

# Agents

**CS 6804: Frontier AI Systems**

*Spring 2026*

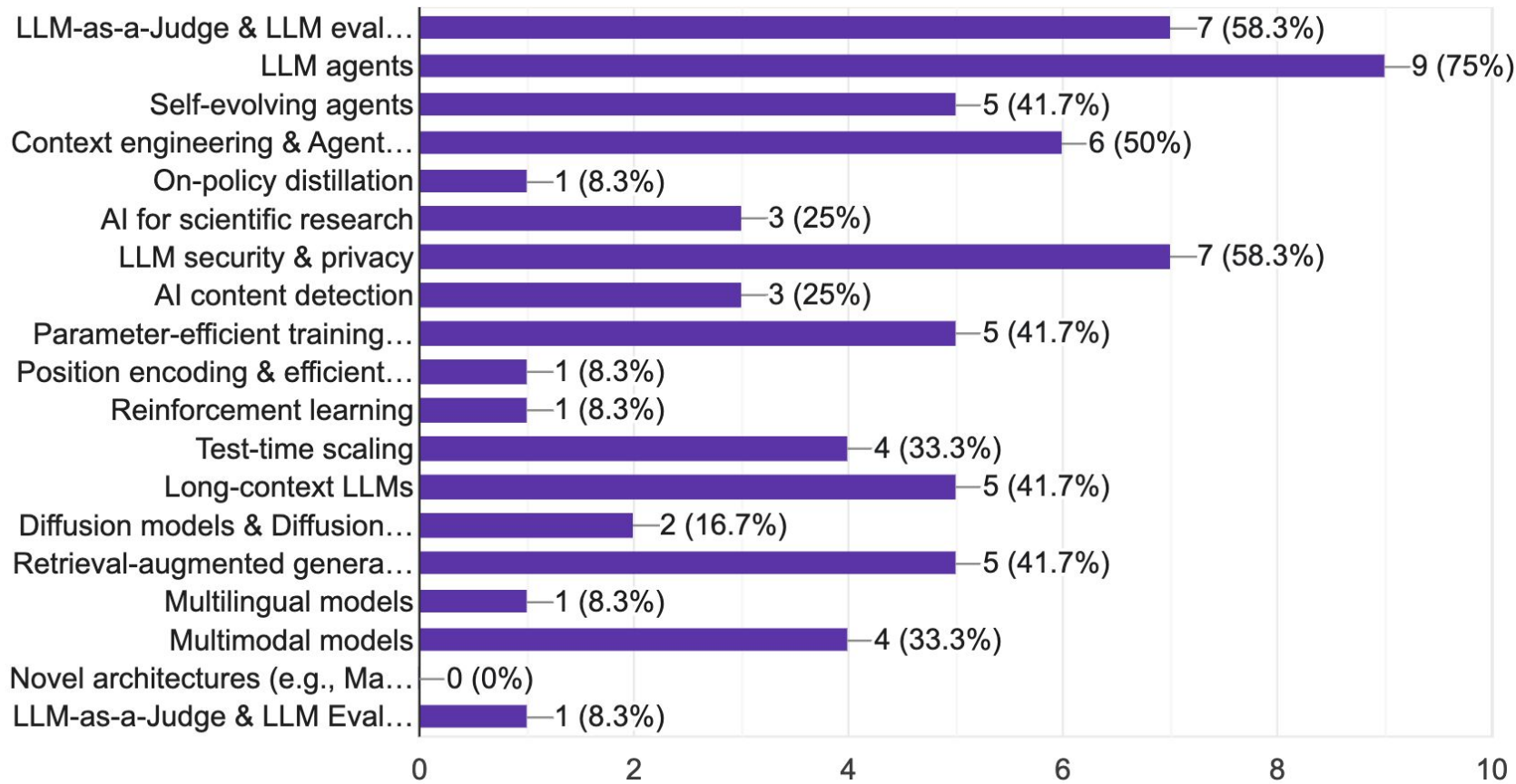
<https://tuvllms.github.io/ai-seminar-spring-2026/>

**Tu Vu**



# Logistics

- **In-class participation** (starting next week)



# First student presentation

- **LLM-as-a-Judge & LLM evaluation**  
Kiymet & Enoch

# Student presentations

## Role-Playing Paper-Reading Seminars

Alec Jacobson and Colin Raffel

March 17th, 2021

[colinraffel.com/blog](https://colinraffel.com/blog)

The paper-reading seminar is a common format for advanced topics courses in Computer Science graduate programs. Over the span of the course, students read a series of academic papers published on a particular topic (for example, [Deep Learning with Limited Labeled Data](#)). For cutting-edge topics close to the frontier of research, standard texts may not exist and academic papers are the primary available references. Some niche areas may not be mature enough to develop a lecture-based course curriculum around. Complementary to course topic-specific learning objectives, paper-reading seminars aim to achieve learning outcomes related to the soft skills of technical reading, public presentation, and critical discussion.

<https://colinraffel.com/blog/role-playing-seminar.html>



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



Andrew Ng

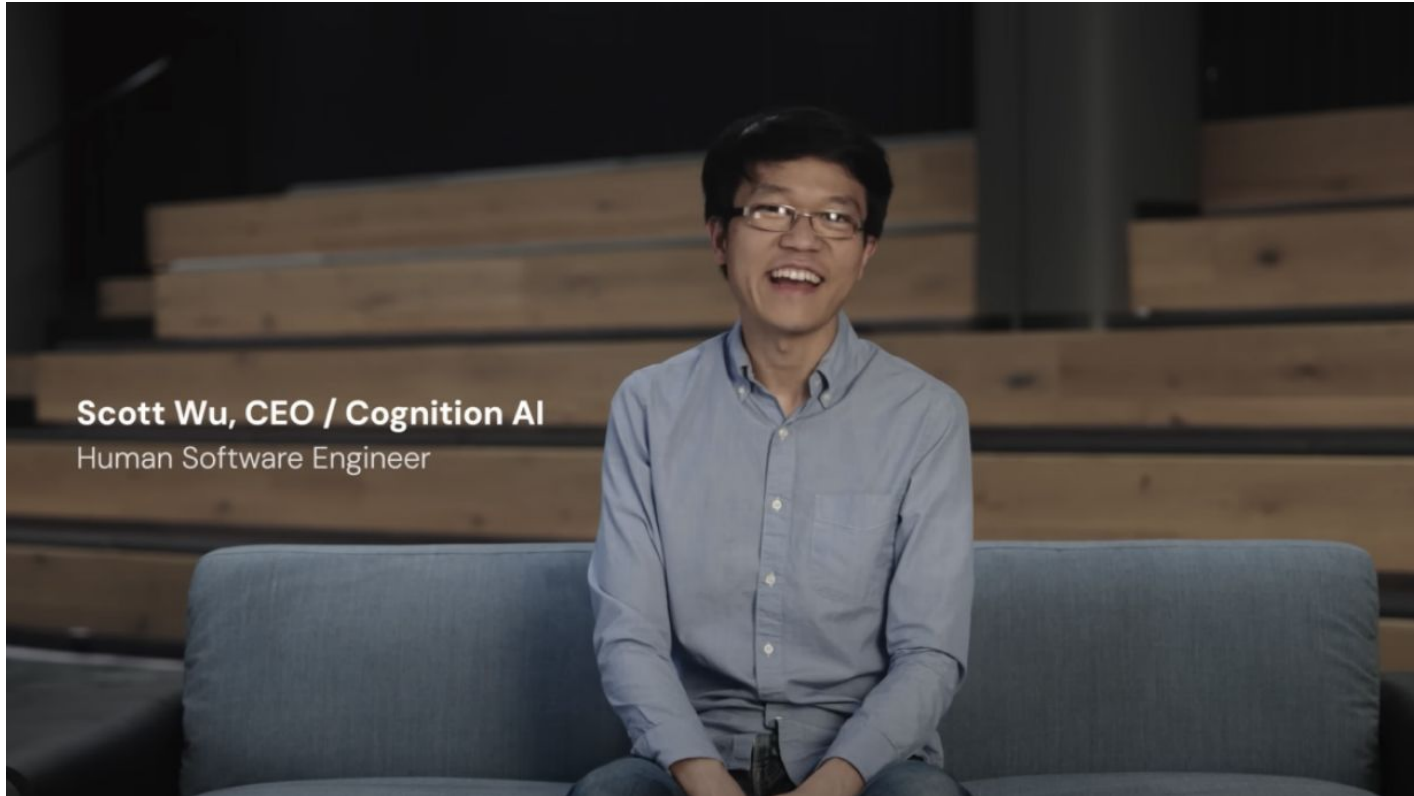
I would say  
Agentic AI



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=fjHtjT7G01c>

# 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.



START



FINISH

## AGENTIC WORKFLOW

Write an essay outline on topic X

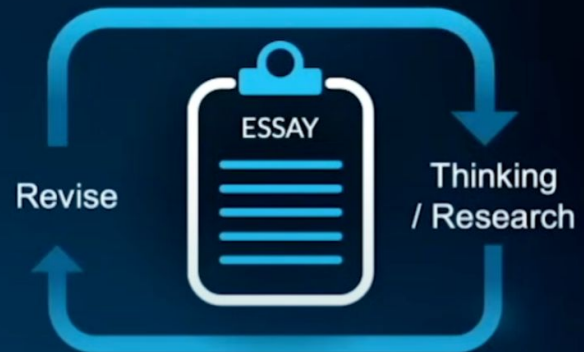
Do you need any web research?

Write a first draft.

Consider what parts need revision or more research.

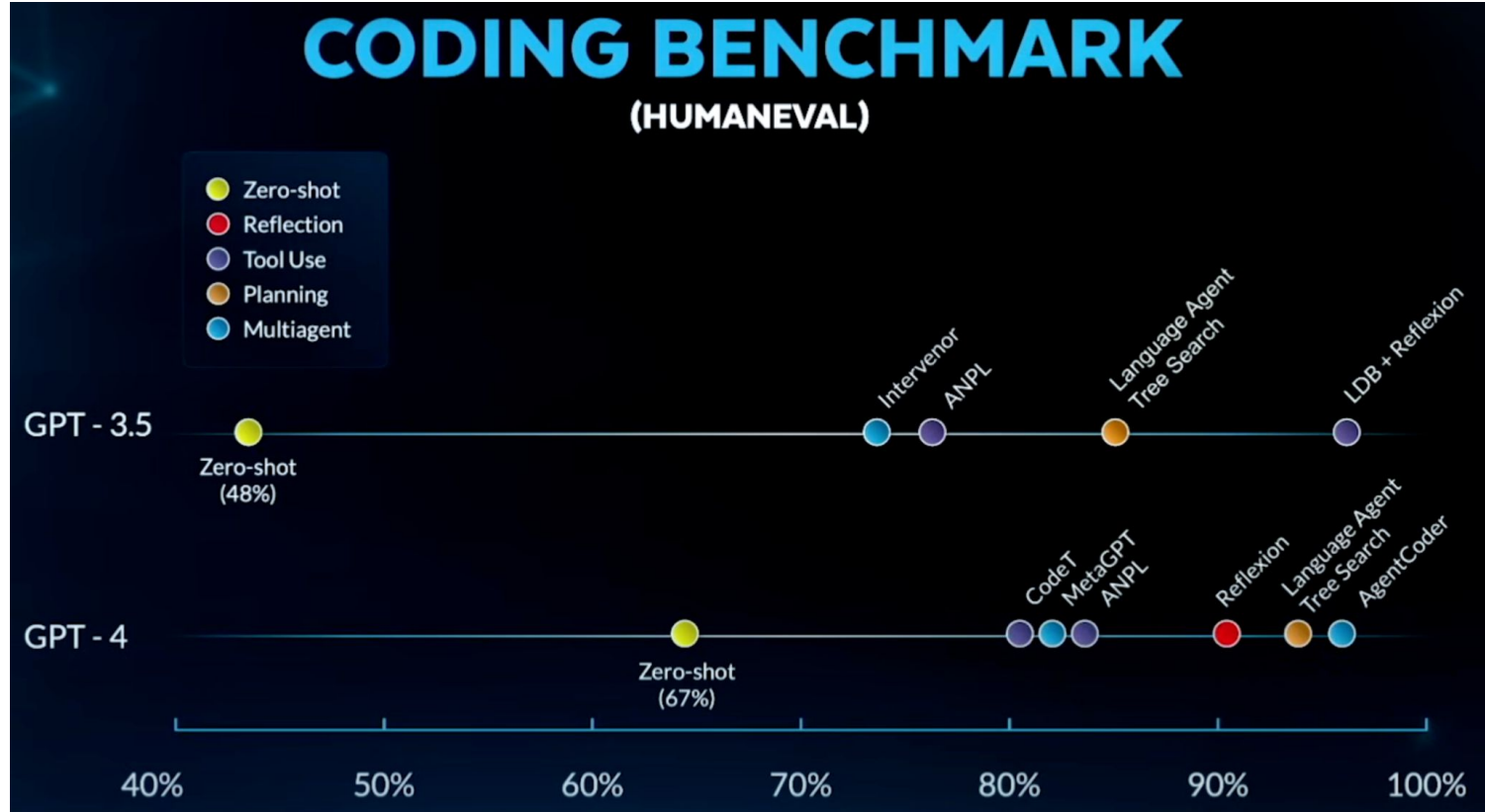
Revise your draft.

