



CS329Z: Engineering AI Agents

Agent Design Patterns and Scaffolds

Diyi Yang, Michael Ryan, John Yang

Announcements

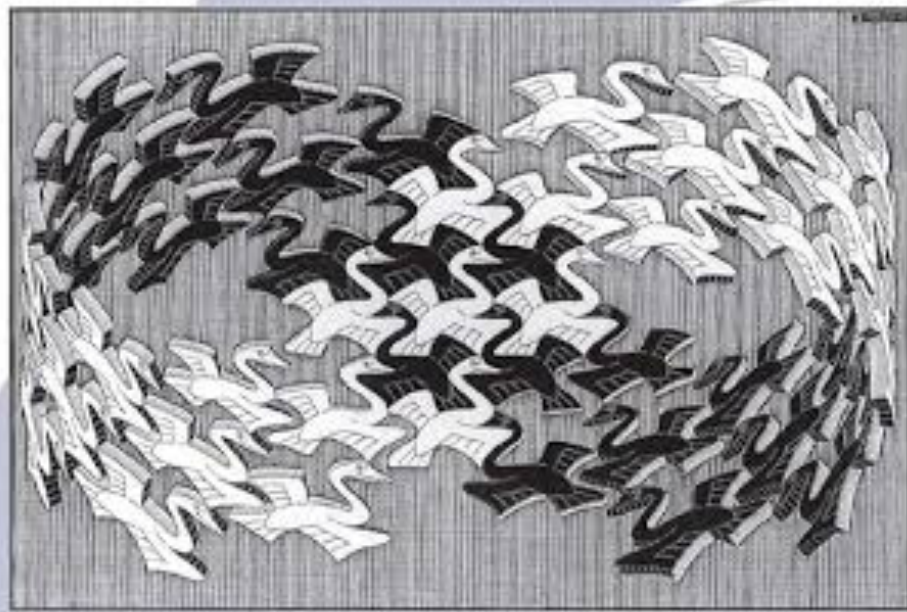
- Google credit submissions are due ASAP. Do this today if you haven't yet! Details on Ed.
- Please be careful about ICLR submission format. We will be mostly lenient about this proposal, but will enforce for midpoint/final reports.
- Homework 1 is due Friday October 30th.
- Midpoint demo due November 4th, report due November 6th.



Design Patterns

Elements of Reusable Object-Oriented Software

Erich Gamma
Richard Helm
Ralph Johnson
John Vlissides

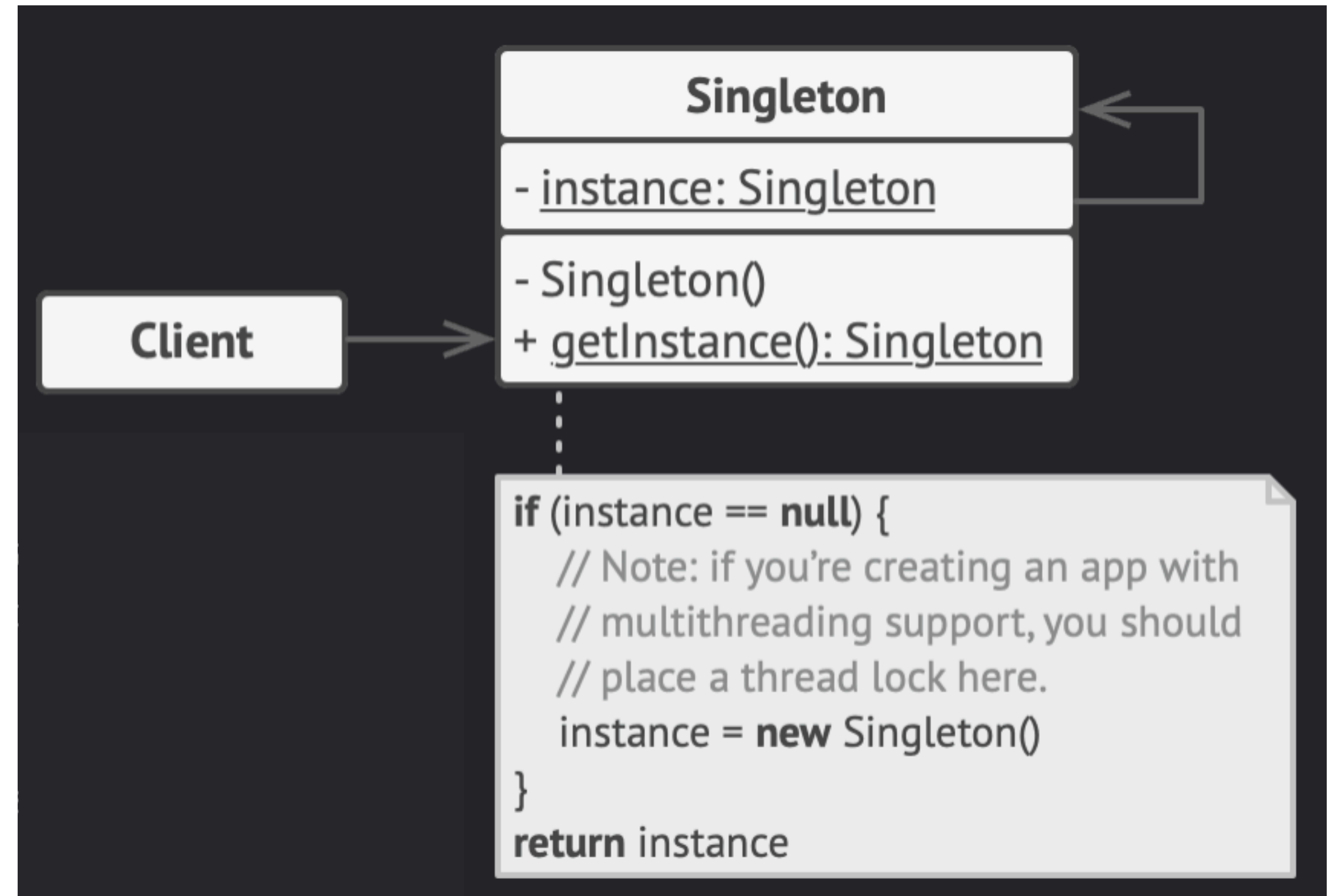


Cover art © 1994 M.C. Escher / Gordon Art - Baarn - Holland. All rights reserved.

Foreword by Grady Booch



ADDISON-WESLEY PROFESSIONAL COMPUTING SERIES



“Patterns are half-baked—meaning you always have to finish them yourself and adapt them to your own environment.”

Martin Fowler — author of “Patterns of Enterprise Application Architecture”



When building an agent system you must make many design decisions.

Today we'll introduce **design patterns**.

In practice you will **customize your solutions**.



Learning Objectives

- **Define** common design patterns for workflows and agents.
- **Compare and Contrast** different design patterns for context management for agents.
- **Analyze** the design decisions present in a production agent harness (e.g. Pi)



Lecture Overview

- Review: Workflows vs Agents (5 minutes)
- Workflow Design Patterns (15 minutes)
- Agent Design Patterns: Reasoning/Acting (20 minutes)
- Agent Design Patterns: Context (20 minutes)
- Case Study: Pi (10 minutes)



