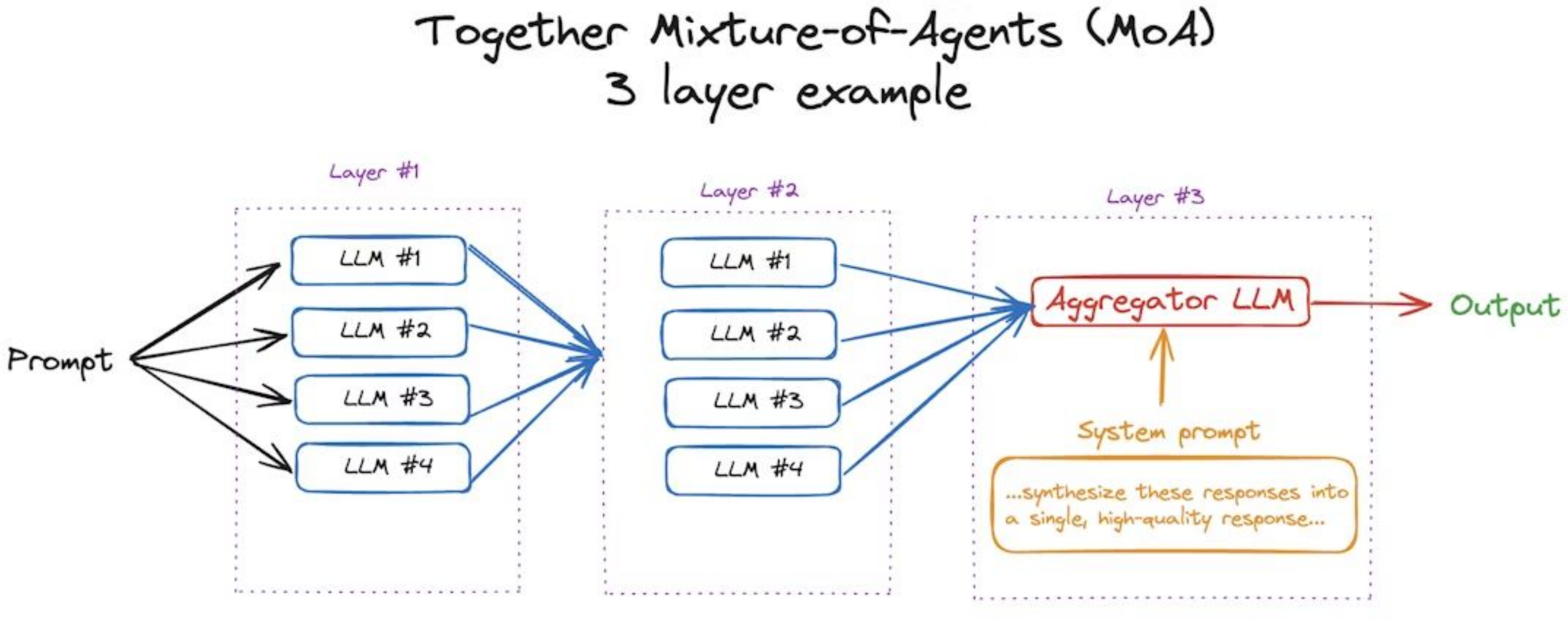
Emre Okcular

AI agents often operate in **long-running, multi-turn interactions**, where keeping the right balance of **context** is critical. If too much is carried forward, the model risks distraction, inefficiency, or outright failure. If too little is preserved, the agent loses coherence.