...



["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](#)

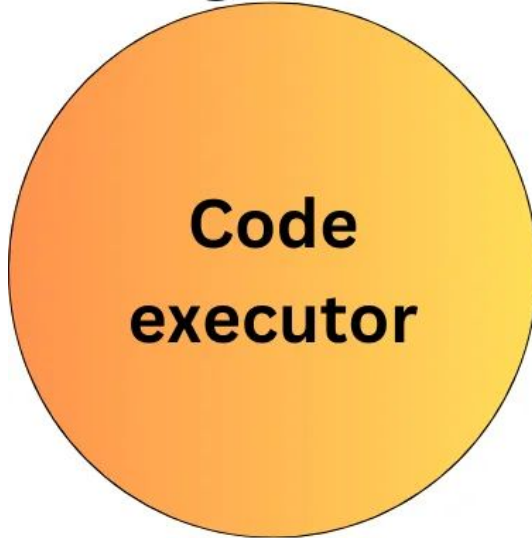
# Design patterns for building agents

- **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.

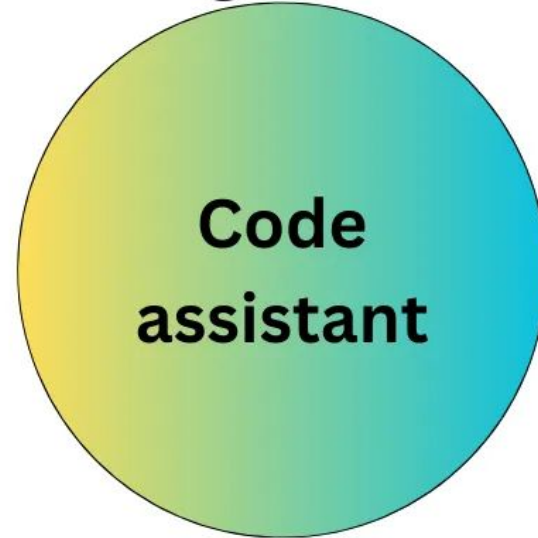
<https://www.deeplearning.ai/the-batch/how-agents-can-improve-llm-performance/>

# 2-agents conversation/collaboration

**Agent 1**



**Agent 2**



# 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.*

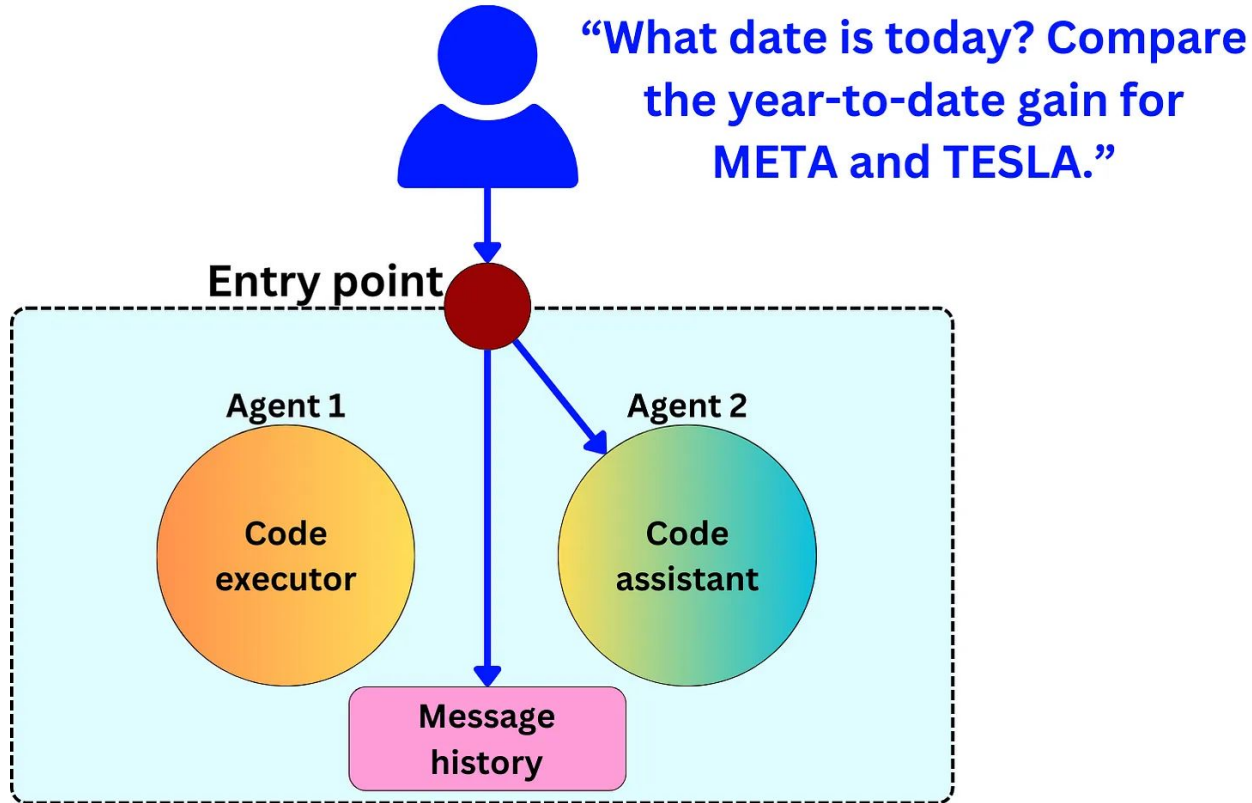
*Solve the task step by step if you need to. If a plan is not provided, explain your plan first. Be clear which step uses code, and which step uses your language skill.*

...

*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.*

## 2-agents conversation/collaboration (cont'd)



# 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.*

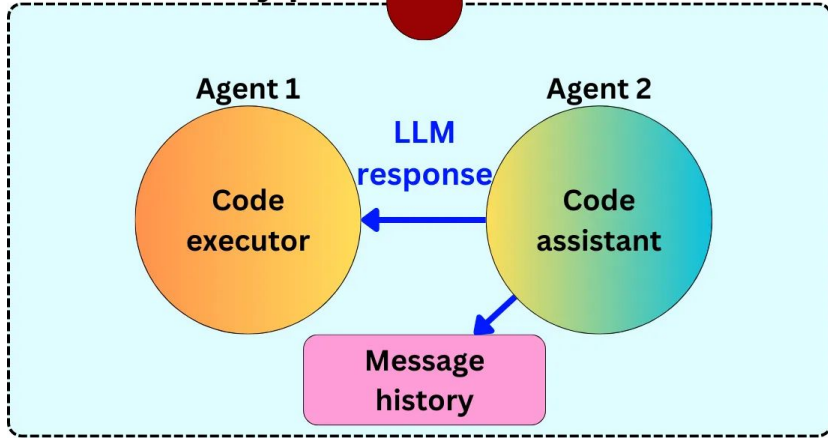
```
```python
from datetime import datetime

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

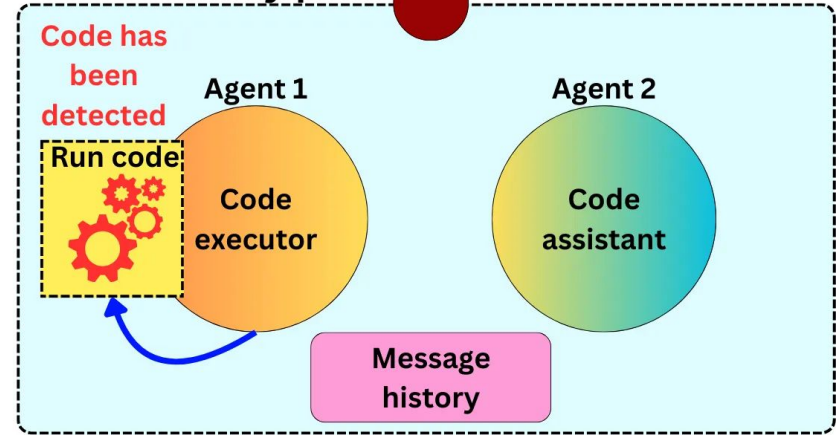
# 2-agents conversation/collaboration (cont'd)



Entry point



Entry point

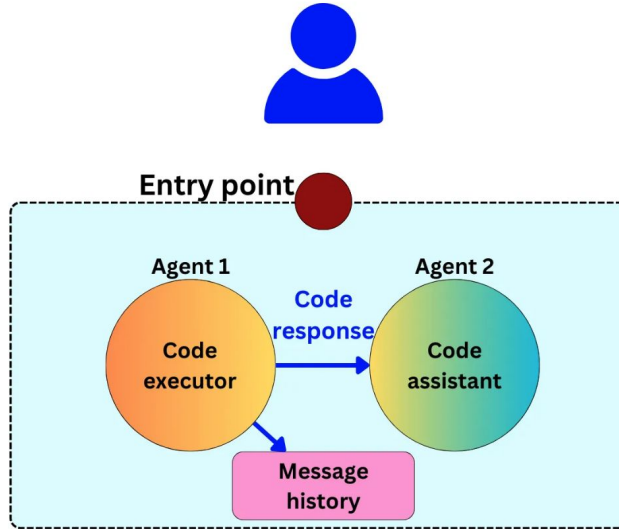


# 2-agents conversation/collaboration (cont'd)

user\_proxy (to assistant):

*exitcode: 0 (execution succeeded)*

*Code output: 2024-09-17*



# 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', 'TESLA']

# 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]

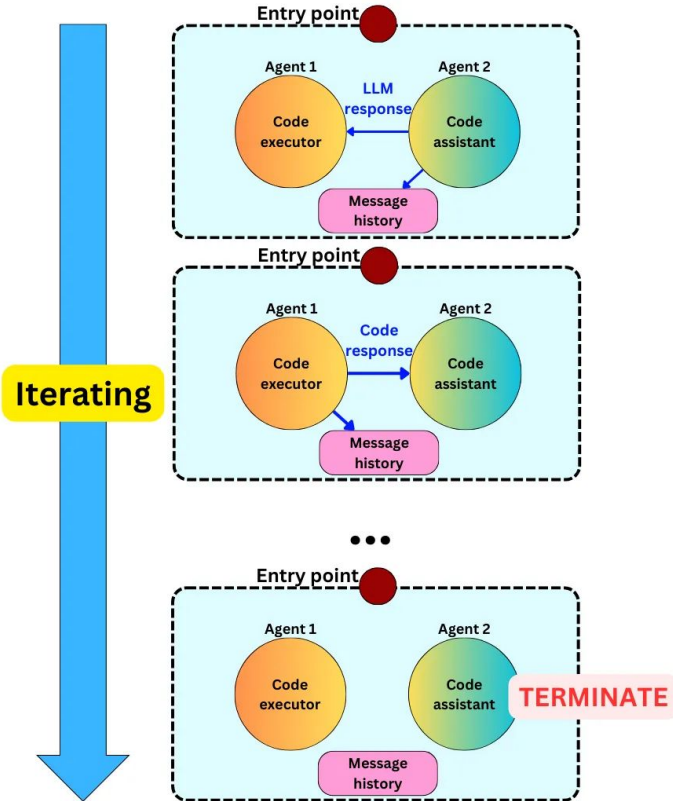
    start_of_year_prices[stock] = stock_data.history(start='2024-01-
01', end='2024-01-02')['Close'].iloc[0]