Review: Workflows and Agents

Workflows vs Agents

Workflow Designs

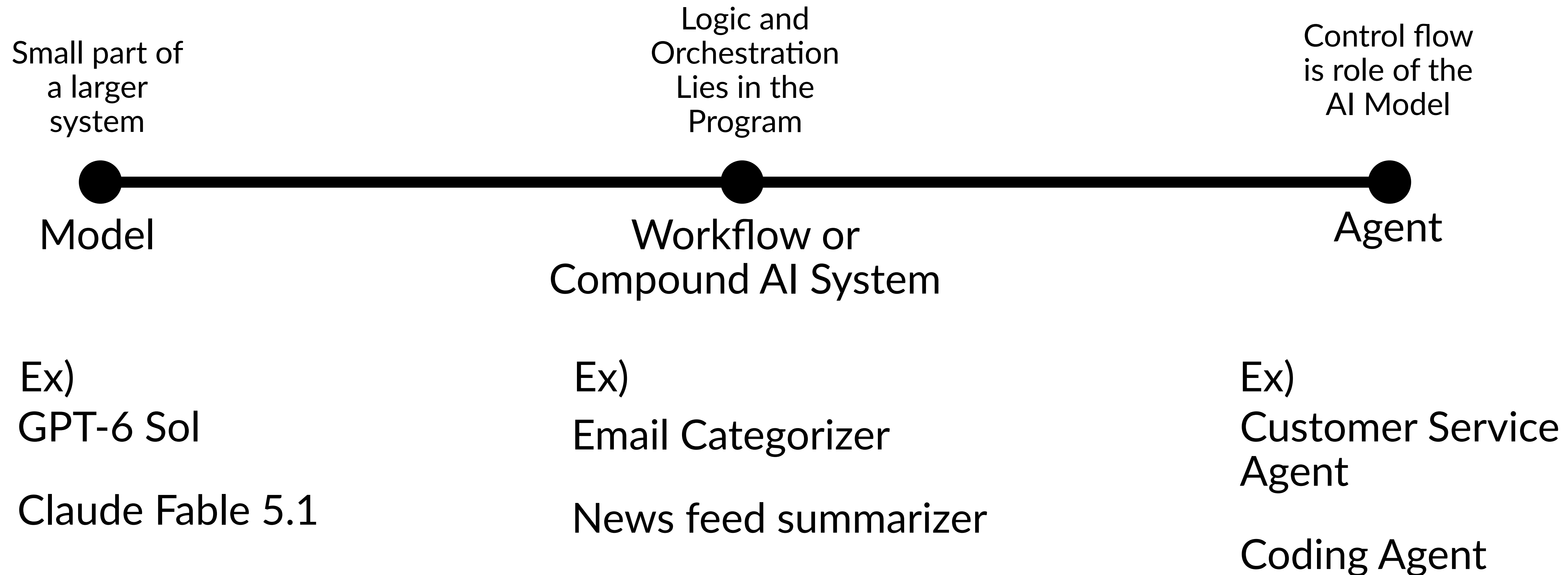
Reasoning Designs

Context Designs

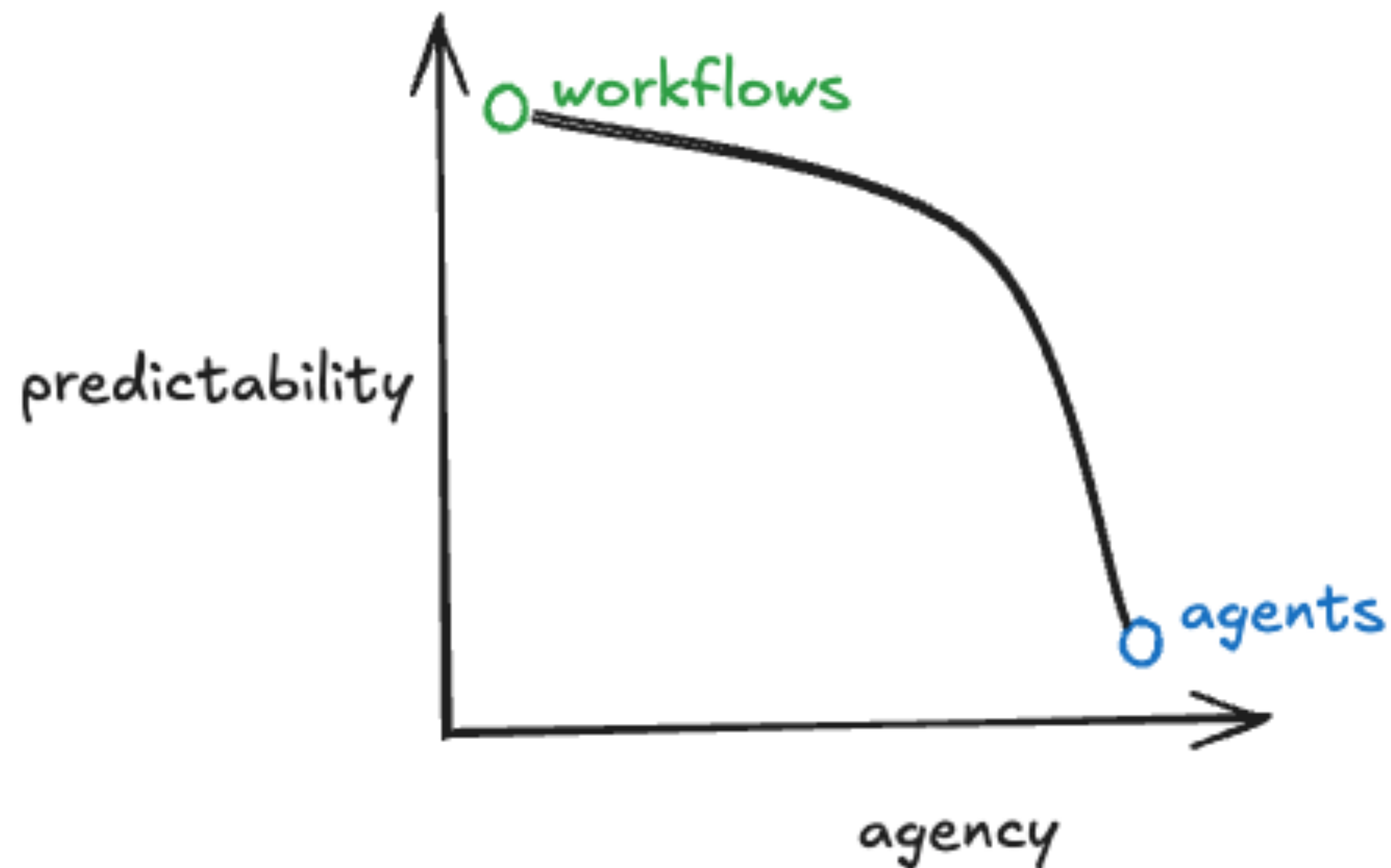
Case Study



Where does the control flow lie?



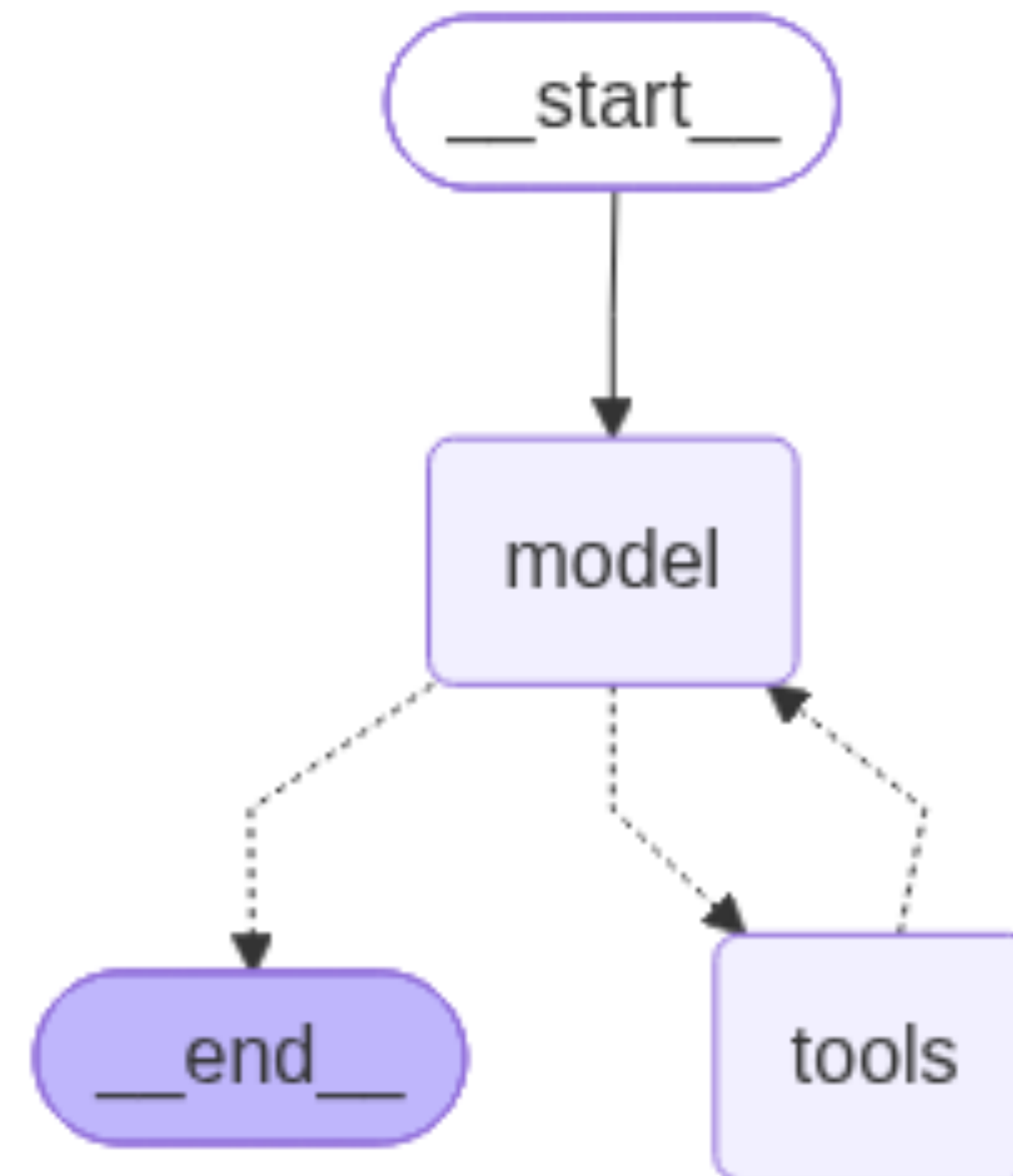
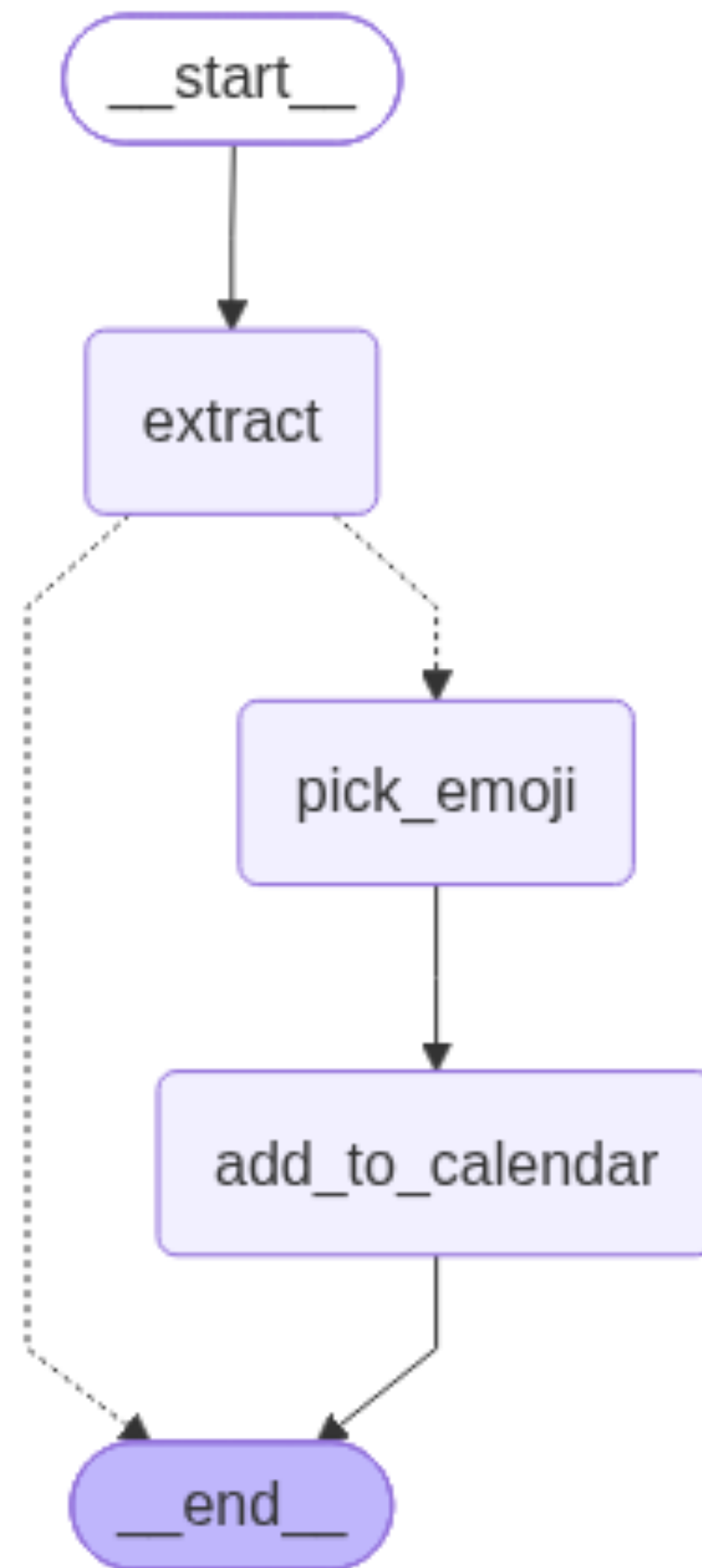
Where does the control flow lie?