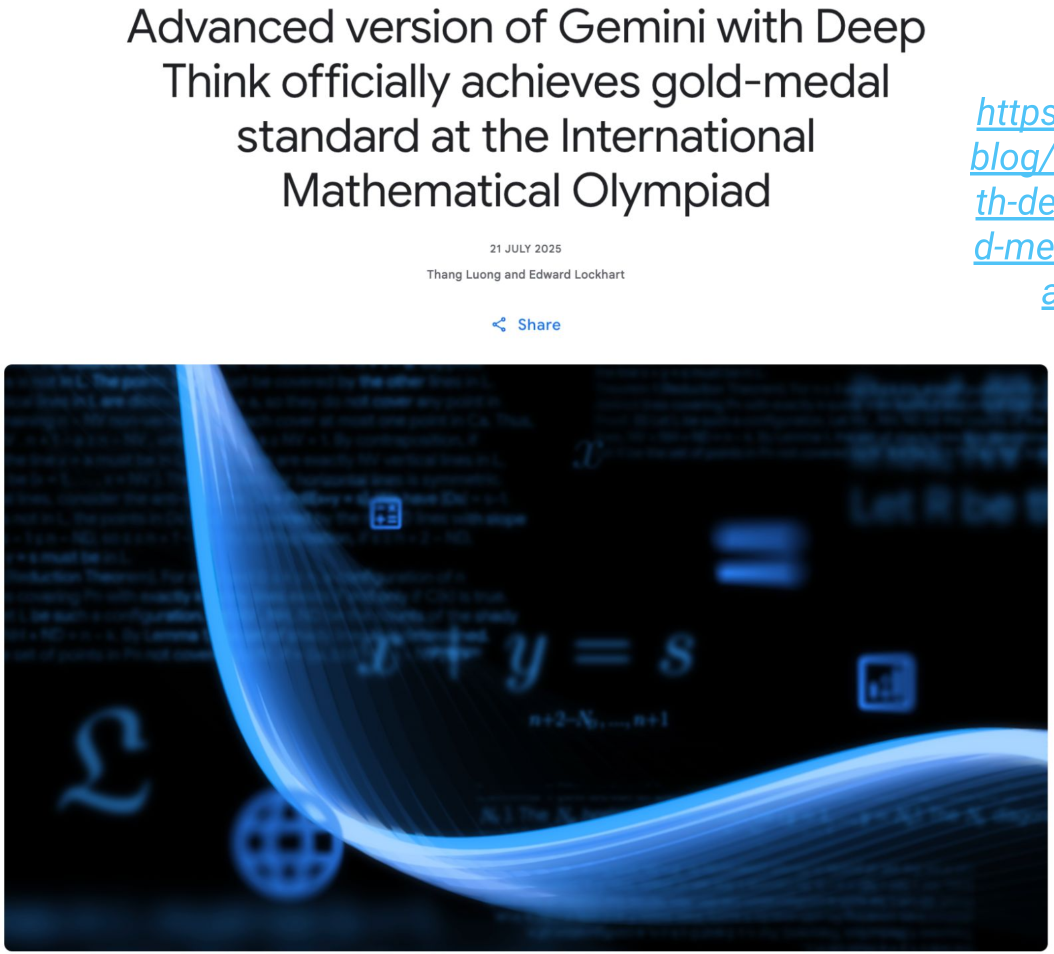
Here, context refers to the total window of tokens (input + output) that the model can attend to at once. For GPT-5, this capacity is up to 272k input tokens and 128k output tokens but even such a large window can be overwhelmed by uncurated histories, redundant tool results, or noisy retrievals. This makes context management not just an optimization, but a necessity.

In this cookbook, we'll explore how to **manage context effectively using the** `Session` **object from the** OpenAI Agents SDK, focusing on two proven context management techniques—**trimming** and **compression**—to keep agents fast, reliable, and cost-efficient.

Mixture-of-Agents



https://docs.together.ai/docs/mixture-of-agents#advanced-moa-example

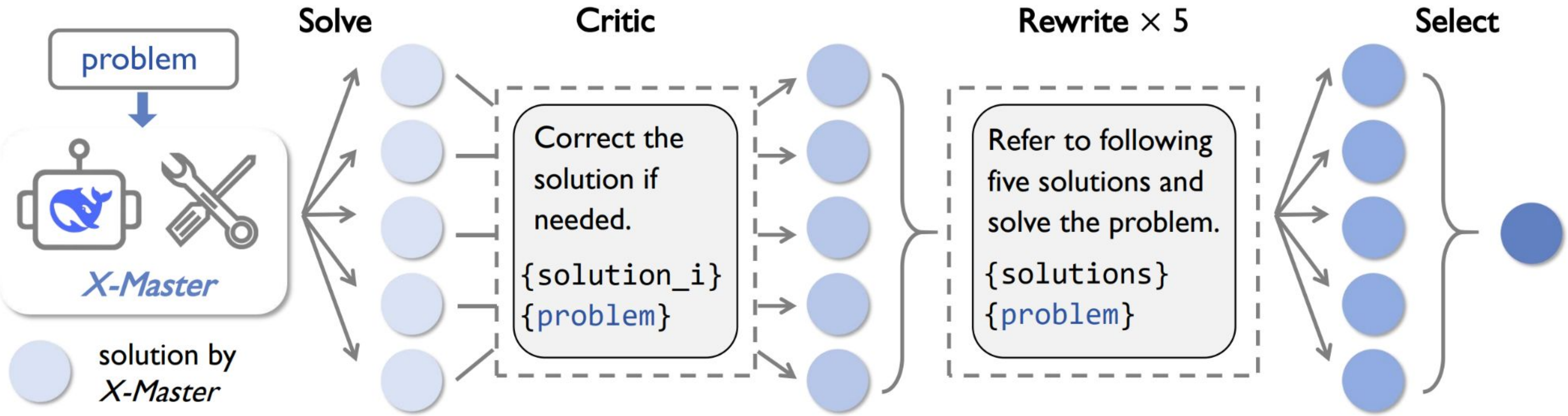


Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad

21 JULY 2025

Thang Luong and Edward Lockhart

https://deepmind.google/discover/blog/advanced-version-of-gemini-with-deep-think-officially-achieves-gold-medal-standard-at-the-international-mathematical-olympiad/