# 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.*

# 2-agents conversation/collaboration (cont'd)



*The execution was successful, and here are the results:*

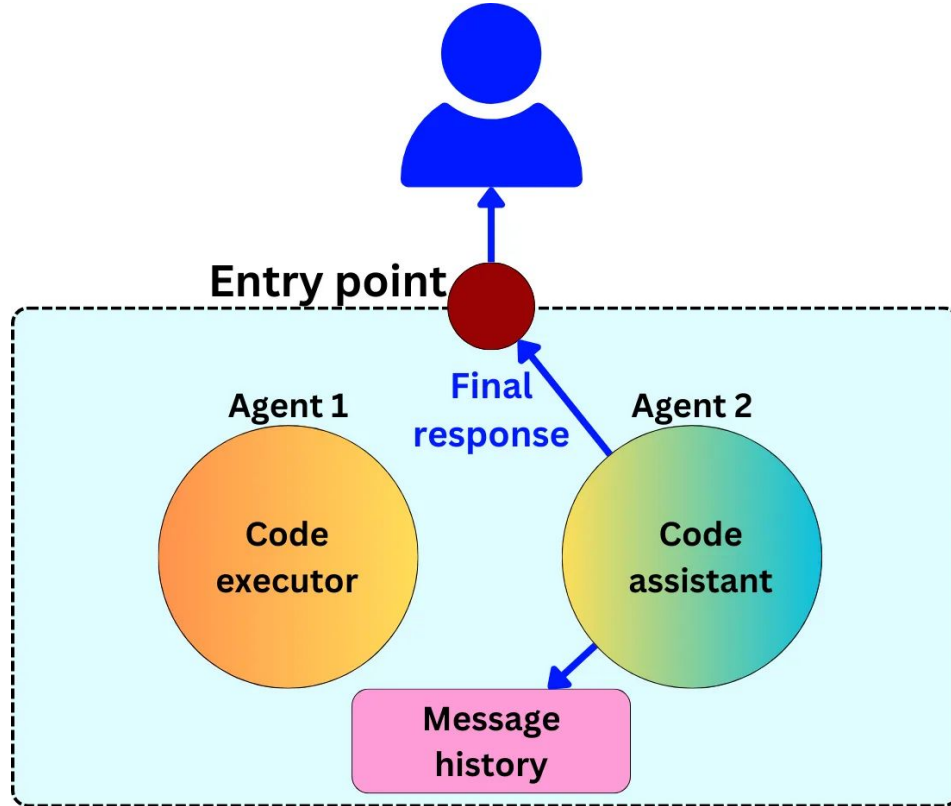
- **\*\*Current Prices:\*\***
  - **META: \$536.32**
  - **TESLA: \$227.87**
- **\*\*Start of Year Prices:\*\***
  - **META: \$345.25**
  - **TESLA: \$248.42**
- **\*\*Year-to-Date Gains:\*\***
  - **META: \*\*55.34%\*\***
  - **TESLA: \*\*-8.27%\*\***

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

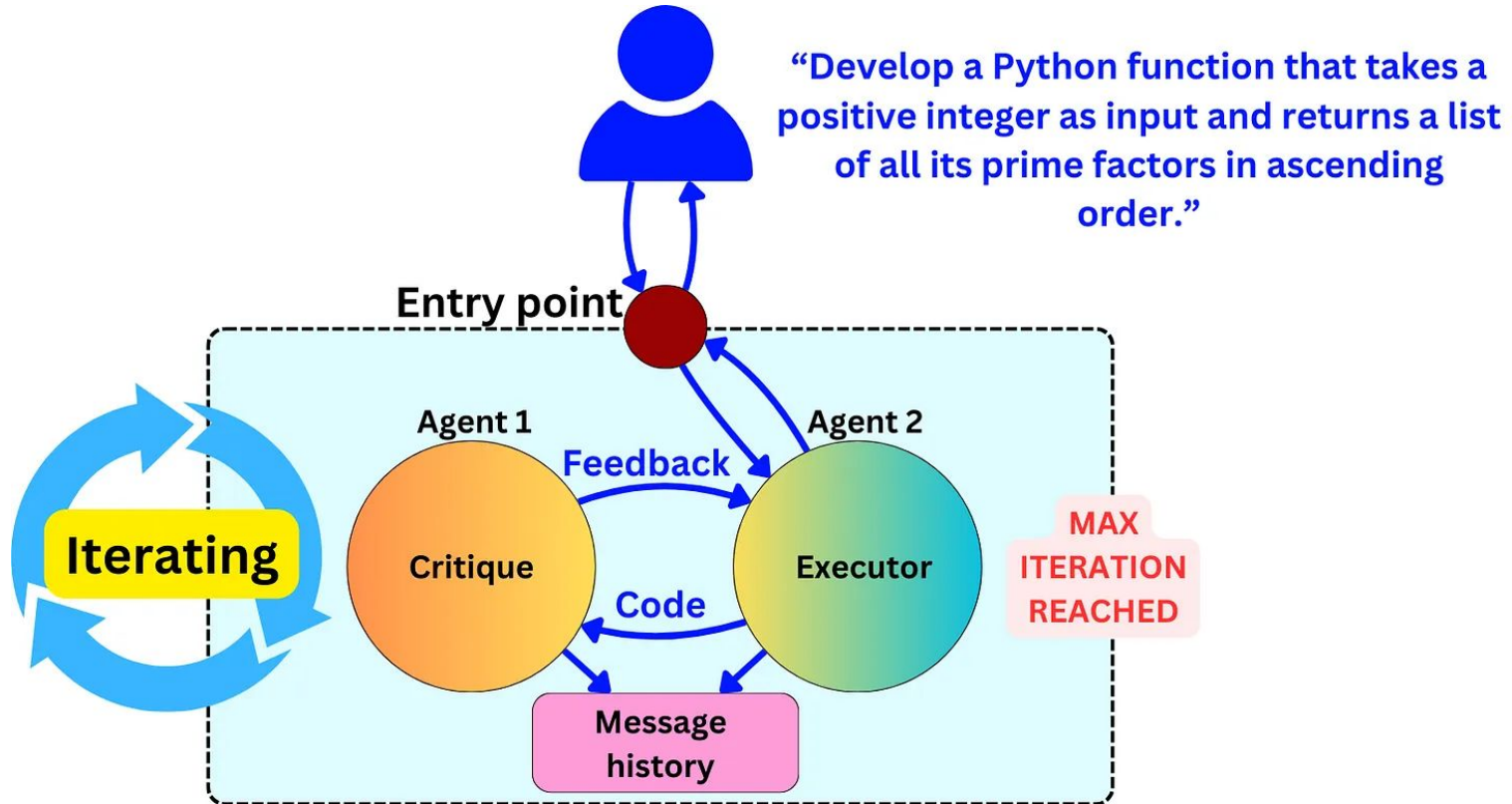
*If you need any further analysis or information, please let me know.*

**TERMINATE**

# 2-agents conversation/collaboration (cont'd)

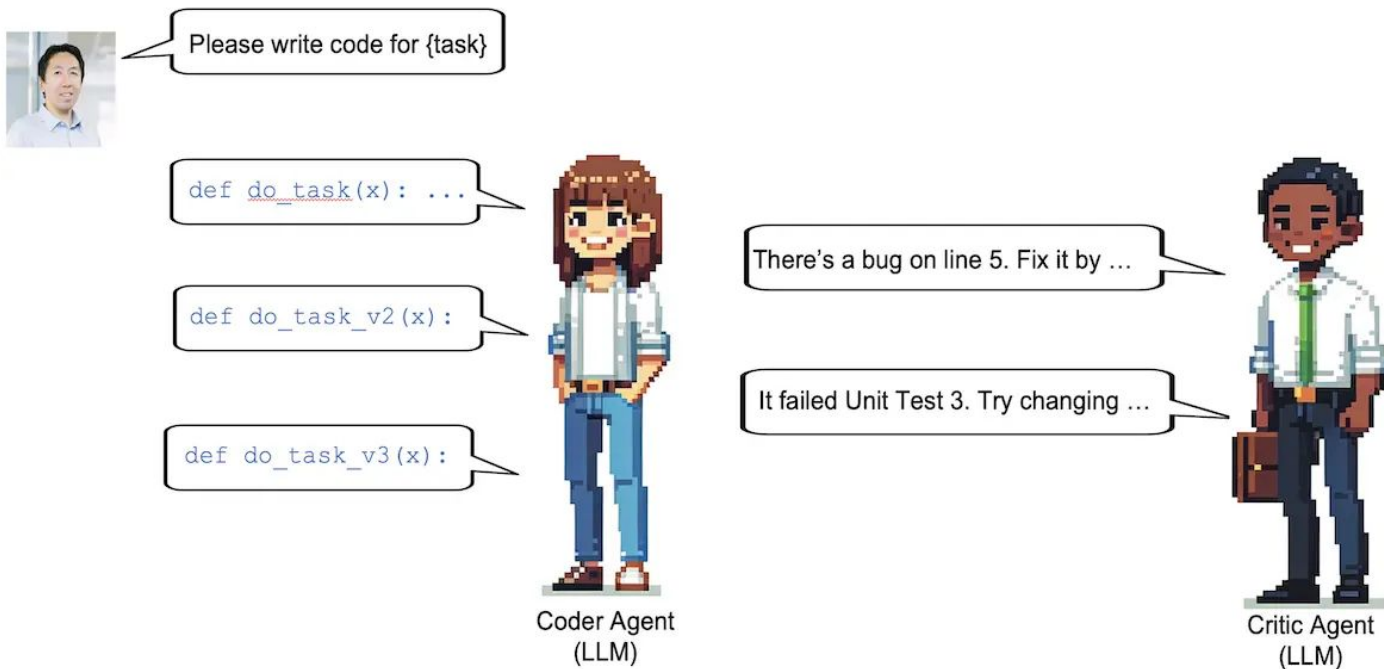


# Reflection



# Reflection (cont'd)

## Agentic Design Patterns: Reflection



# Reflection (cont'd)

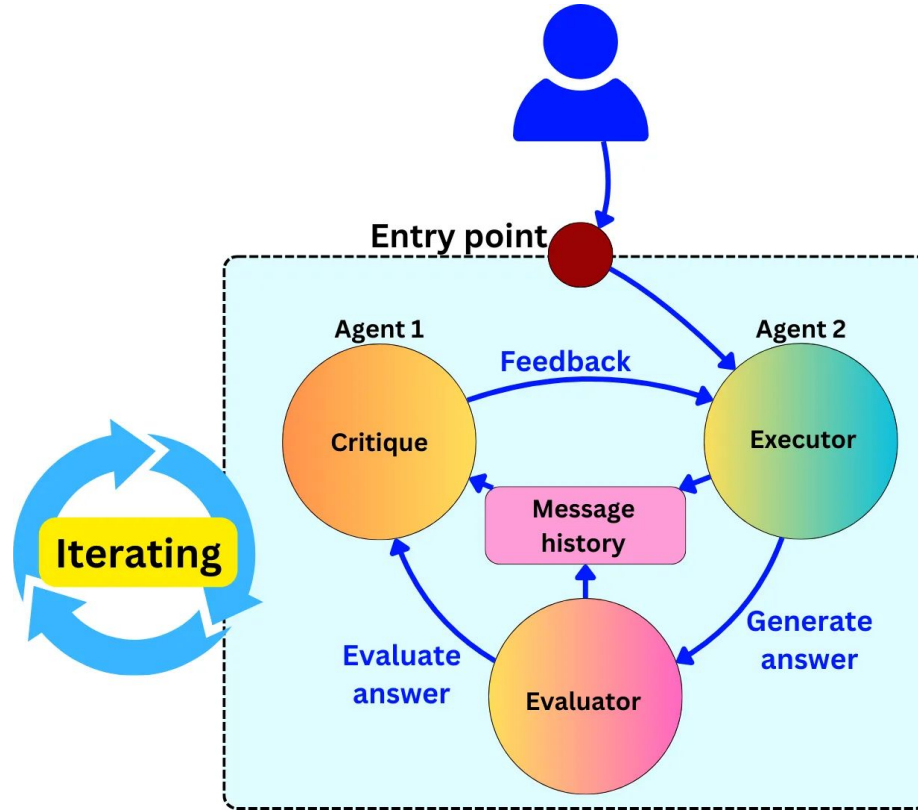
***"Your task is to write Python code that solves the problem presented to you. Ensure that the code is correct, efficient, well-structured, and handles edge cases. Comment the code where necessary and ensure it's readable and maintainable.***