- Workflows are more predictable than agents because the control flow lies in the program logic.
- Agents are more adaptable/flexible than workflows because the control flow lies in the LLMs decisions.



Workflows vs Agents (Example)



Workflows vs Agents

	Workflows	Agents
Control Flow	Code	LLM
Adaptation	Fixed (programmed) branches, routes, and loops	Chooses next step based on observations
Best Fit	Tasks with a clear, repeatable structure	Tasks requiring exploration and flexible decisions
Example	Classify support ticket -> retrieve policy -> draft automatic reply	Resolve support ticket using tools to investigate/solve the problem.

Real systems often combine workflows and agents.



Workflow Design Patterns

Workflows vs Agents

Workflow Designs

Reasoning Designs

Context Designs

Case Study



Anthropic Workflow Taxonomy

<https://www.anthropic.com/engineering/building-effective-agents>

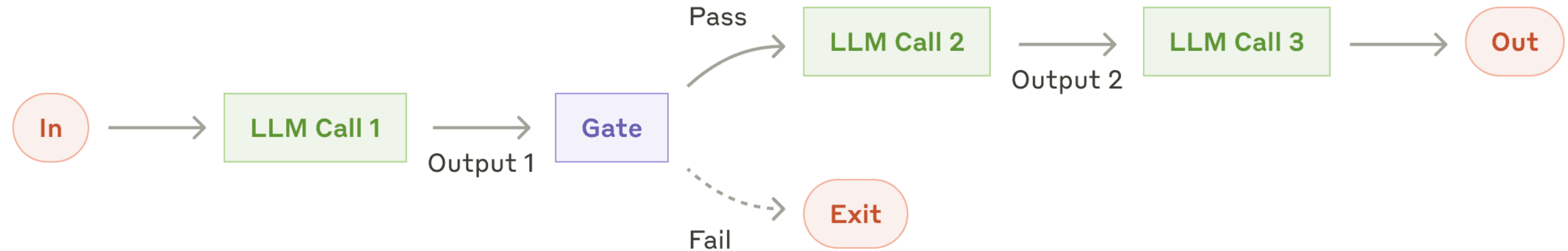


Workflow Design Patterns

- **Prompt Chaining:** Decomposing the problem into smaller steps.
- **Routing:** Route the task to appropriate LLM.
- **Parallelization:** Divide the work amongst multiple threads.
- **Orchestrator-Workers:** Identify subtasks and dispatch workers to complete them.
- **Evaluator-Optimizer:** Generate an output, critique or respond, and iterate on the output.



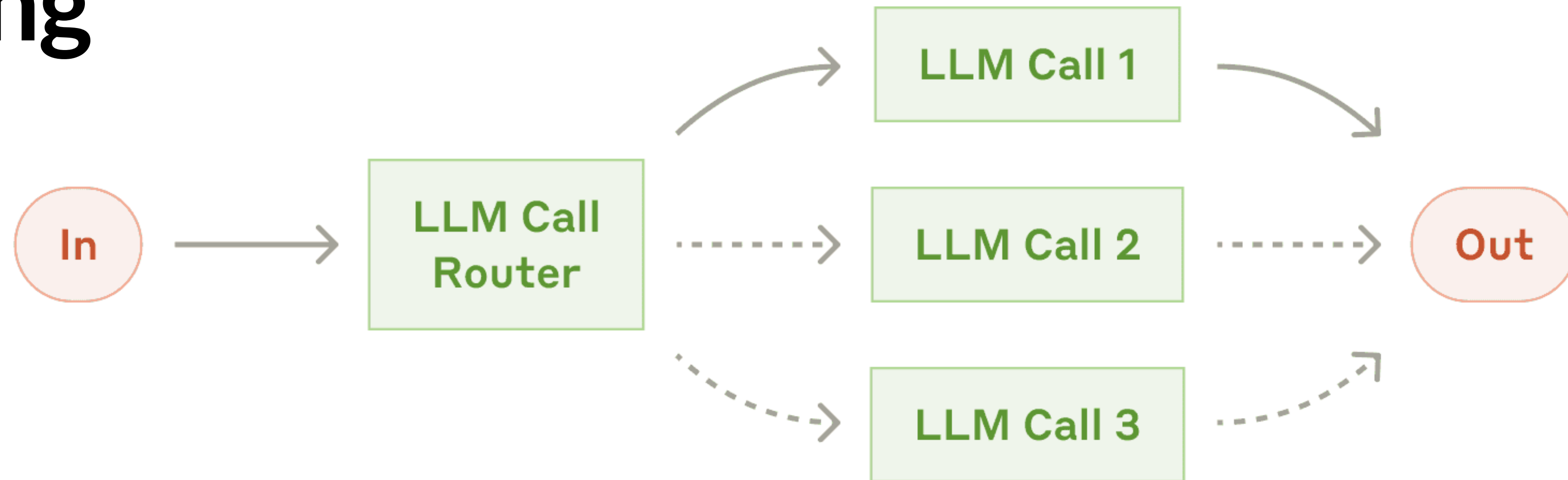
Prompt Chaining



- Decompose a larger task into a series of smaller steps
- Trades off latency for accuracy
- Useful when a task is complicated and can benefit from decomposition



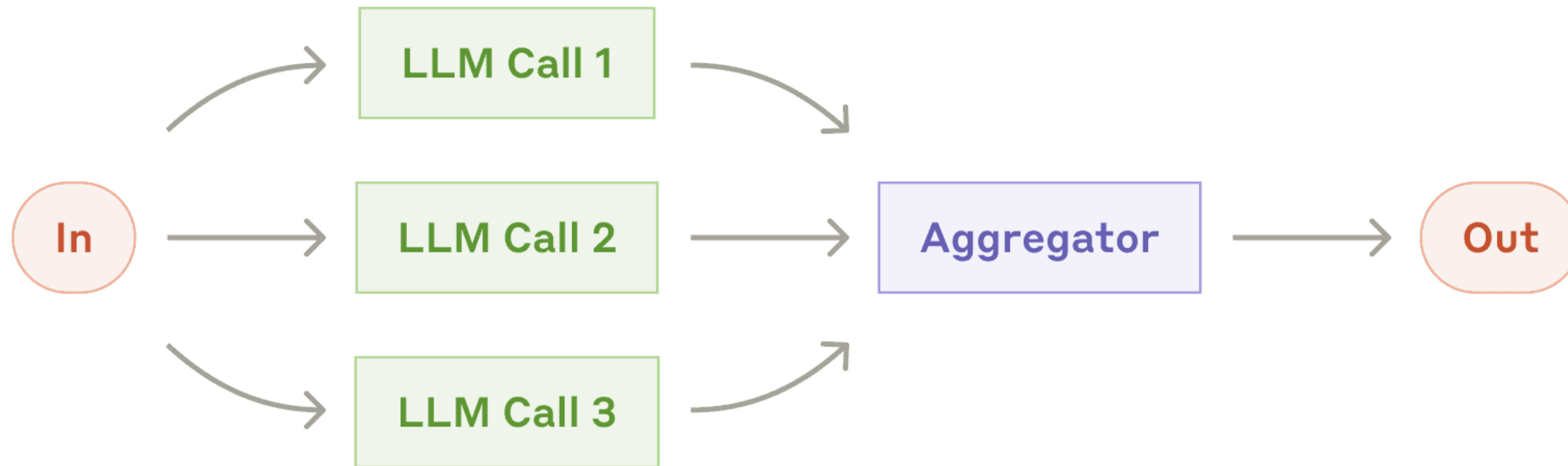
Routing



- Classify an input and route it to the appropriate LLM (could vary on model or prompt to handle different cases)
- Helps when a complex task has several categories that could help to process differently
- Can also be used to save money/time (weaker model for easier cases)



Parallelization



- **Sectioning:** Splitting up parallel independent work (e.g. flagging a query as safe/unsafe while simultaneously processing that query)
- **Voting:** Inference-time scaling where you process the same input several times and the aggregator makes the final conclusion
- Enables focusing each call on an individual aspect



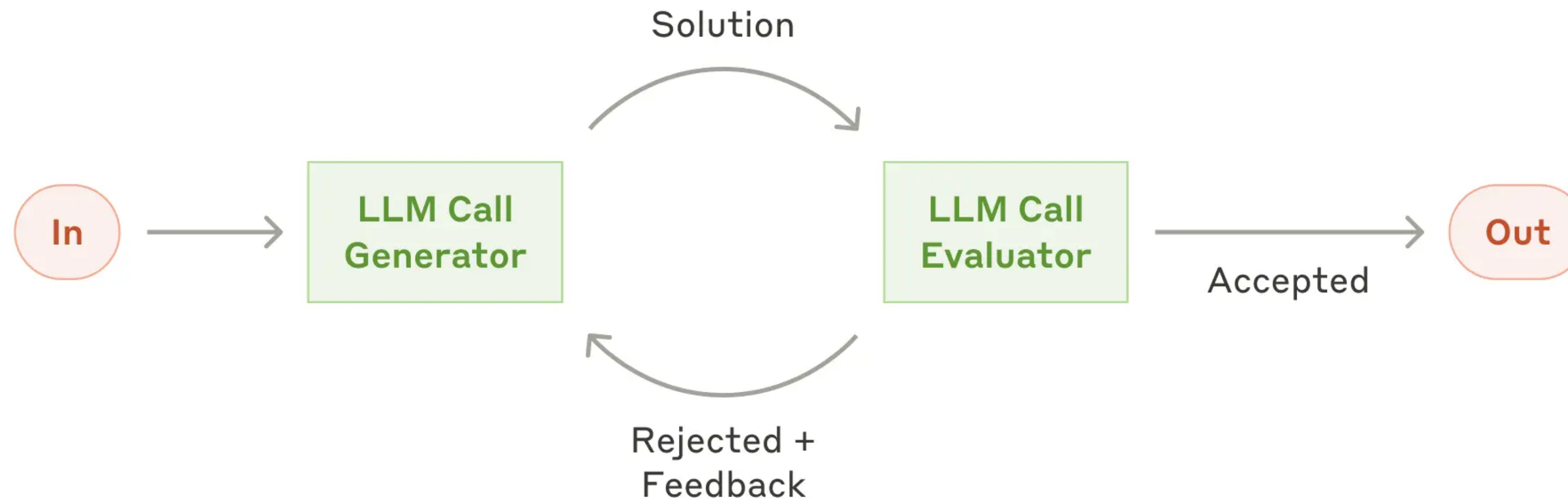
Orchestrator-Workers



- One LLM dynamically breaks down tasks and delegates to worker LLMs
- Useful when it is difficult to anticipate the nature and number of subtasks needed
- Similar to parallelization but in this case the subtasks are dynamically created.