[SciMaster: Towards General-Purpose Scientific AI Agents, Part I. X-Master as Foundation: Can We Lead on Humanity's Last Exam?]

Debate or Vote: Which Yields Better Decisions in Multi-Agent Large Language Models?

Hyeong Kyu Choi **Xiaojin Zhu** **Sharon Li***
Department of Computer Sciences, University of Wisconsin-Madison
`{froilanchoi, jerryzhu, sharonli}@cs.wisc.edu`

Majority Voting alone accounts for most of the performance gains typically attributed to MAD

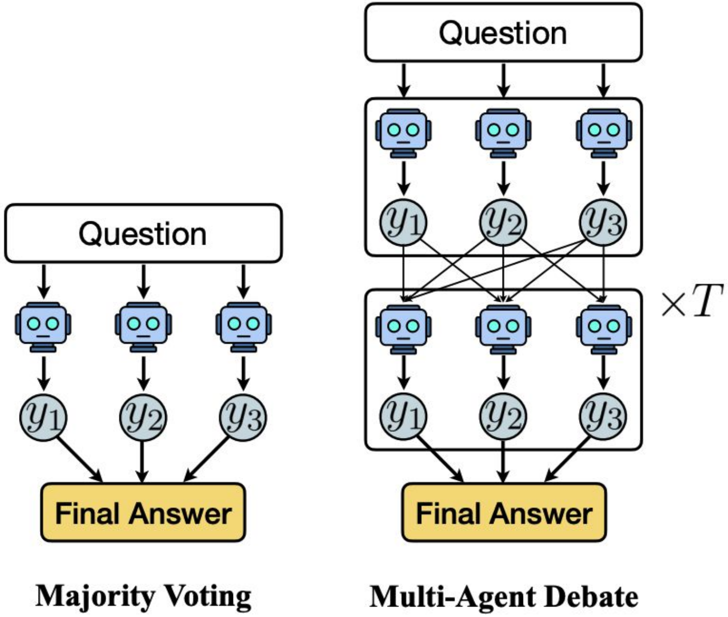


Figure 1: Majority Voting vs. MAD overview.

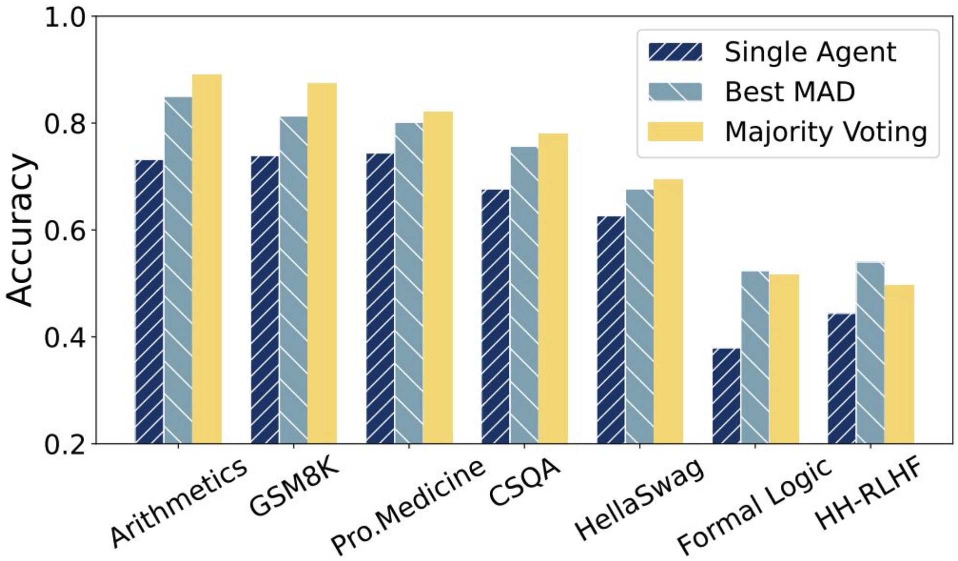


Figure 2: Majority Voting is the main contributor to MAD.

Thank you!