***If the user provides critique, respond with a revised version of your previous attempts."***

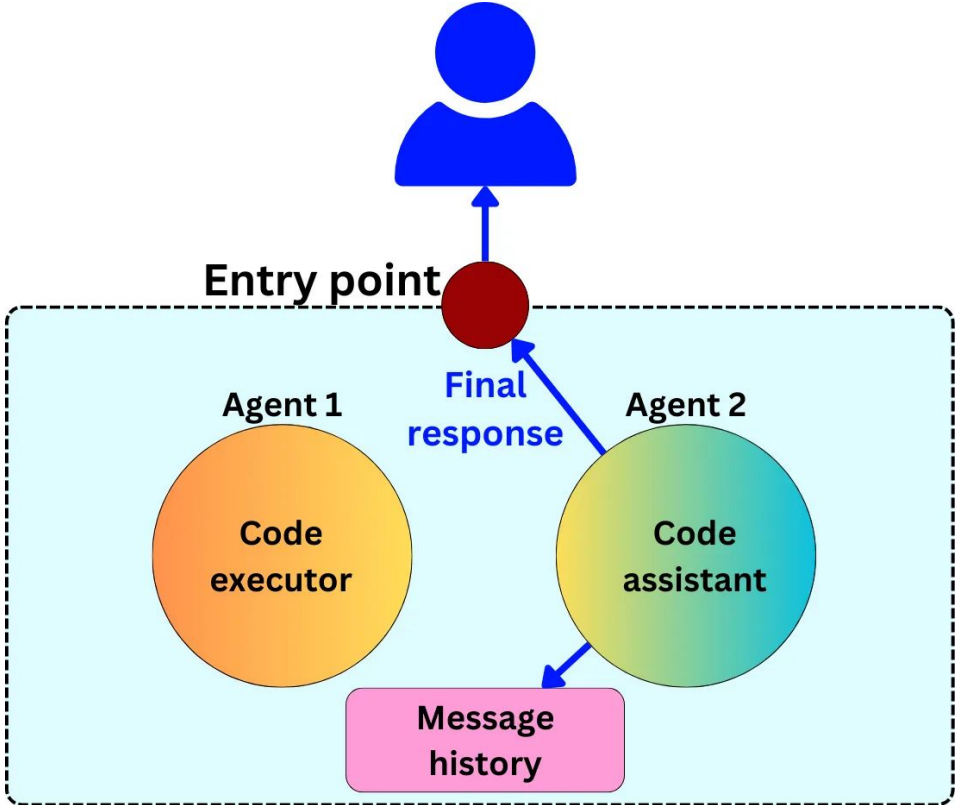
And for the critique

***"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."***

# Reflection (cont'd)



# Tool use



# 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:*

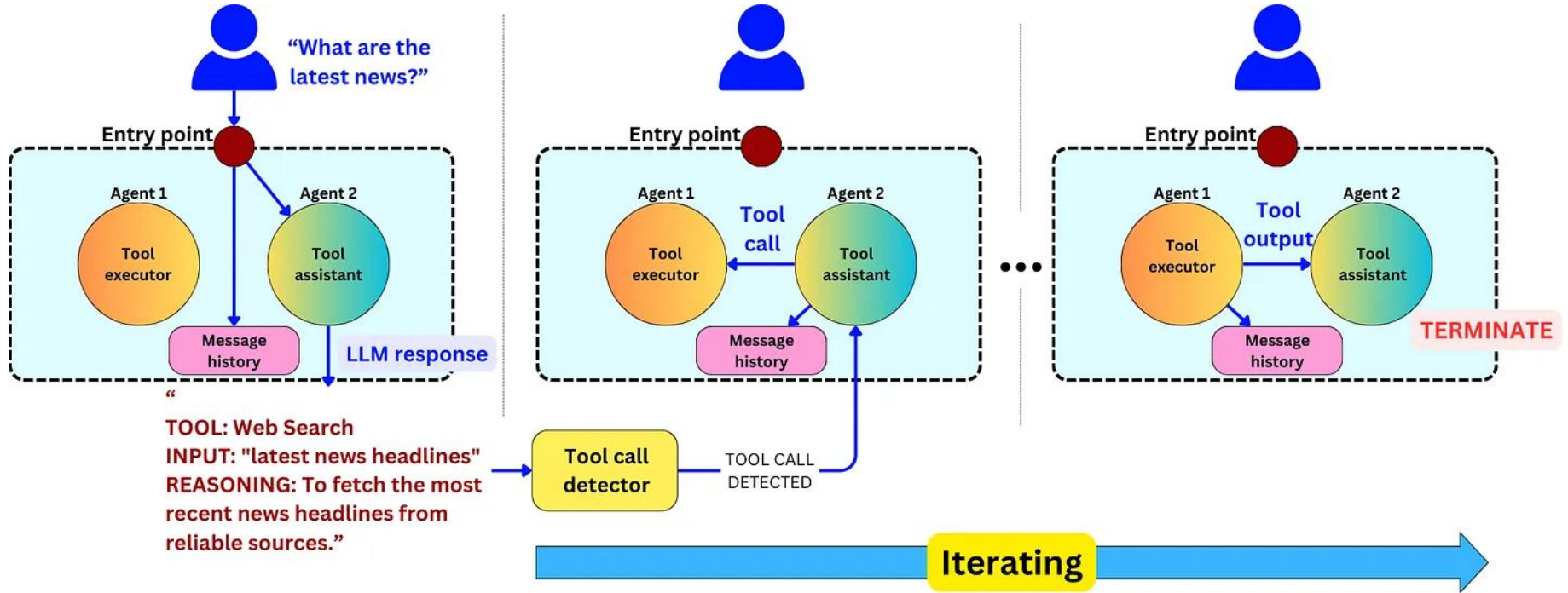
- *TOOL: [Tool name]*
- *INPUT: [Tool input]*
- *REASONING: [Brief explanation]*

*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.*

# Tool use



# When we have many tools

## Google products

Filter by:

Featured



Android



Chrome



Gemini



Gmail



Google Calendar



Google Docs



Google Maps



Google Photos



Google Play



Pixel



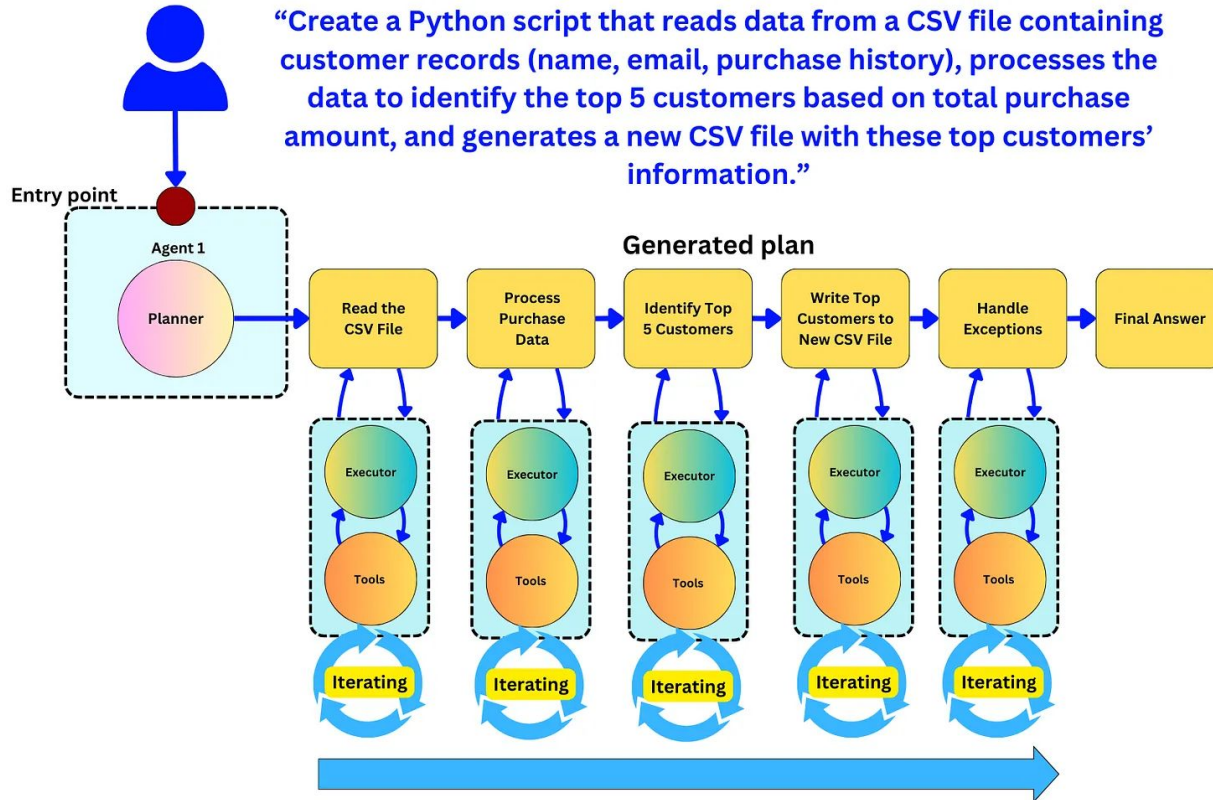
Search



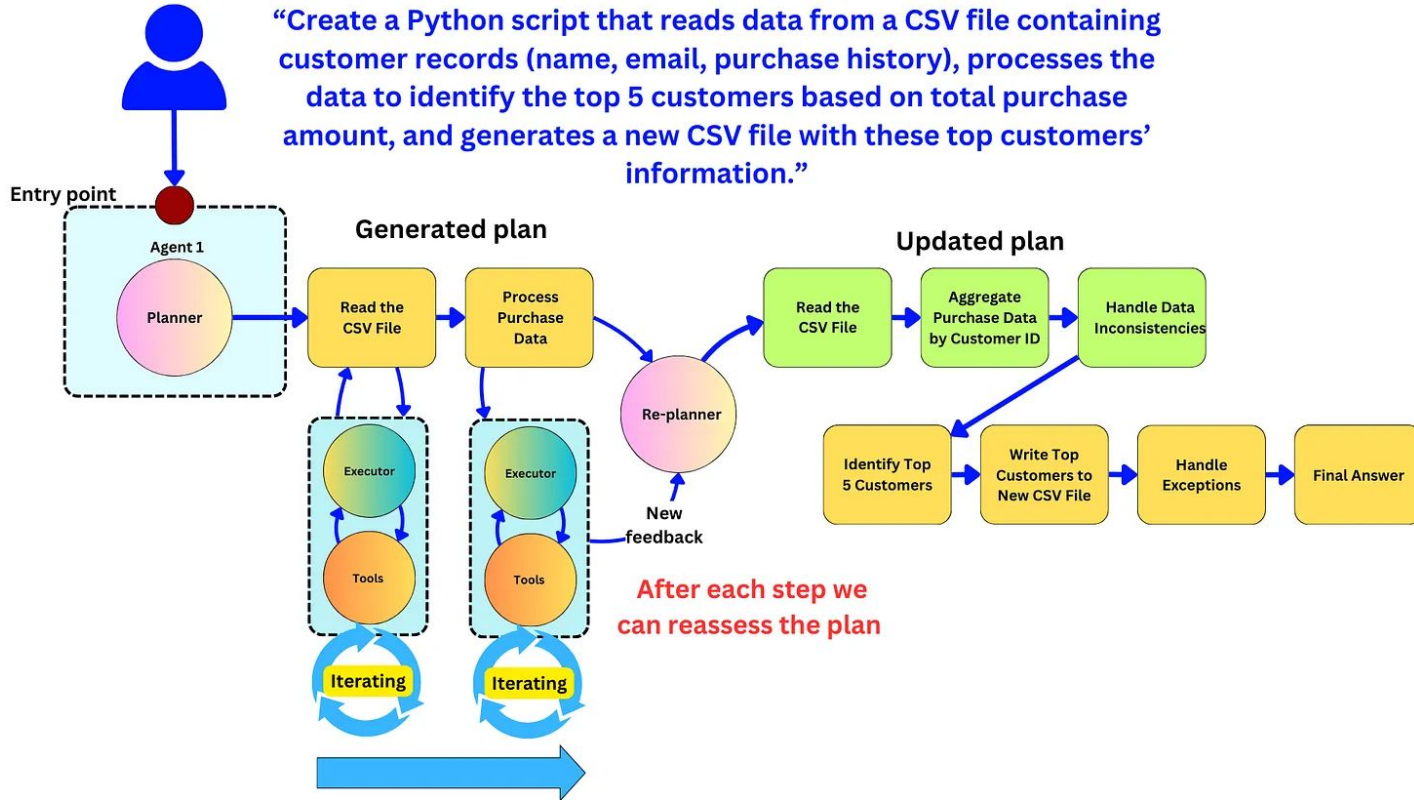
YouTube



# Planning



# Planning (cont'd)



# Planning (cont'd)

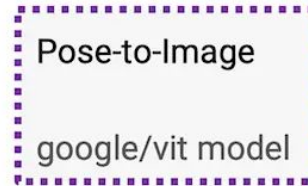
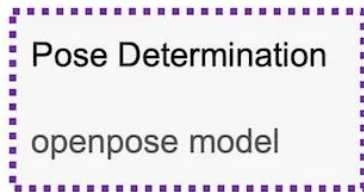
## Agentic Design Patterns: Planning