Evaluator-Optimizer



- One LLM generates the response and another provides evaluation and feedback in a loop.
- Two necessary conditions: (1) Clear articulable feedback and (2) an LLM is capable of providing that feedback.
- Also important that the generator can improve based on the feedback.



Composition of Workflow Patterns

Real systems are compositions of these basic patterns.



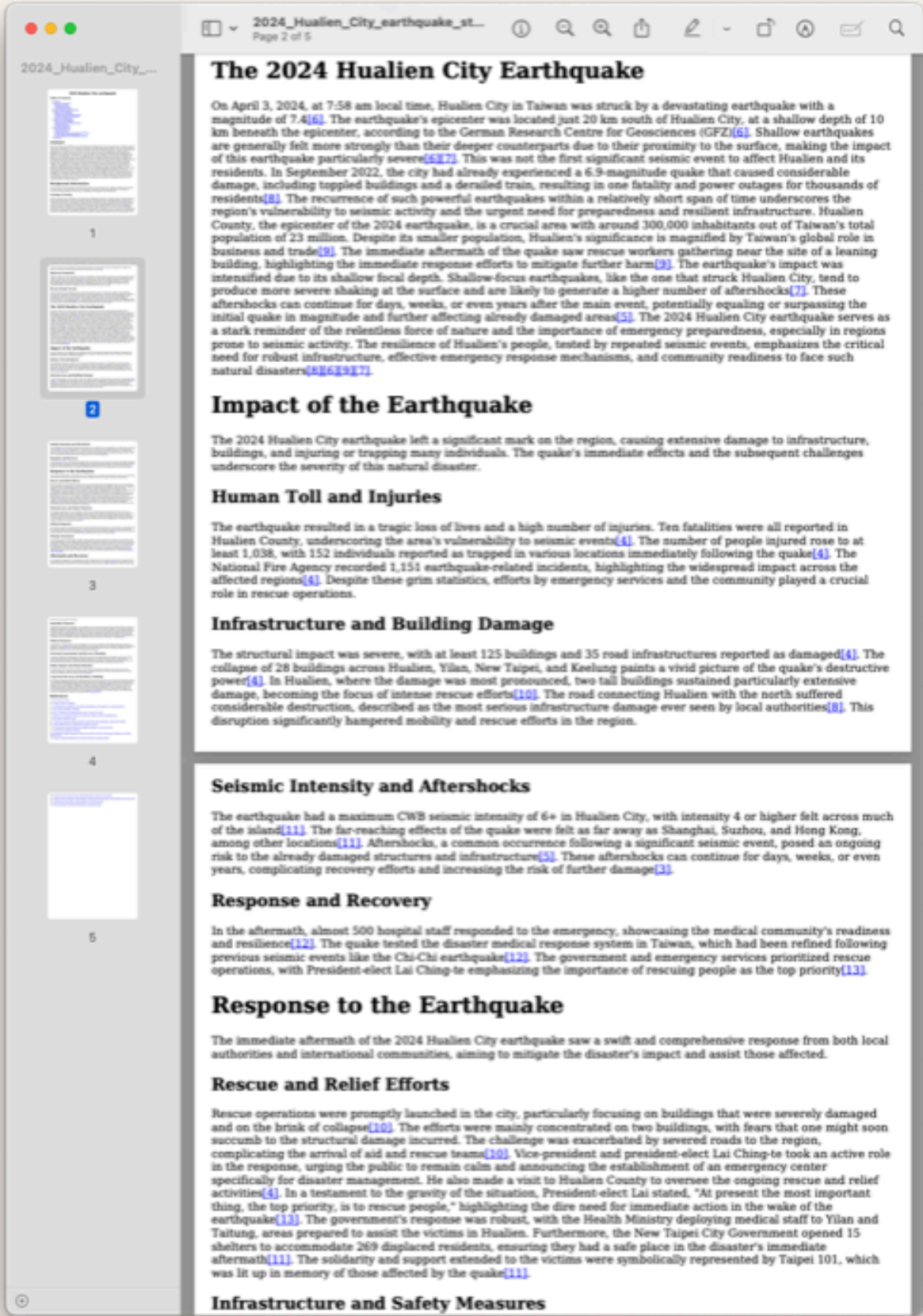
Example: STORM

I'd like to know more about:
2024 Hualien City earthquake

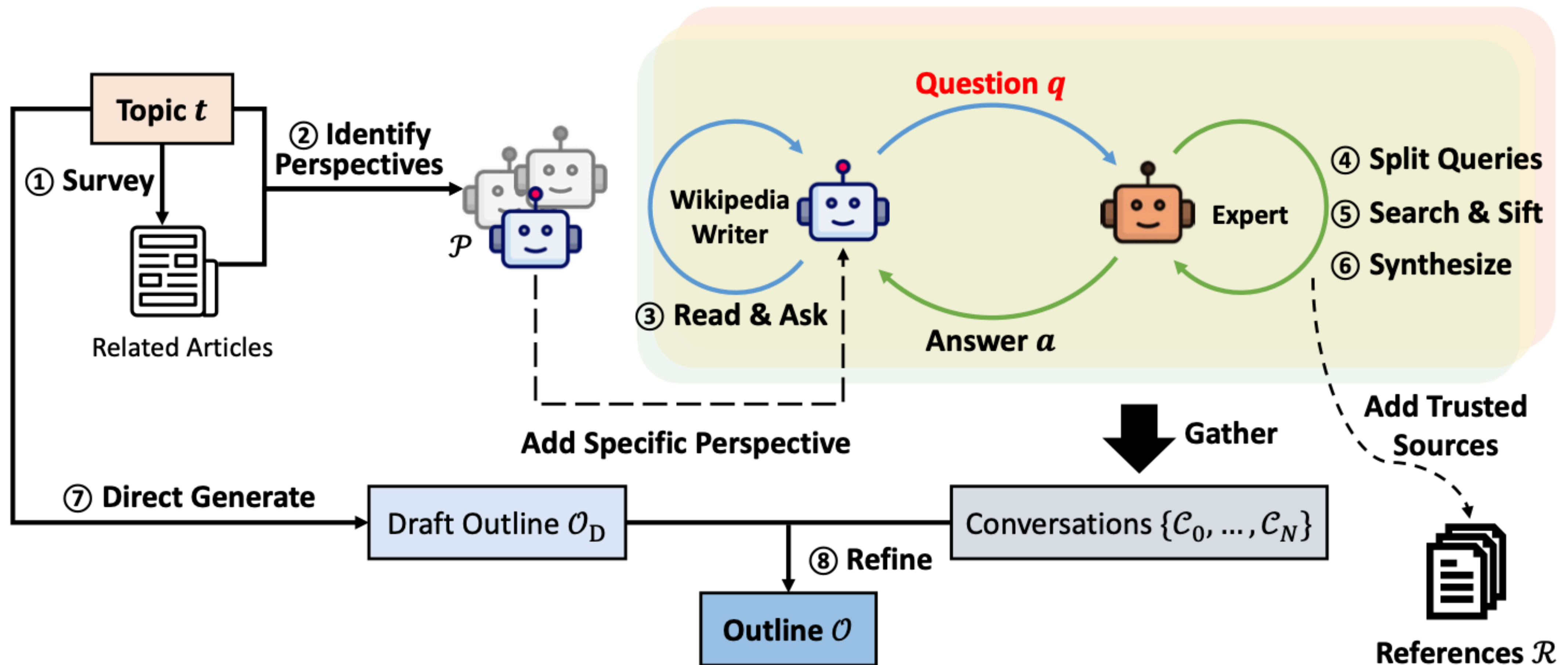
STORM 

Table of contents

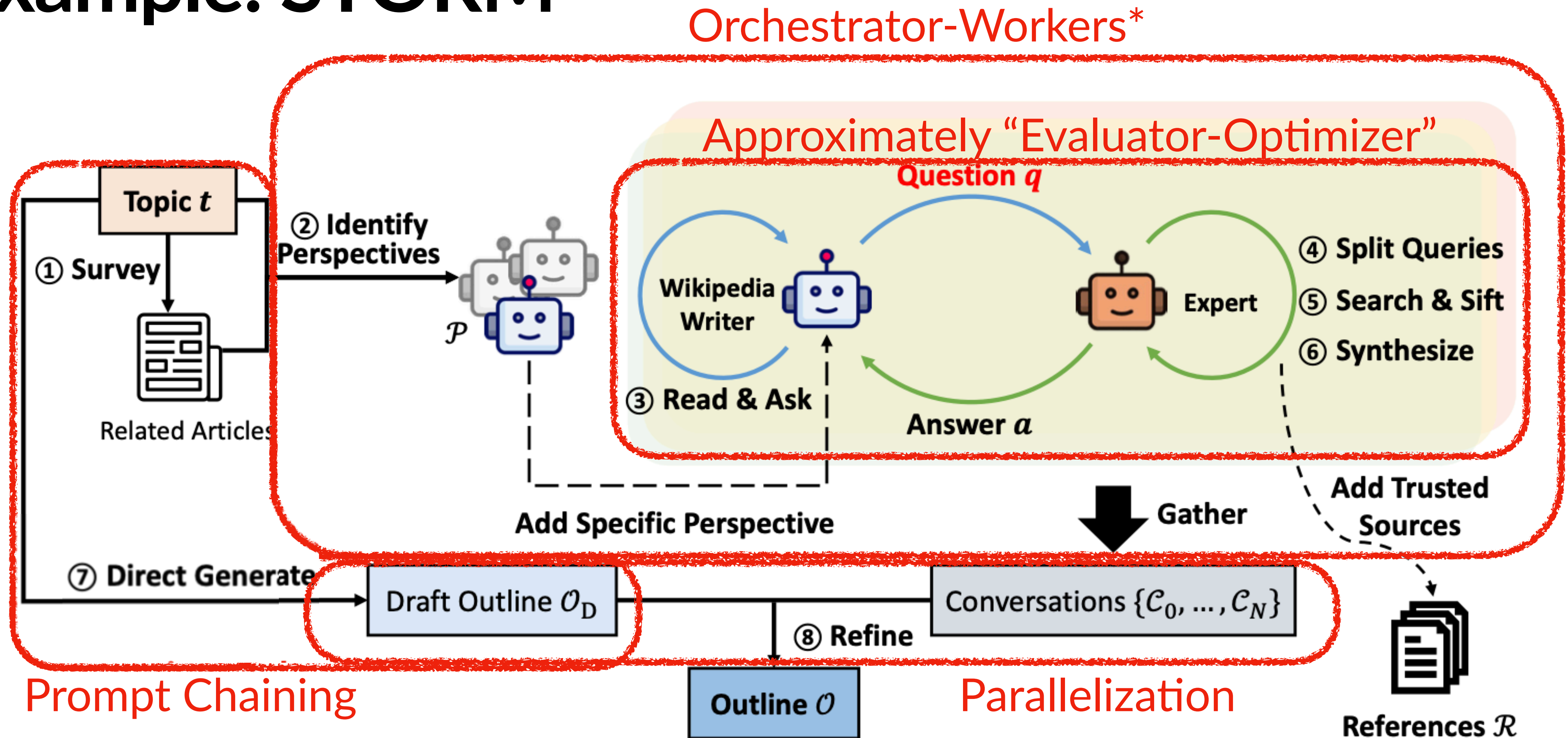
- Background Information
 - Geological Setting
 - Historical Seismicity
 - Recent Seismic Events
- The 2024 Hualien City Earthquake
- Impact of the Earthquake
 - Human Toll and Injuries
 - Infrastructure and Building Damage
 - Seismic Intensity and Aftershocks
 - Response and Recovery
- Response to the Earthquake
 - Rescue and Relief Efforts
 - Infrastructure and Safety Measures
 - Medical Response
 - Damage Assessment
- Aftermath and Recovery
 - Immediate Response
 - Medical Response
 - Structural Assessments and Recovery Planning
 - Public Support and Memorialization
 - Long-Term Recovery and Resilience