image.jpg



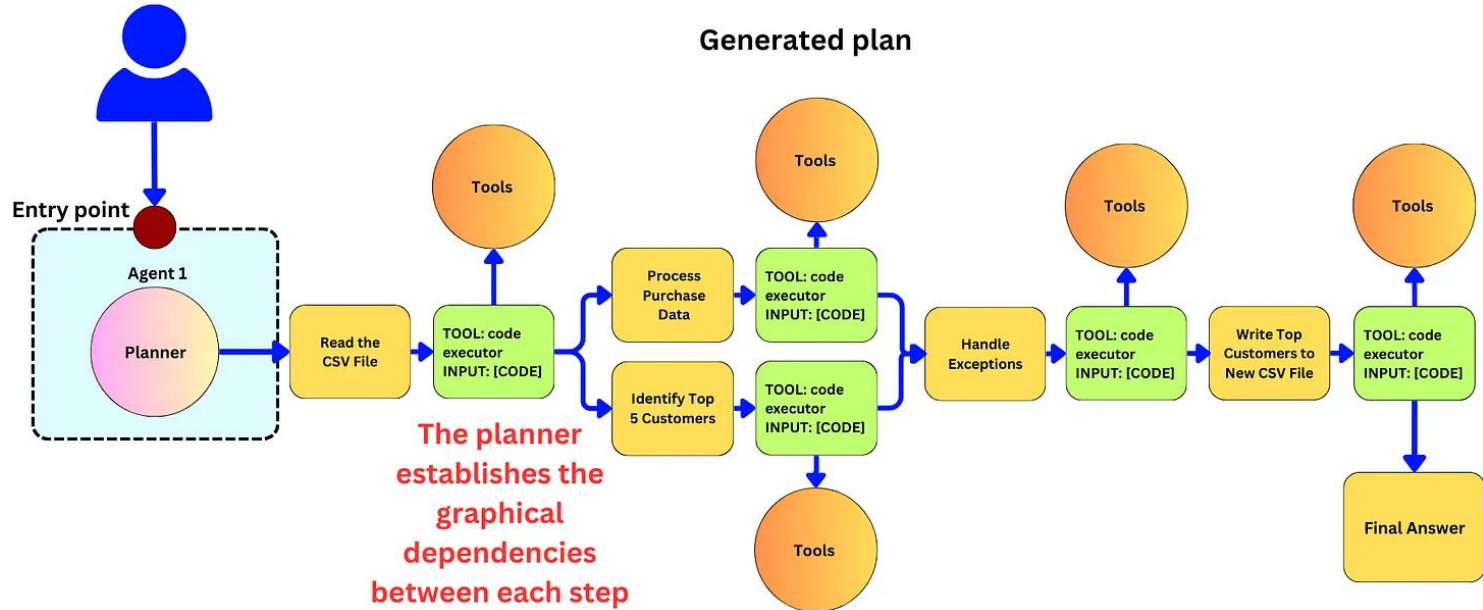
final.jpg



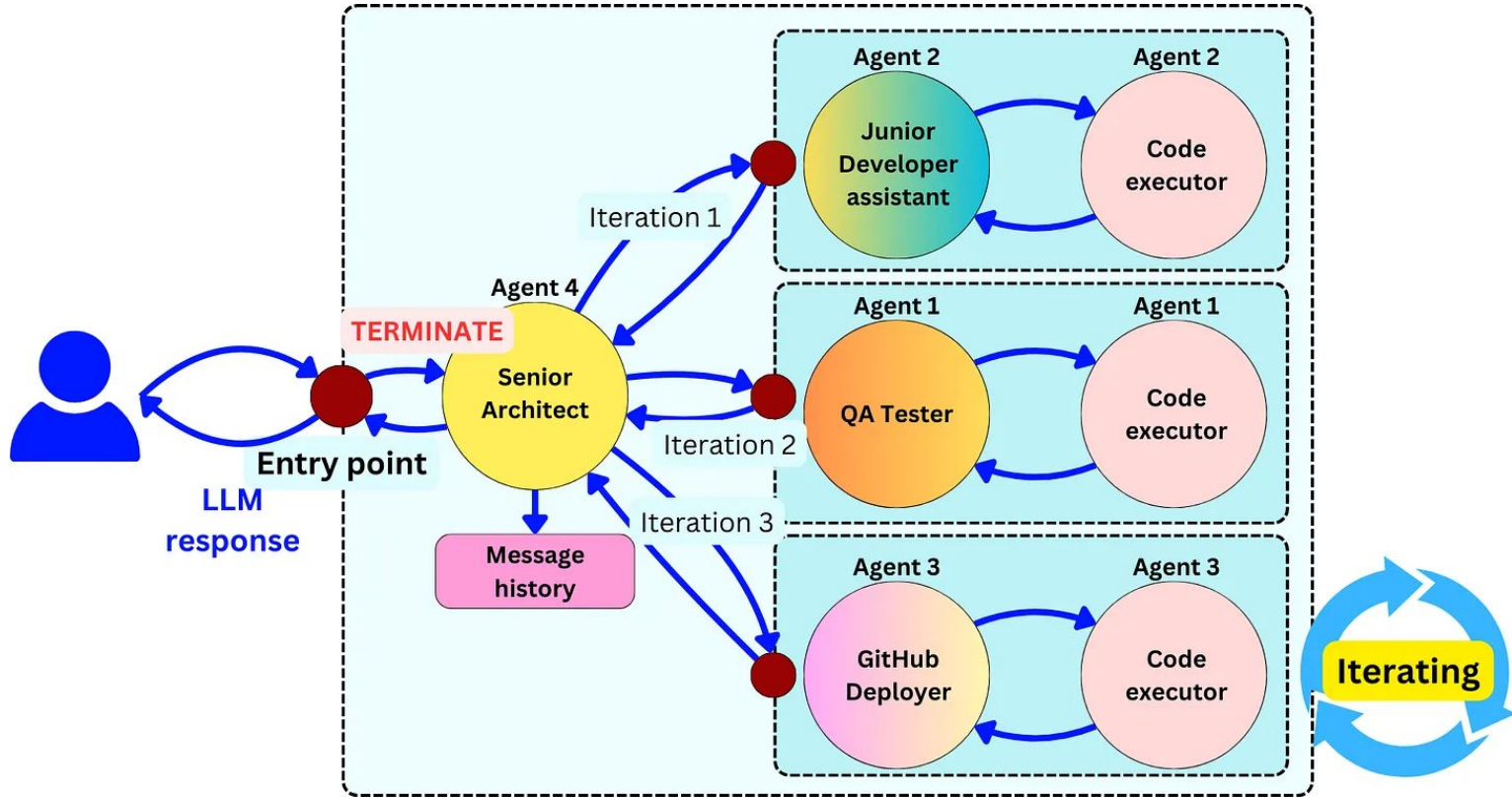
Example adapted from "HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face," Shen et al. (2023)

# Planning (cont'd)

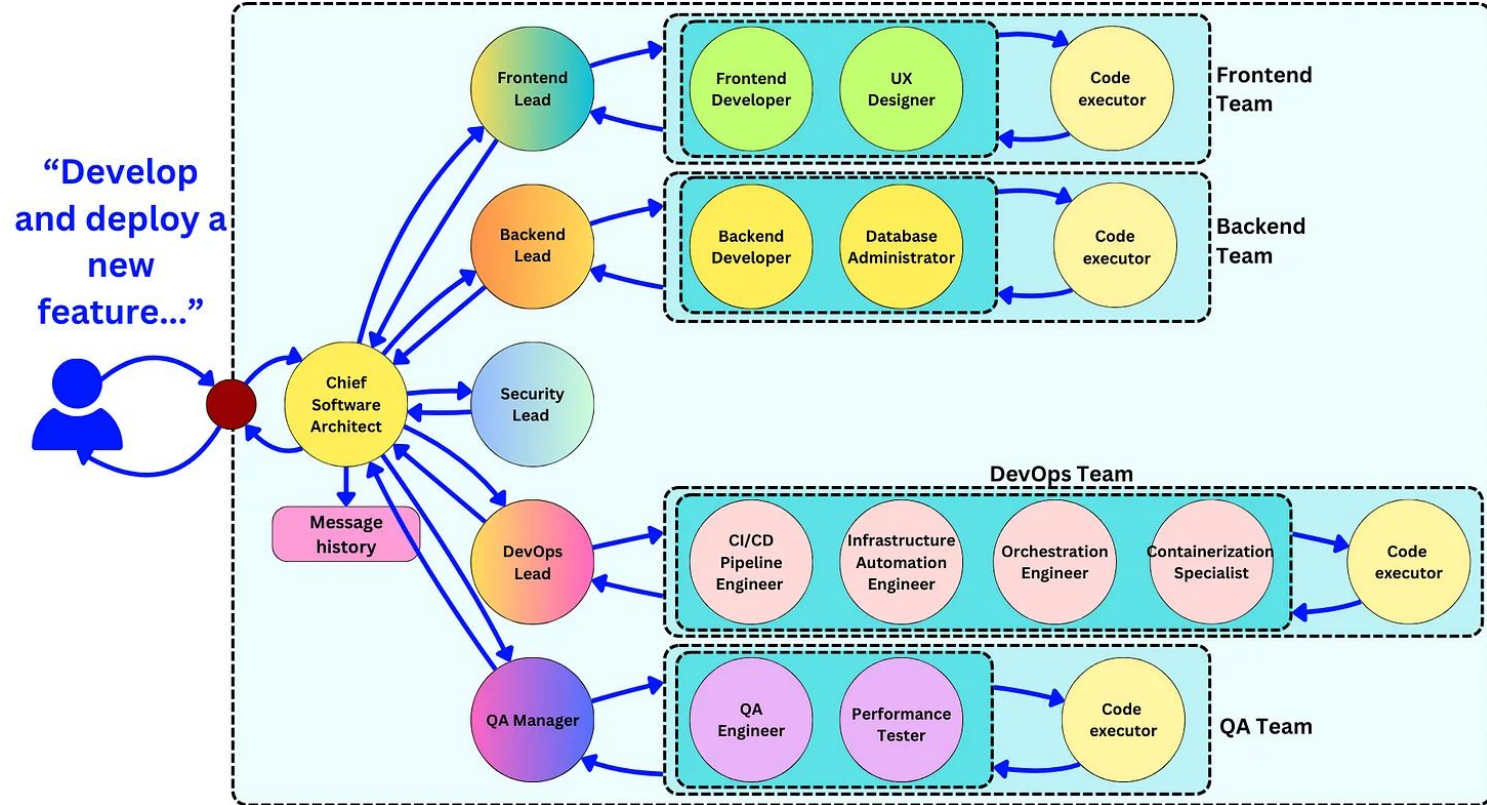
“Create a Python script that reads data from a CSV file containing customer records (name, email, purchase history), processes the data to identify the top 5 customers based on total purchase amount, and generates a new CSV file with these top customers’ information.”



# Multi-agent collaboration



# Multi-agent collaboration (cont'd)



# Main types of agents

- **Coding agents**
- **Computer-use agents**
- **Deep research / Scientific Agents**

# Popular agent use cases

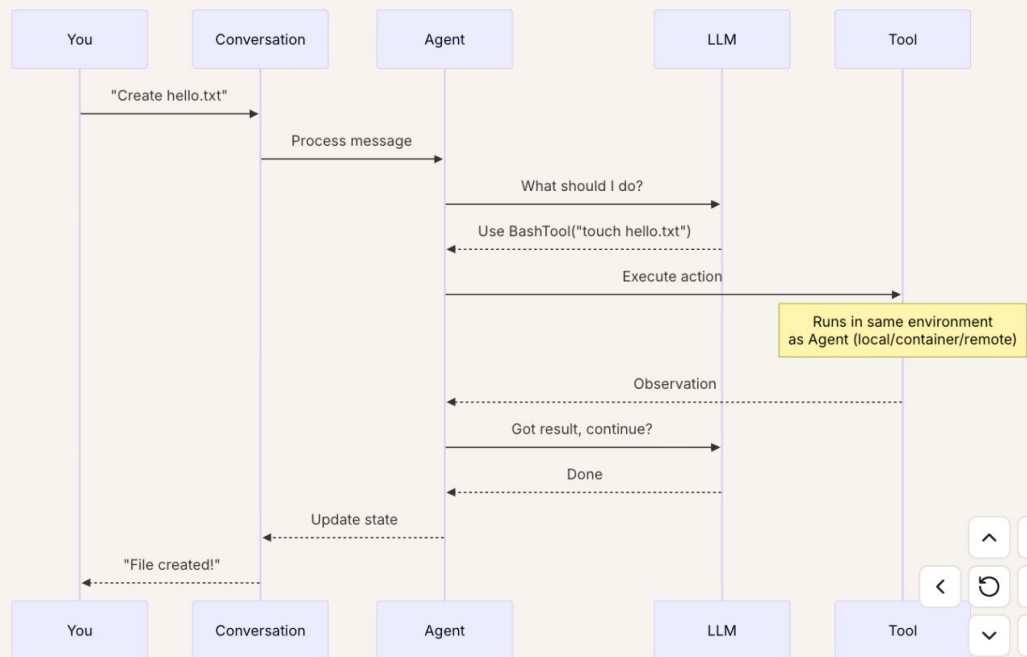
- **Coding agents**
- **Computer-use agents**
- **Deep research / Scientific Agents**

# Coding agents: Cursor demo

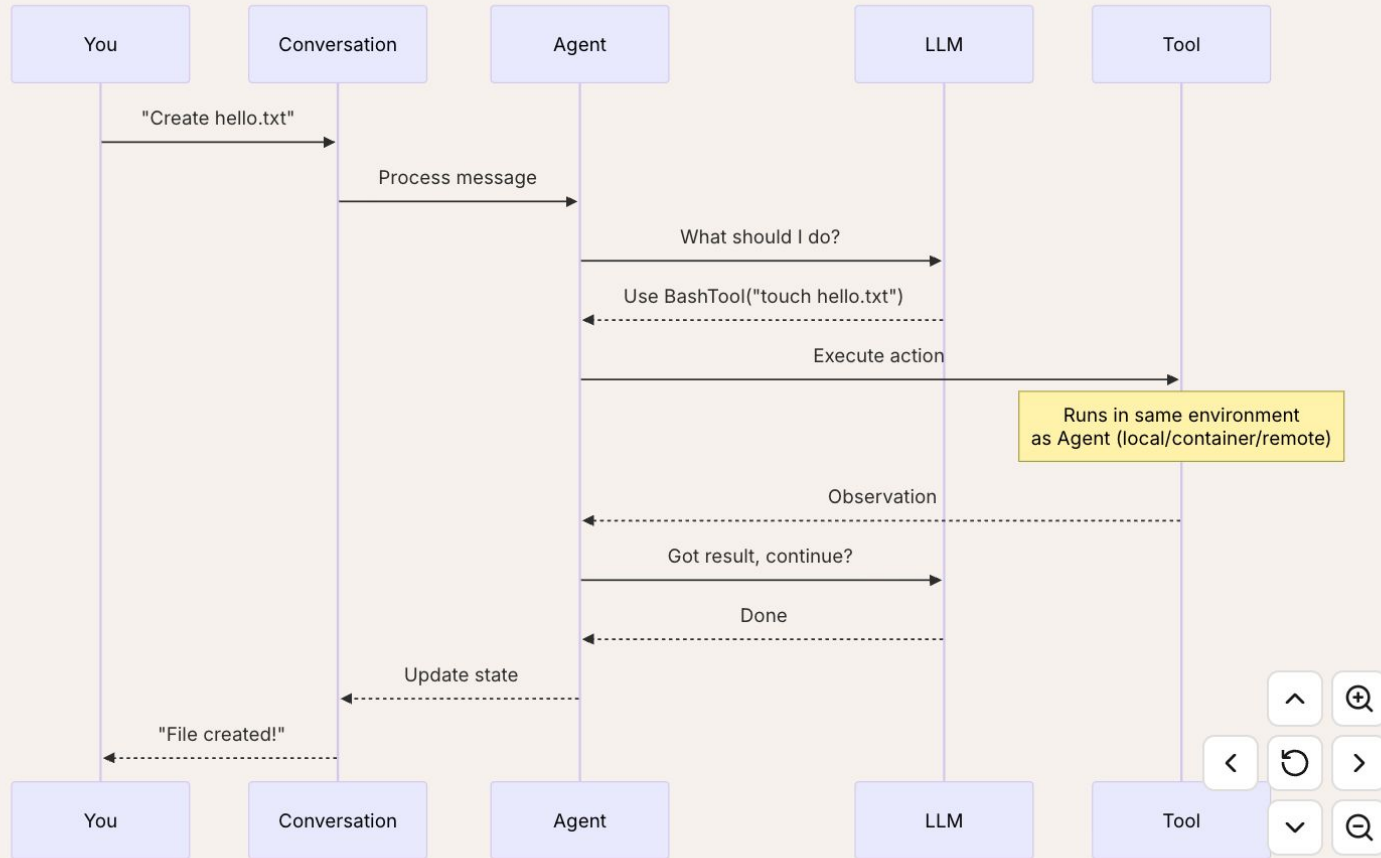
# How Components Work Together

## Basic Execution Flow (Local)

When you send a message to an agent, here's what happens:

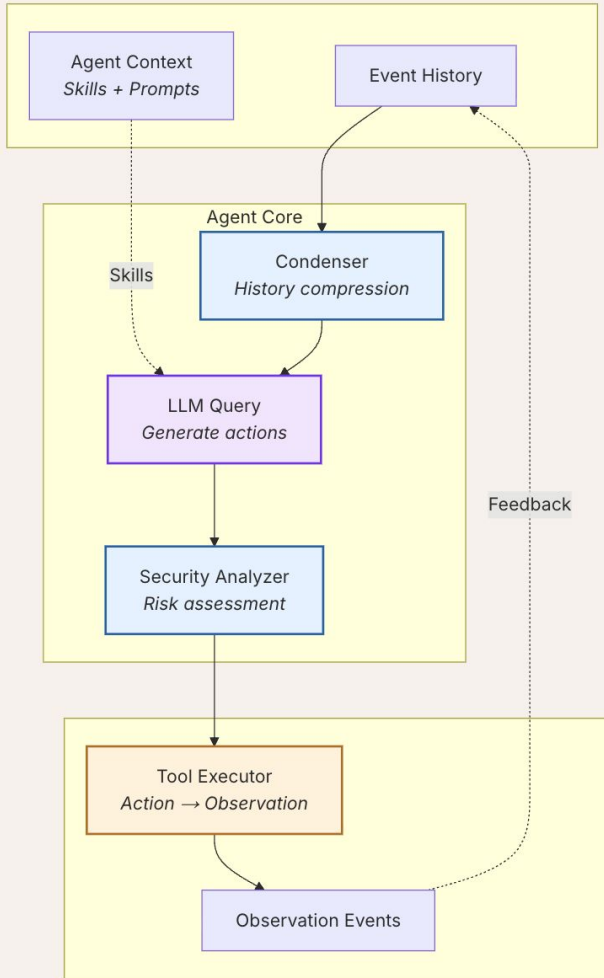


When you send a message to an agent, here's what happens:



<https://docs.openhands.dev/sdk/arch/overview>

## Architecture



# Tools

<https://github.com/OpenHands/software-agent-sdk/tree/main/openhands-tools/openhands/tools>

# Computer use agents

# OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments

Tianbao Xie<sup>1</sup>, Danyang Zhang<sup>1</sup>, Jixuan Chen<sup>1</sup>, Xiaochuan Li<sup>1</sup>,  
Siheng Zhao<sup>1</sup>, Ruisheng Cao<sup>1</sup>, Toh Jing Hua<sup>1</sup>, Zhoujun Cheng<sup>1</sup>, Dongchan Shin<sup>1</sup>, Fangyu Lei<sup>1</sup>, Yitao Liu<sup>1</sup>,  
Yiheng Xu<sup>1</sup>, Shuyan Zhou<sup>3</sup>, Silvio Savarese<sup>2</sup>, Caiming Xiong<sup>2</sup>, Victor Zhong<sup>4</sup>, Tao Yu<sup>1</sup>

<sup>1</sup>The University of Hong Kong, <sup>2</sup>Salesforce Research, <sup>3</sup>Carnegie Mellon University, <sup>4</sup>University of Waterloo

**2025-07-28: Major Upgrade! OSWorld has been enhanced and is now [OSWorld-Verified](#) with comprehensive improvements: fixed community-reported examples, **AWS support** reducing evaluation time to **within 1 hour**, and **updated benchmark results**. See the verified benchmark results in the [Benchmark section](#) below. Please compare your OSWorld results with the new benchmark results when running the latest version.**

 Paper

 Code

 Doc

 Data

 Data Viewer

 Slides

 Twitter

 Discord

# Data viewer

<https://os-world.github.io/explorer.html>

---

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

---

**Qingyun Wu<sup>†</sup>, Gagan Bansal\*, Jieyu Zhang<sup>±</sup>, Yiran Wu<sup>†</sup>, Beibin Li\***

**Erkang Zhu\*, Li Jiang\*, Xiaoyun Zhang\*, Shaokun Zhang<sup>†</sup>, Jiale Liu<sup>‡</sup>**

**Ahmed Awadallah\*, Ryen W. White\*, Doug Burger\*, Chi Wang\*<sup>1</sup>**

\*Microsoft Research, <sup>†</sup>Pennsylvania State University

<sup>±</sup>University of Washington, <sup>‡</sup>Xidian University

# ChatGPT Agents demo

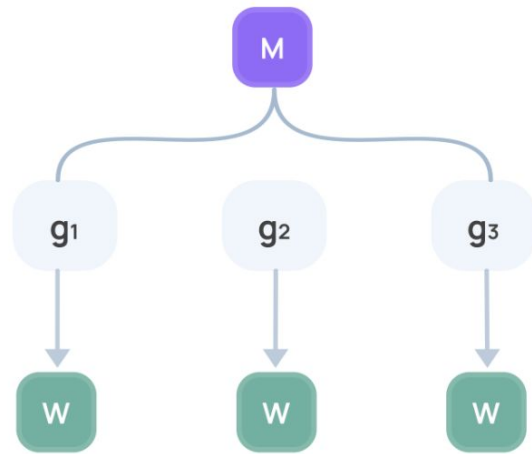
In the game Pizza Car on CrazyGames, the player is first told to follow the arrows. If the player instead turns backwards and follows the right lanes of the road in front of them, how many street lights do they pass before reaching the first intersection?