Example: STORM



Example: STORM



*This is close to parallelization in this instance

Image Credit: <https://arxiv.org/pdf/2402.14207>



Example: STORM

	Comparsion with Human-written Articles			Rubric Grading			
	ROUGE-1	ROUGE-L	Entity Recall	Interest Level	Organization	Relevance	Coverage
Direct Gen	25.62	12.63	5.08	2.87	4.60	3.10	4.16
RAG	28.52	13.18	7.57	3.14	4.22	3.05	4.08
oRAG	44.26	16.51	12.57	3.90	4.79	4.09	4.70
STORM	45.82	16.70	14.10[†]	3.99[†]	4.82	4.45[†]	4.88[†]
w/o Outline Stage	26.77	12.77	7.39	3.33	4.87	3.35	4.37

Table 2: Results of automatic article quality evaluation. [†] denotes significant differences ($p < 0.05$) from a paired t -test between STORM and the best baseline, *i.e.*, oRAG. The rubric grading uses a 1-5 scale.



Archon: An Architecture Search Framework for Inference-Time Techniques

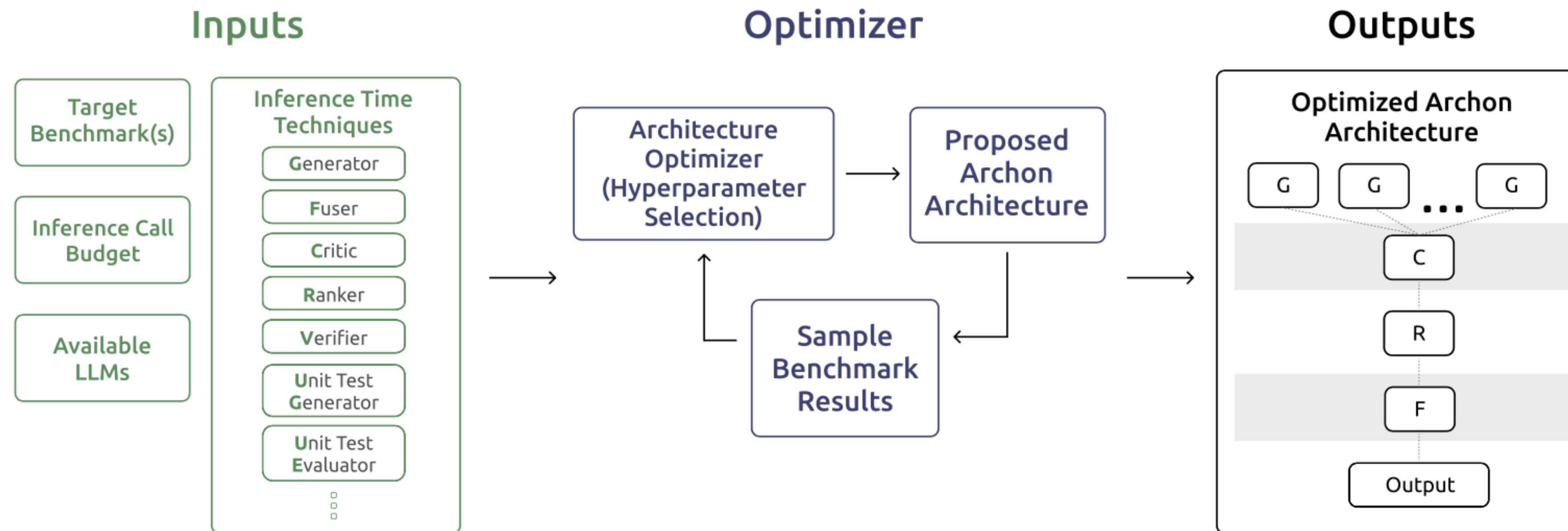


Figure 2: **Overview of ARCHON Framework:** ARCHON's search algorithm requires the following inputs: target benchmarks, inference call budget, available LLMs, and available inference-time techniques (**left**). The search algorithm uses Bayesian optimization (Snoek et al., 2012) to construct and evaluate different ARCHON configurations (**middle**) before returning the optimized ARCHON architecture (**right**) for the target benchmarks (Section 3.3).



Archon

Inference-Time Technique	Definition	Input	Output	Inference Cost	Domains
Generator	Generates a candidate response from an instruction prompt	Instruction Prompt	Candidate Response(s)	1 call per cand.	All Domains
Fuser	Merges multiple candidate responses into a single response	Instruction Prompt + Candidate Response(s)	Fused Candidate Response(s)	1 call per output	All Domains
Critic	Generates strengths/weaknesses for each candidate response	Instruction Prompt + Candidate Response(s)	Candidate Response(s) Strengths/Weaknesses	1 call	All Domains
Ranker	Returns top-K candidate responses	Instruction Prompt + Candidate Response(s)	Ranked Candidate Response(s)	1 call	All Domains
Verifier	Returns the candidate responses with verified reasoning	Instruction Prompt + Candidate Response(s)	Verified Candidate Response(s)	2 calls per cand.	Reasoning Tasks
Unit Test Generator	Generates unit tests to evaluate the candidate responses	Instruction Prompt	Instruction Prompt + Unit Tests	1 call	Reasoning Tasks
Unit Test Evaluator	Uses generated unit tests to evaluate candidate response	Instruction Prompt + Unit Tests + Candidate Response(s)	Scored Candidate Response(s)	1 call per cand.	Reasoning Tasks



Archon

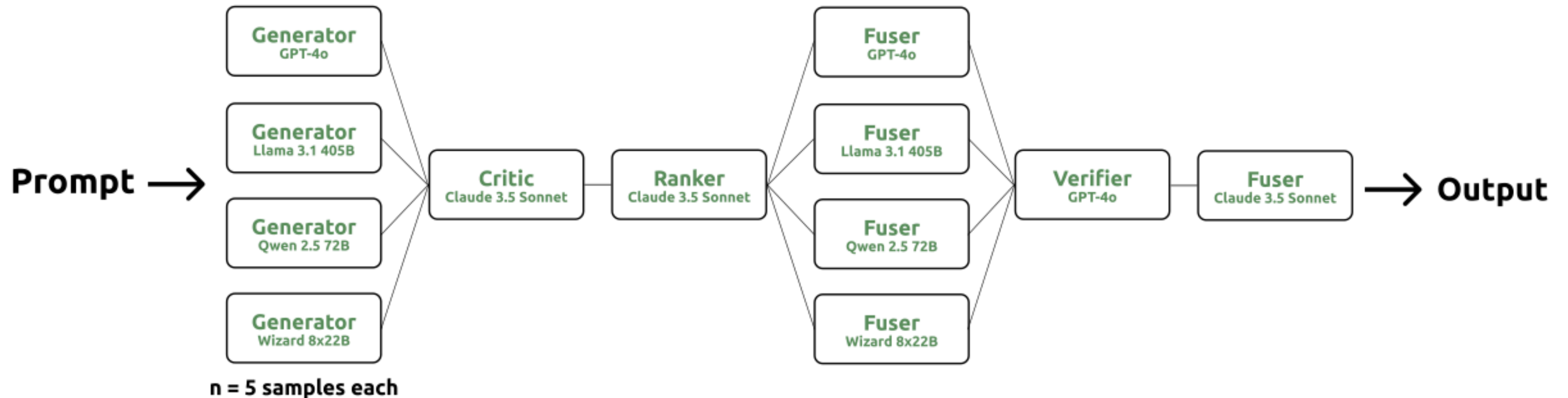


Figure 15: **All-Source ARCHON Architecture for Math:** Using ARCHON’s architecture search, we found this all-source ARCHON configuration to be effective across the math benchmarks explored (MATH).