# Agent S2: A Compositional Generalist-Specialist Framework for Computer Use Agents

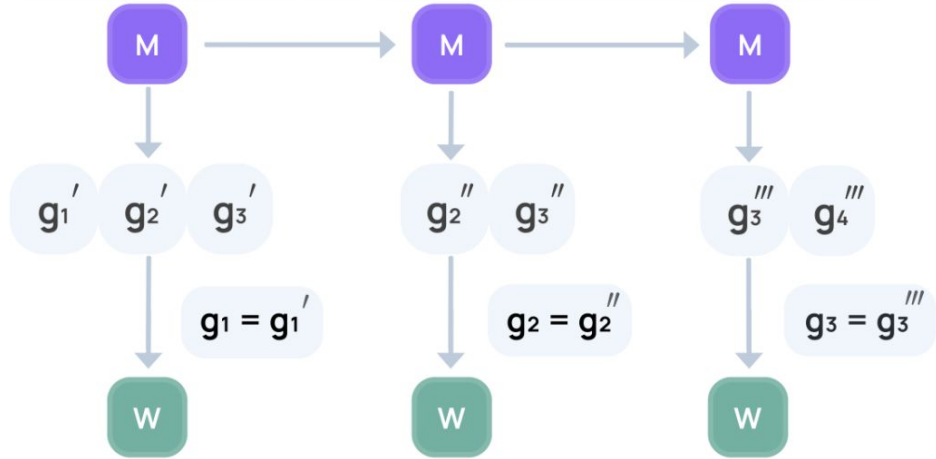
Saaket Agashe\*, Kyle Wong\*, Vincent Tu\*, Jiachen Yang, Ang Li, Xin Eric Wang  
Similar Research

## Abstract

Computer use agents automate digital tasks by directly interacting with graphical user interfaces (GUIs) on computers and mobile devices, offering significant potential to enhance human productivity by completing an open-ended space of user queries. However, current agents face significant challenges: imprecise grounding of GUI elements, difficulties with long-horizon task planning, and performance bottlenecks from relying on single generalist models for diverse cognitive tasks. To this end, we introduce Agent S2, a novel compositional framework that delegates cognitive responsibilities across various generalist and specialist models. We propose a novel Mixture-of-Grounding technique to achieve precise GUI localization and introduce Proactive Hierarchical Planning, dynamically refining action plans at multiple temporal scales in response to evolving observations. Evaluations demonstrate that Agent S2 establishes new state-of-the-art (SOTA) performance on three prominent computer use benchmarks. Specifically, Agent S2 achieves 18.9% and 32.7% relative improvements over leading baseline agents such as Claude Computer Use and UI-TARS on the OSWorld 15-step and 50-step evaluation. Moreover, Agent S2 generalizes effectively to other operating systems and applications, surpassing previous best methods by 52.8% on WindowsAgentArena and by 16.52% on AndroidWorld relatively. Code available at <https://github.com/similar-ai/Agent-S>.



Reactive Planning



Proactive Planning

Figure 3: Comparison between Reactive and Proactive Planning. Proactive planning re-evaluates and updates the remainder of the plan after every subtask, while reactive planning adheres to a fixed plan and only revises it in response to subtask failures.

---

# DEEP RESEARCH AGENTS: A SYSTEMATIC EXAMINATION AND ROADMAP

---

Yuxuan Huang<sup>1</sup>, Yihang Chen<sup>†2</sup>, Haozheng Zhang<sup>†2</sup>, Kang Li<sup>3</sup>, Huichi Zhou<sup>4</sup>, Meng Fang<sup>1</sup>, Linyi Yang<sup>4</sup>,  
Xiaoguang Li<sup>2</sup>, Lifeng Shang<sup>2</sup>, Songcen Xu<sup>2</sup>, Jianye Hao<sup>2</sup>, Kun Shao<sup>†2</sup>, and Jun Wang<sup>†4</sup>

<sup>1</sup> University of Liverpool   <sup>2</sup> Huawei Noah's Ark Lab   <sup>3</sup> University of Oxford   <sup>4</sup> University College London

## ABSTRACT

The rapid progress of Large Language Models (LLMs) has given rise to a new category of autonomous AI systems, referred to as Deep Research (DR) agents. These agents are designed to tackle complex, multi-turn informational research tasks by leveraging a combination of dynamic reasoning, adaptive long-horizon planning, multi-hop information retrieval, iterative tool use, and the generation of structured analytical reports. In this paper, we conduct a detailed analysis of the foundational technologies and architectural components that constitute Deep Research agents. We begin by reviewing information acquisition strategies, contrasting API-based retrieval methods with browser-based exploration. We then examine modular tool-use frameworks, including code execution, multimodal input processing, and the integration of Model Context Protocols (MCPs) to support extensibility and ecosystem development. To systematise existing approaches, we propose a taxonomy that differentiates between static and dynamic workflows, and we classify agent architectures based on planning strategies and agent composition, including single-agent and multi-agent configurations. We also provide a critical evaluation of current benchmarks, highlighting key limitations such as restricted access to external knowledge, sequential execution inefficiencies, and misalignment between evaluation metrics and the practical objectives of DR agents. Finally, we outline open challenges and promising directions for future research. A curated and continuously updated repository of DR agent research is available at: <https://github.com/ai-agents-2030/awesome-deep-research-agent>.

# Deep Research Agents: A Systematic Examination And Roadmap

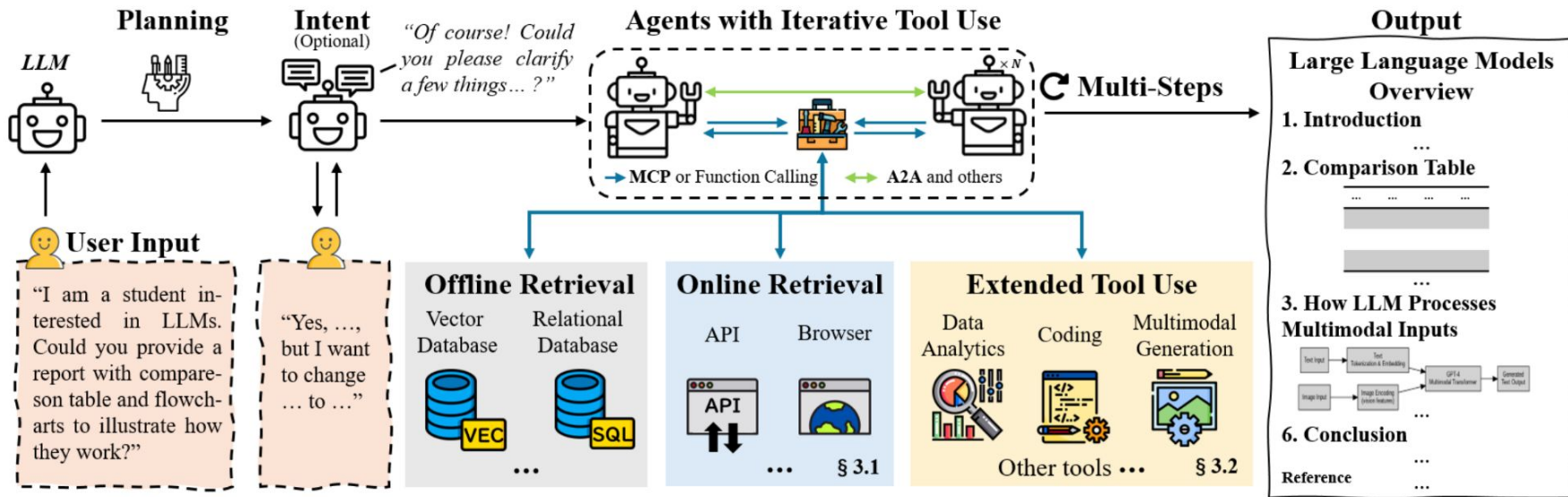


Figure 1: A structural overview of a DR agent in a multi-agent architecture for ease of illustration.

# Deep Research demo

# Synthetic Data Generation & Multi-Step RL for Reasoning & Tool Use

Anna Goldie<sup>\*1</sup> Azalia Mirhoseini<sup>\*1</sup> Hao Zhou<sup>2</sup> Irene Cai<sup>2</sup> Christopher D. Manning<sup>1</sup>

<sup>1</sup> Department of Computer Science, Stanford University

<sup>2</sup> Google DeepMind

<sup>\*</sup> Equal contribution

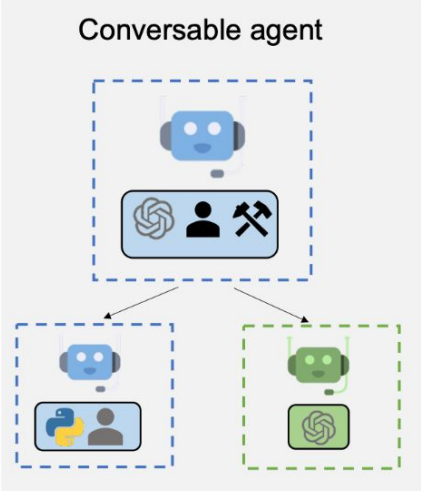
{agoldie, azalia}@cs.stanford.edu

## Abstract

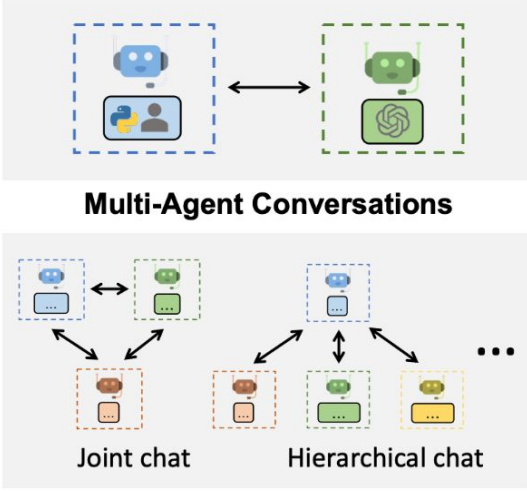
Reinforcement learning has been shown to improve the performance of large language models. However, traditional approaches like RLHF or RLAIIF treat the problem as single-step. As focus shifts toward more complex reasoning and agentic tasks, language models must take multiple steps of text generation, reasoning and environment interaction before generating a solution. We propose a synthetic data generation and RL methodology targeting multi-step optimization scenarios. This approach, called Step-Wise Reinforcement Learning (SWiRL), iteratively generates multi-step reasoning and tool use data, and then learns from that data. It employs a simple step-wise decomposition that breaks each multi-step trajectory into multiple sub-trajectories corresponding to each action by the original model. It then applies synthetic data filtering and RL optimization on these sub-trajectories. We evaluated SWiRL on a number of multi-step tool use, question answering, and mathematical reasoning tasks. Our experiments show that SWiRL outperforms baseline approaches by 21.5%, 12.3%, 14.8%, 11.1%, and 15.3% in relative accuracy on GSM8K, HotPotQA, CofCA, MuSiQue, and BeerQA, respectively. Excitingly, the approach exhibits generalization across tasks: for example, training only on HotPotQA (text question-answering) improves zero-shot performance on GSM8K (a math dataset) by a relative 16.9%.

504.04736v2 [cs.AI] 28 Apr 2025

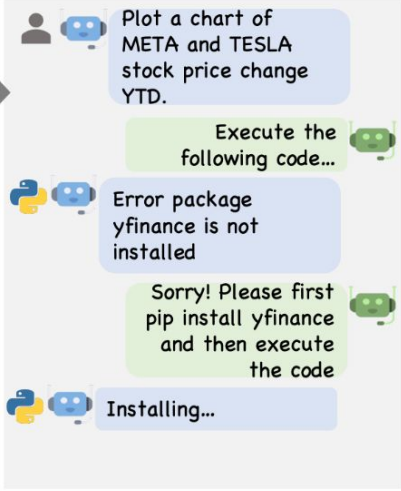
# AutoGen enables diverse LLM-based applications using multi-agent conversations



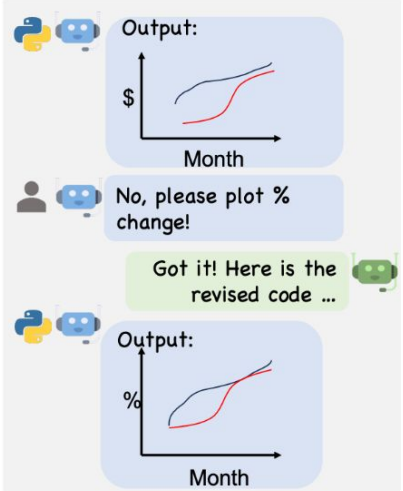
Agent Customization



Flexible Conversation Patterns



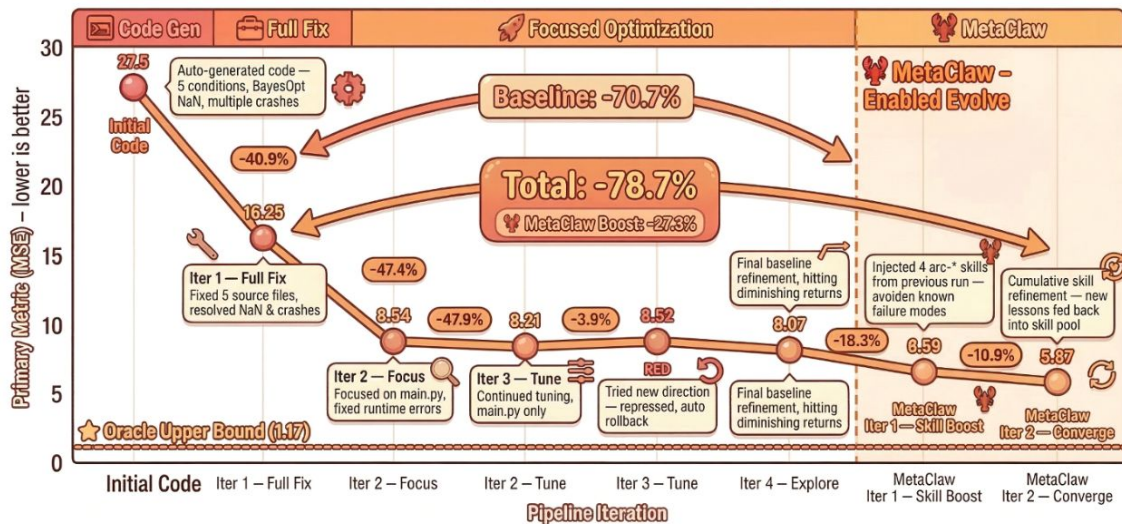
Example Agent Chat





# Autonomous Iterative Improvement

RL for Physical Law Discovery



UNC-Chapel Hill

UC Berkeley

CMU

UC Santa Cruz

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

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

<sup>1</sup>Stanford University, <sup>2</sup>UC Berkeley, <sup>3</sup>Carnegie Mellon University,