Archon

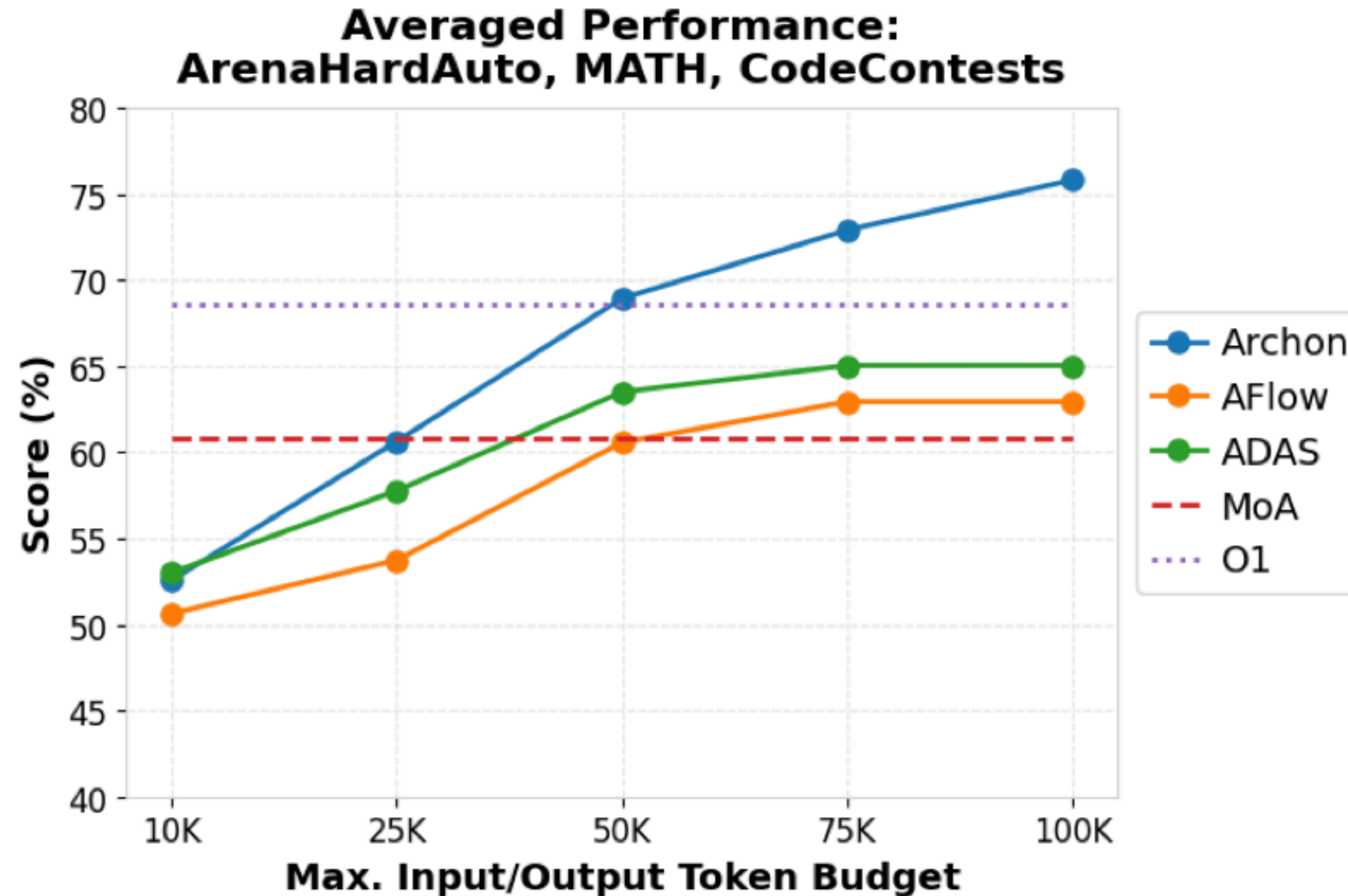
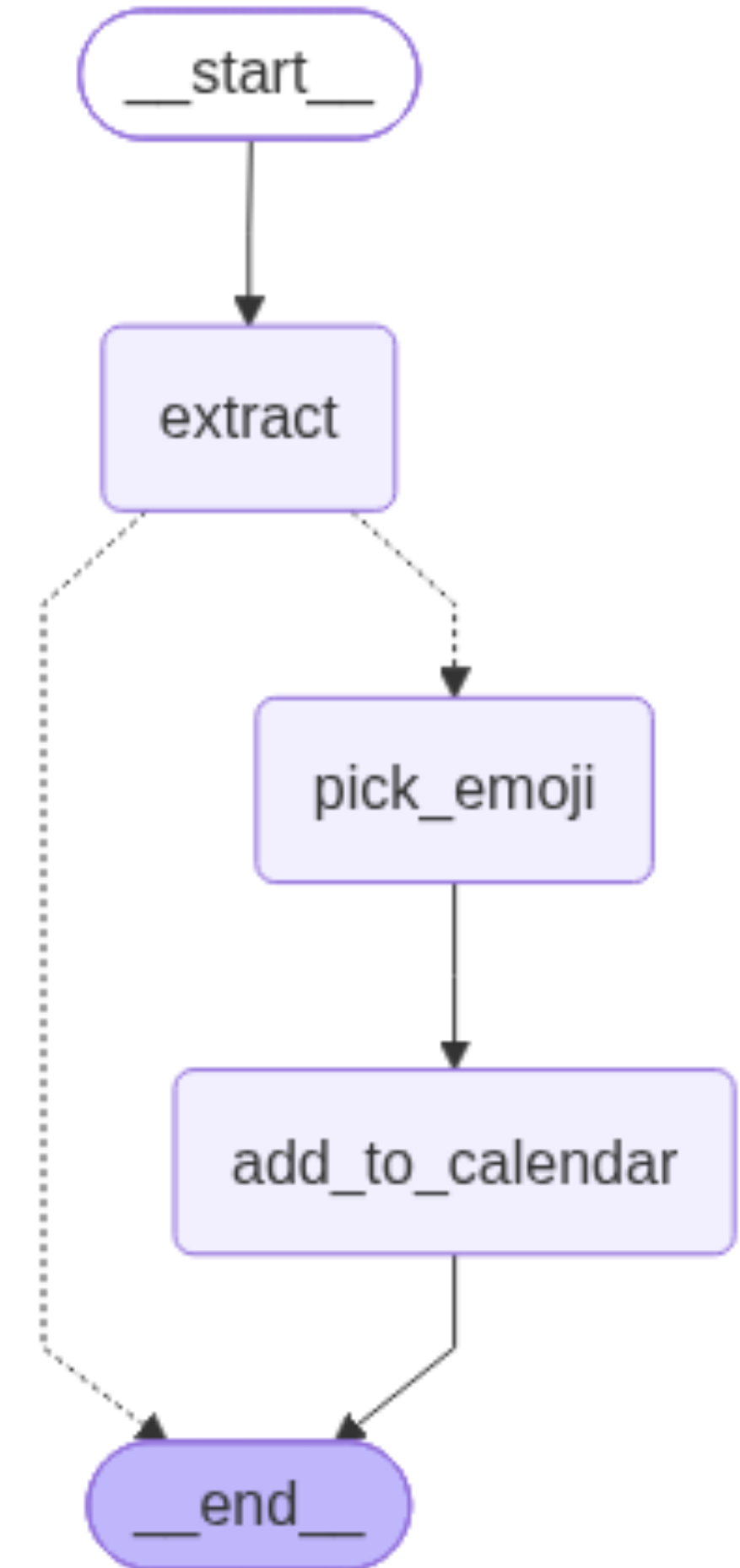


Figure 1: ARCHON's Performance Effectively Scales with Increasing Inference Budget. Individual dataset analysis included in [Figure 8](#).



Recap

- Workflows are simpler to reproduce and debug than agents
- We covered five common workflow patterns: prompt chaining, routing, parallelization, orchestrator-workers, and evaluator-optimizer
- In practice real workflows are often larger compositions of the individual components



Agent Design Patterns: Reasoning/Acting

Workflows vs Agents

Workflow Designs

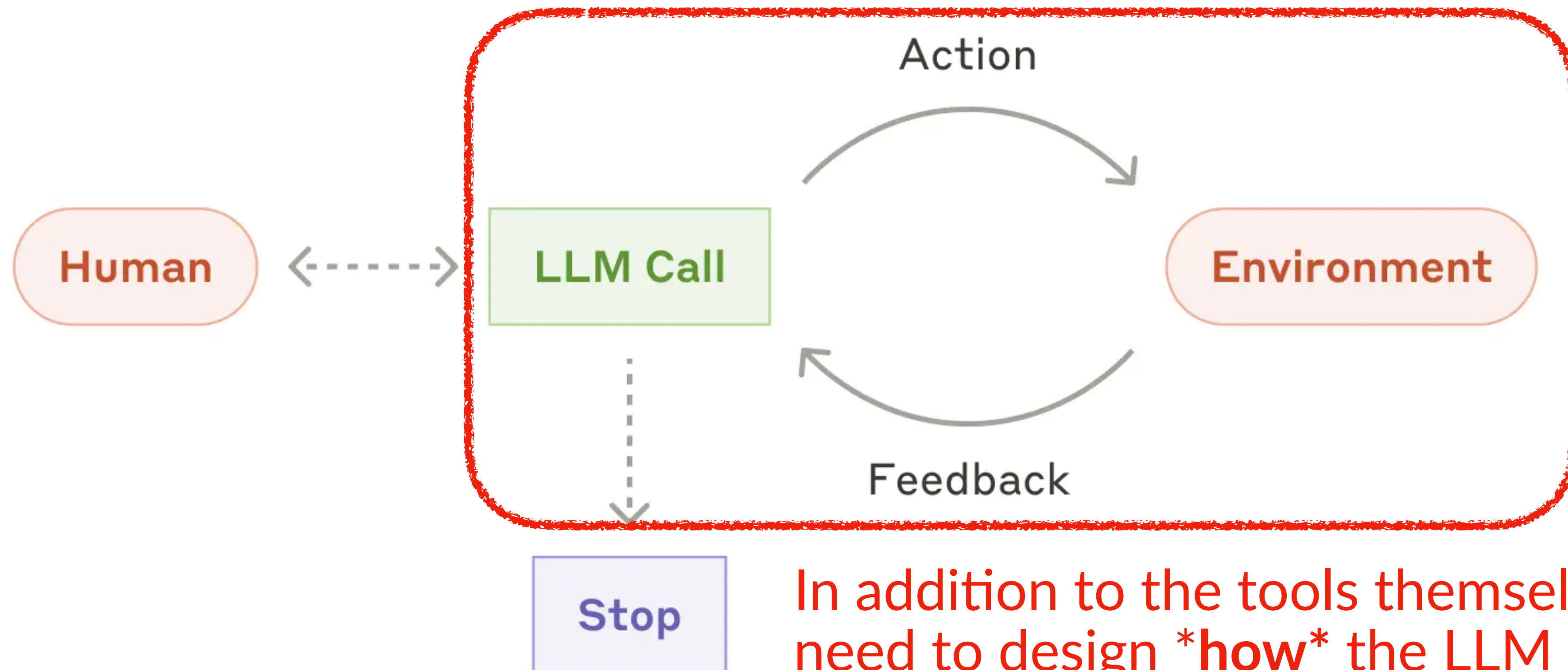
Reasoning Designs

Context Designs

Case Study



Agents



In addition to the tools themselves we need to design ***how*** the LLM reasons and interacts with the environment.



ReAct (2023)

Key Insight: reason before taking actions



ReAct: Reason + Act

(1) Hotspot QA

Question: Aside from the Apple Remote, what other device can control the program Apple Remote was originally designed to interact with?

(1a) Standard

Answer: iPod



(1b) CoT (Reason Only)

Thought: Let's think step by step. Apple Remote was originally designed to interact with Apple TV. Apple TV can be controlled by iPhone, iPad, and iPod Touch. So the answer is iPhone, iPad, and iPod Touch.

Answer: iPhone, iPad, iPod Touch



(1c) Act-Only

Act 1: Search[Apple Remote]

Obs 1: The Apple Remote is a remote control ...

Act 2: Search[Front Row]

Obs 2: Could not find [Front Row]. Similar: ...

Act 3: Search[Front Row (software)]

Obs 3: Front Row is a discontinued media center software ...

Act 4: Finish[yes]



(1d) ReAct (Reason + Act)

Thought 1: I need to search Apple Remote and find the program it was originally designed to interact with.

Act 1: Search[Apple Remote]

Obs 1: The Apple Remote is a remote control introduced in October 2005 by Apple ... originally designed to control the Front Row media center program ...

Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.

Act 2: Search[Front Row]

Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports', 'Front Row (software)', ...]

Thought 3: Front Row is not found. I need to search Front Row (software) .

Act 3: Search[Front Row (software)]

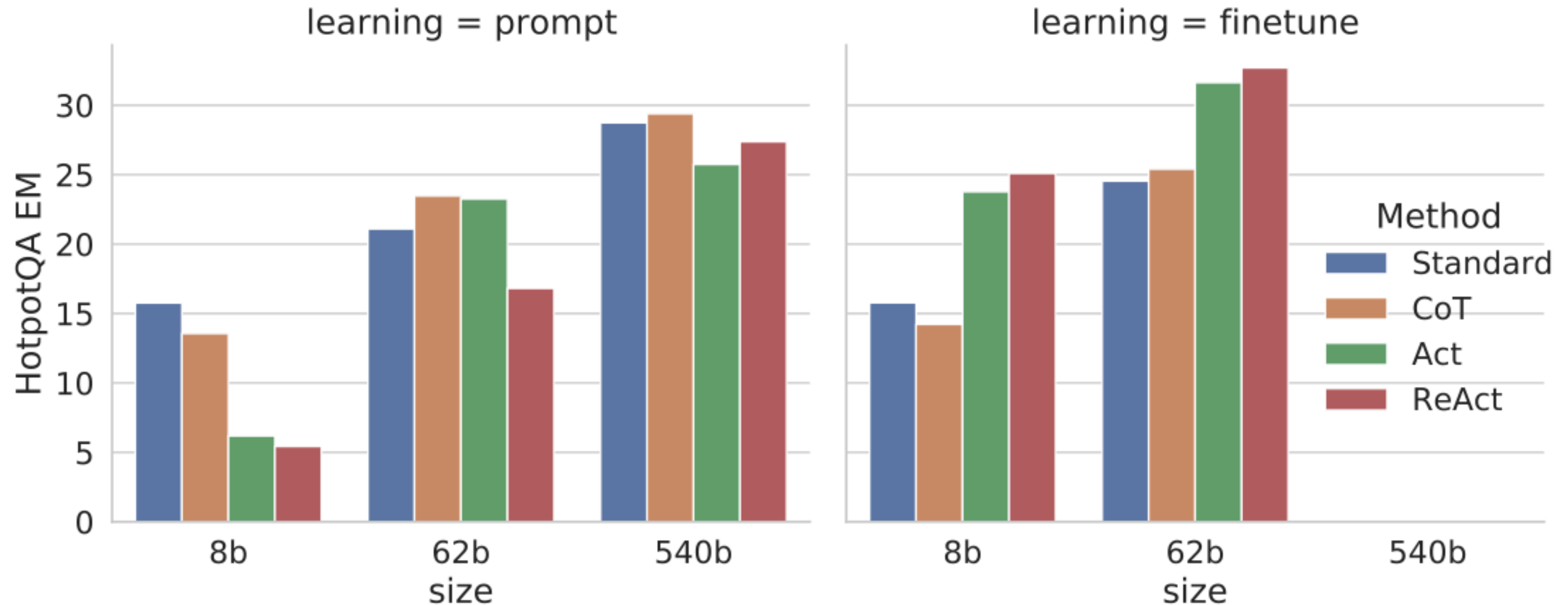
Obs 3: Front Row is a discontinued media center software ...

Thought 4: Front Row (software) is controlled by an Apple Remote or the keyboard function keys. So the answer is keyboard function keys.

Act 4: Finish[keyboard function keys]



ReAct: Finetuning for Agents



Interestingly, the original paper found that you need to fine-tune for best performance. Now fine-tuning models to act as agents is common practice during post-training.



ReAct: Importance of Reasoning

Prompt Method ^a	HotpotQA (EM)	Fever (Acc)
Act	25.7	58.9
ReAct	27.4	60.9

Table 1: PaLM-540B prompting results on HotpotQA and Fever.

- Useful to reason before taking actions.
- Including thoughts before actions is still very common today.

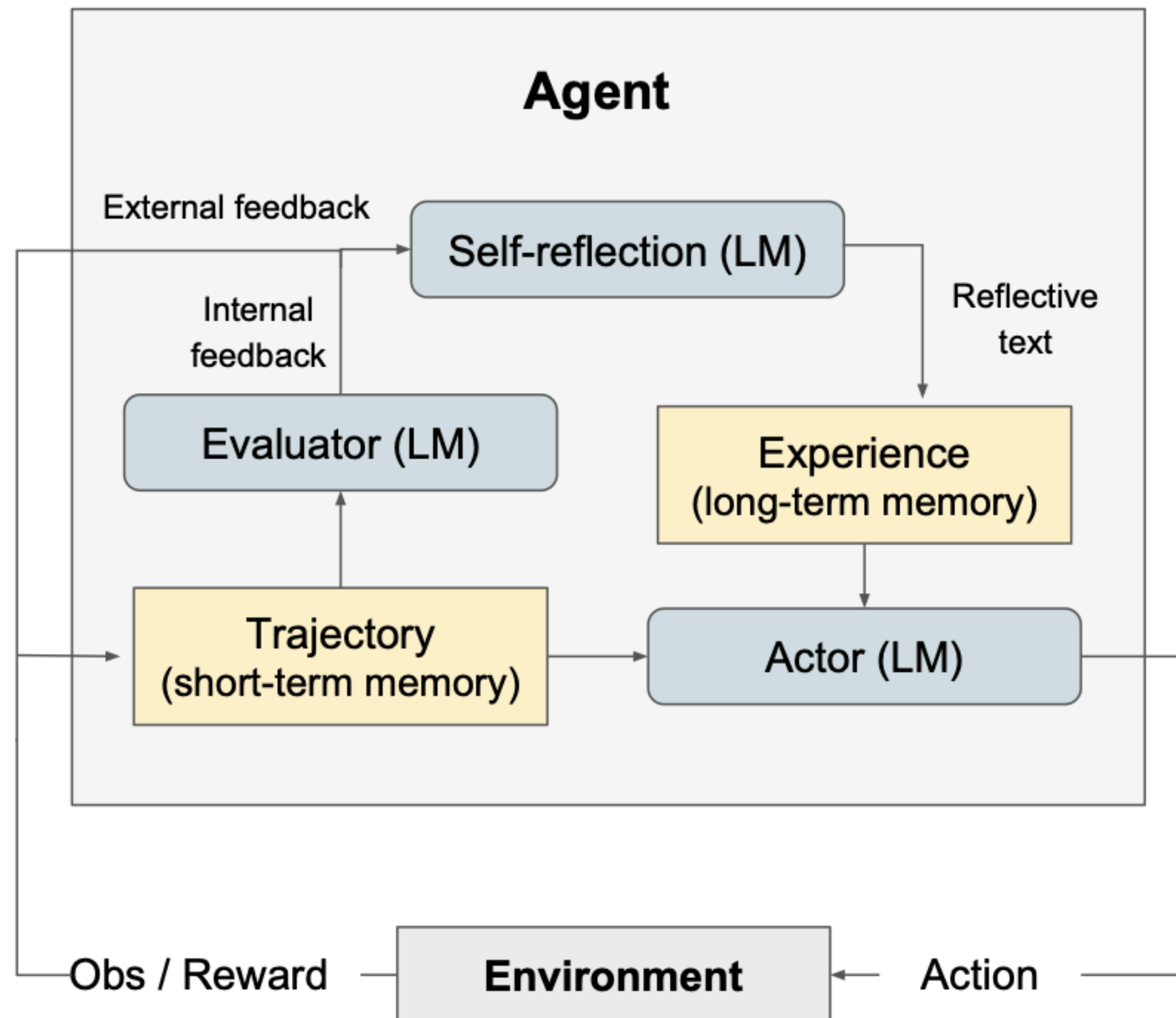


Reflexion (2023)

Key Insight: self-reflection is useful



Reflexion: Algorithm



Algorithm 1 Reinforcement via self-reflection

Initialize Actor, Evaluator, Self-Reflection:

M_a, M_e, M_{sr}

Initialize policy $\pi_\theta(a_i|s_i), \theta = \{M_a, mem\}$

Generate initial trajectory using π_θ

Evaluate τ_0 using M_e

Generate initial self-reflection sr_0 using M_{sr}

Set $mem \leftarrow [sr_0]$

Set $t = 0$

while M_e not pass or $t < \text{max trials}$ **do**

 Generate $\tau_t = [a_0, o_0, \dots a_i, o_i]$ using π_θ

 Evaluate τ_t using M_e

 Generate self-reflection sr_t using M_{sr}

 Append sr_t to mem

 Increment t

end while

return

Key Insight: Turn Evaluations/Reflections into “Memory” (context) for the actor LLM.