<sup>4</sup>Amazon Alexa AI, <sup>5</sup>Dashworks Technologies, Inc.,

<sup>6</sup>IIT Bombay, <sup>7</sup>Calera Capital, <sup>8</sup>Microsoft, <sup>9</sup>Two Sigma Investments

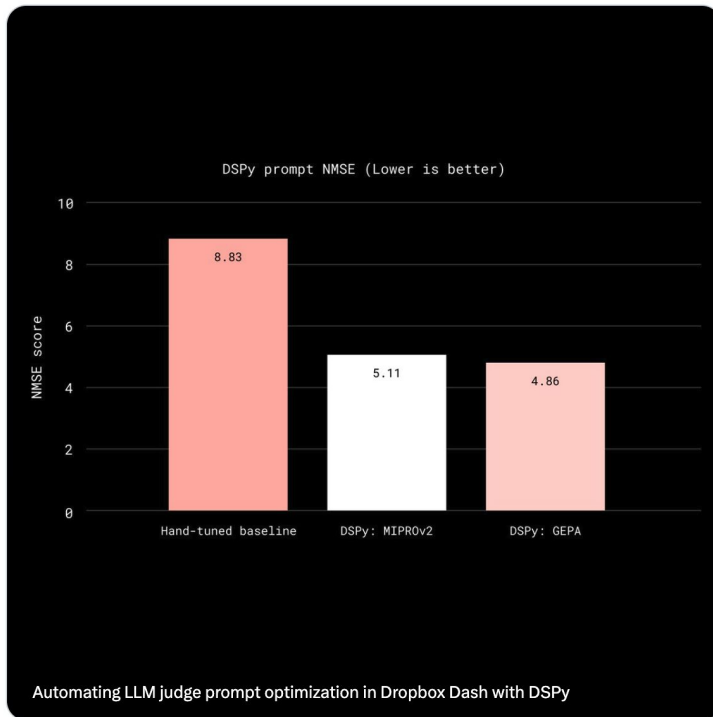
okhattab@cs.stanford.edu

```
1 class ChainOfThought(dspy.Module):
2     def __init__(self, signature):
3         # Modify signature from '*inputs -> *outputs' to '*inputs -> rationale, *outputs'.
4         rationale_field = dspy.OutputField(prefix="Reasoning: Let's think step by step.")
5         signature = dspy.Signature(signature).prepend_output_field(rationale_field)
6
7         # Declare a sub-module with the modified signature.
8         self.predict = dspy.Predict(signature)
9
10    def forward(self, **kwargs):
11        # Just forward the inputs to the sub-module.
12        return self.predict(**kwargs)
```

← Post



How we used DSPy to turn our relevance judge into a measurable optimization loop, making it more reliable and scalable in Dropbox Dash.



From [dropbox.tech](https://dropbox.tech)

3:30 PM · Mar 17, 2026 · 76.1K Views

# GEPA: REFLECTIVE PROMPT EVOLUTION CAN OUTPERFORM REINFORCEMENT LEARNING

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

<sup>1</sup>UC Berkeley   <sup>2</sup>Stanford   <sup>3</sup>BespokeLabs.ai   <sup>4</sup>Notre Dame   <sup>5</sup>Databricks   <sup>6</sup>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* often provides a much richer learning medium for LLMs, compared to policy gradients derived from sparse, scalar rewards. To test this, we introduce GEPA (**Genetic-Pareto**), 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 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 six tasks, GEPA outperforms GRPO by 6% 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% (e.g., +12% accuracy on AIME-2025), and demonstrates promising results as an inference-time search strategy for code optimization. We release our code at <https://github.com/gepa-ai/gepa>.

# 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.

The screenshot shows a structured prompt for a second-hop query generation task. It includes sections for the task description, input fields, purpose and context, key observations and lessons, and practical strategy. The prompt is designed to guide the model in generating a query that retrieves relevant documents not found in the first hop, based on the original question and a summary of the first hop's results.

**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:**

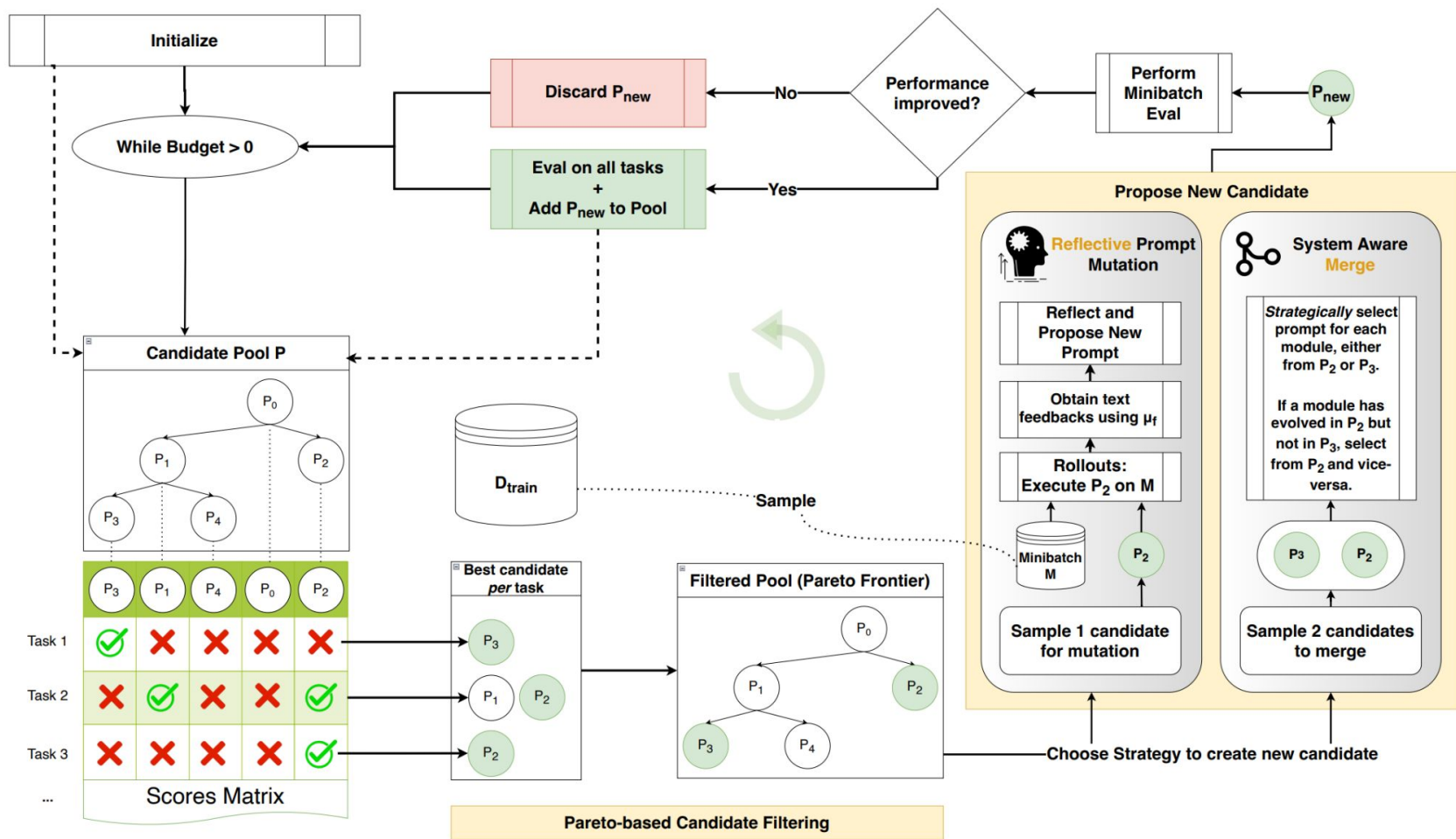
- Identify entities or topics mentioned in summary\_1 that are related but different from first-hop documents.
- Refine the query to explicitly mention those broader or related entities/concepts in the original question.
- Include relevant keywords from the question to maintain specificity, but shift focus to the missing piece.
- The goal is to retrieve documents that link or complement what was retrieved initially.

**Practical Strategy:**

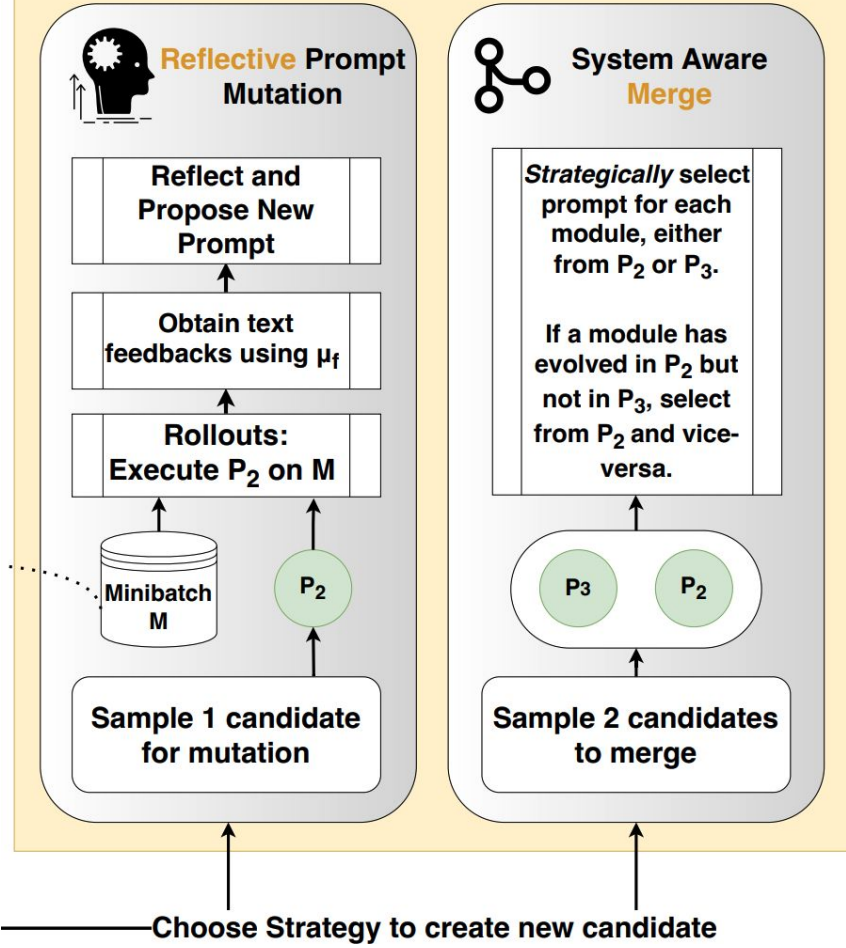
- Read the summary\_1 carefully to spot references to higher concepts or other entities not covered in the first hop.
- Ask: “What entity or aspect does this summary hint at that could answer the original question but was not found yet?”
- Formulate a precise, focused second-hop query targeting that 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 retains logical links to the original question while expanding the breadth of 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 retrieved. . . .



# Propose New Candidate



**Thank you!**