Reflexion: Example Trajectories

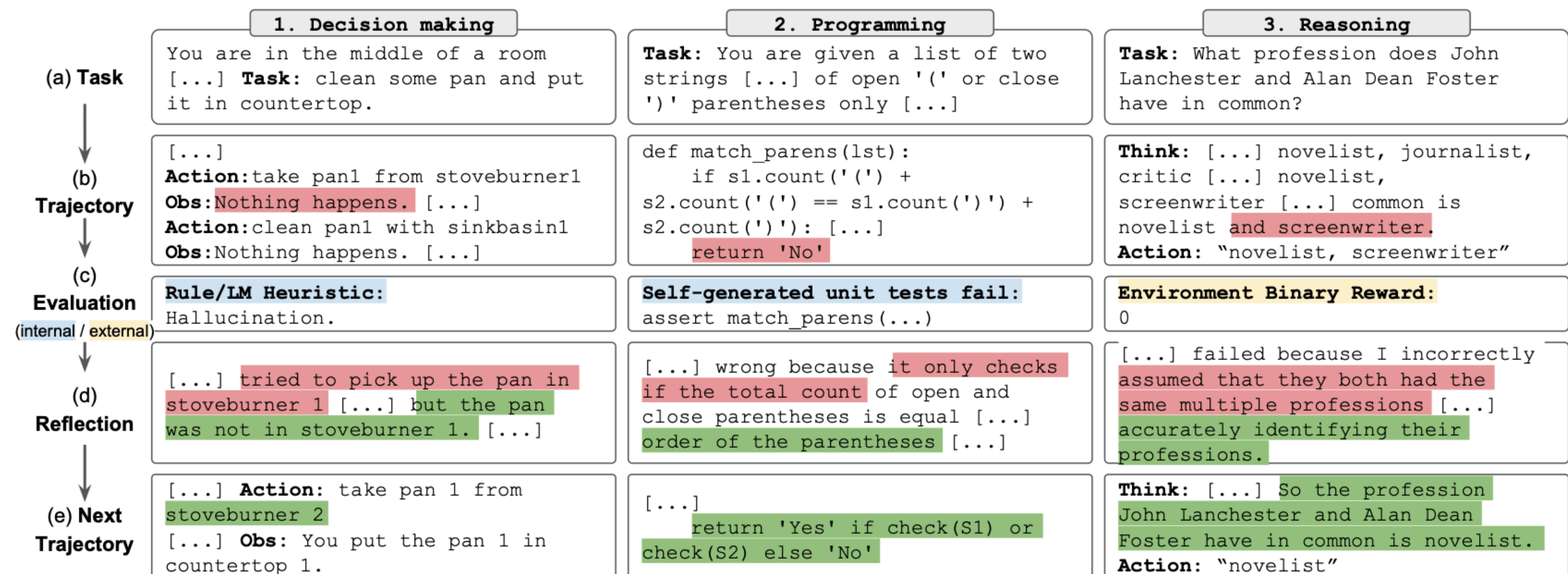
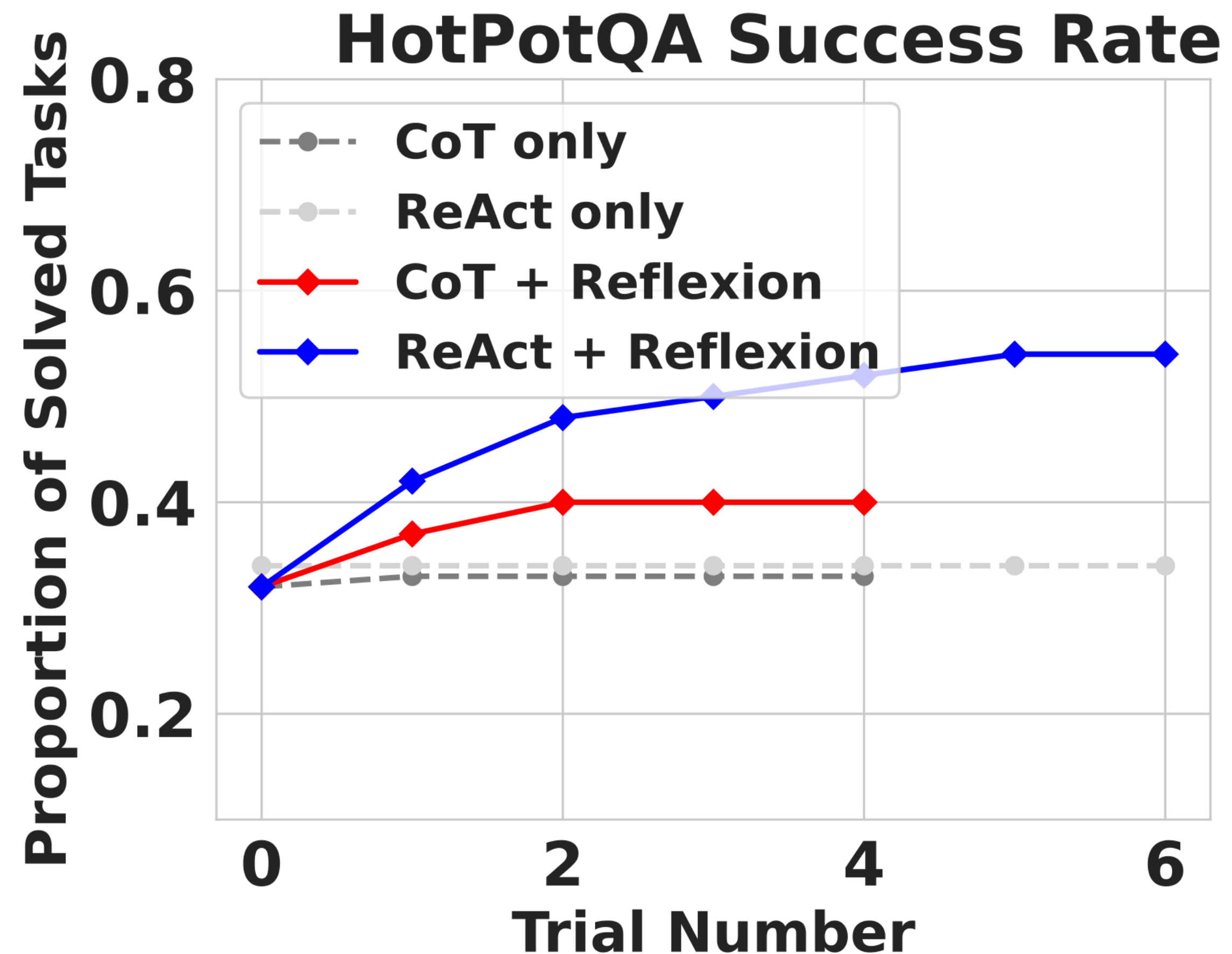


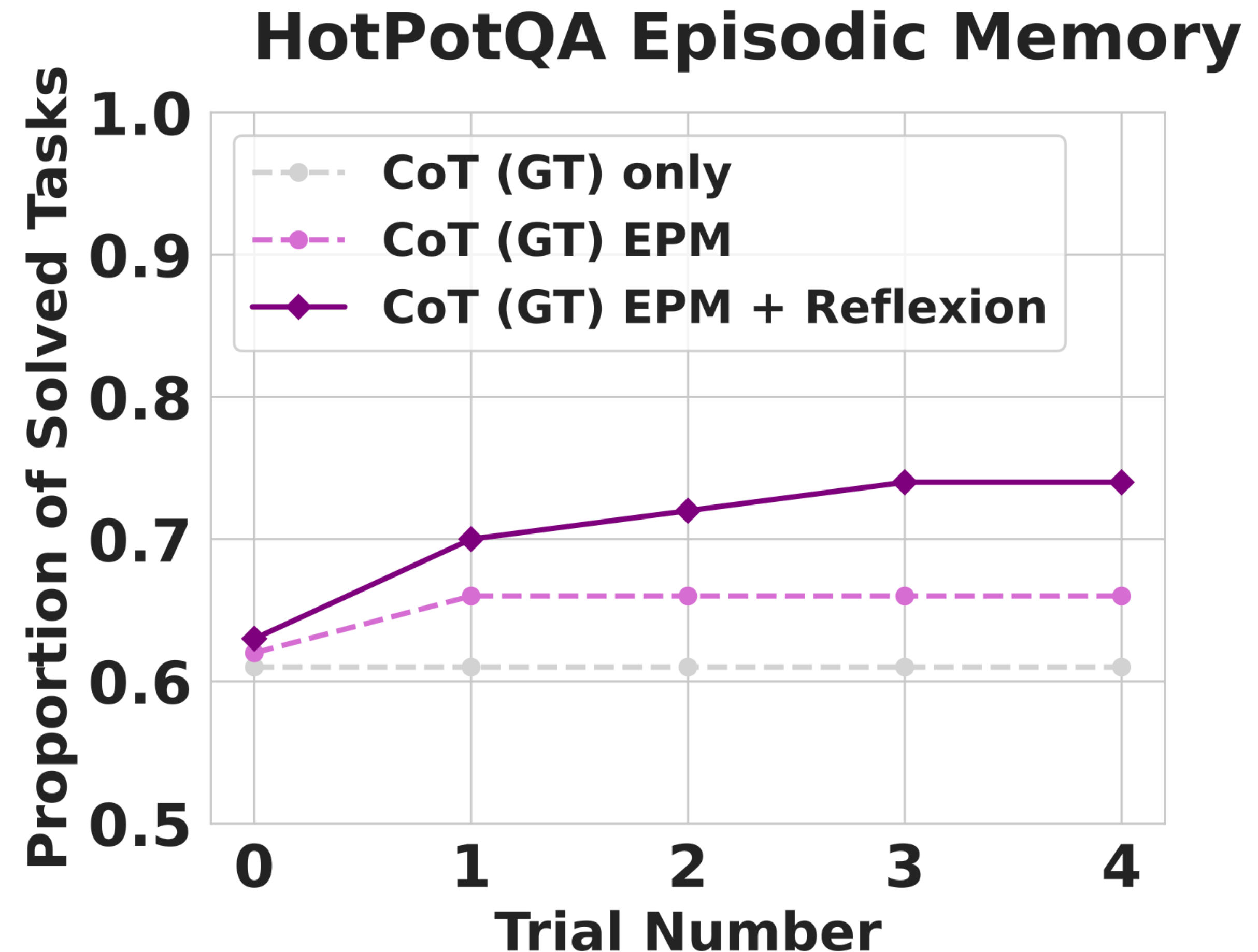
Figure 1: Reflexion works on decision-making 4.1, programming 4.3, and reasoning 4.2 tasks.



Reflexion



Reflexion improves accuracy over trials



It's the LLM reflection that helps, not just the past trajectories

Limitations of Self-Reflection

LARGE LANGUAGE MODELS CANNOT SELF-CORRECT REASONING YET

Jie Huang^{1,2*} Xinyun Chen^{1*} Swaroop Mishra¹ Huaixiu Steven Zheng¹ Adams Wei Yu¹
Xinying Song¹ Denny Zhou¹

¹Google DeepMind ²University of Illinois at Urbana-Champaign

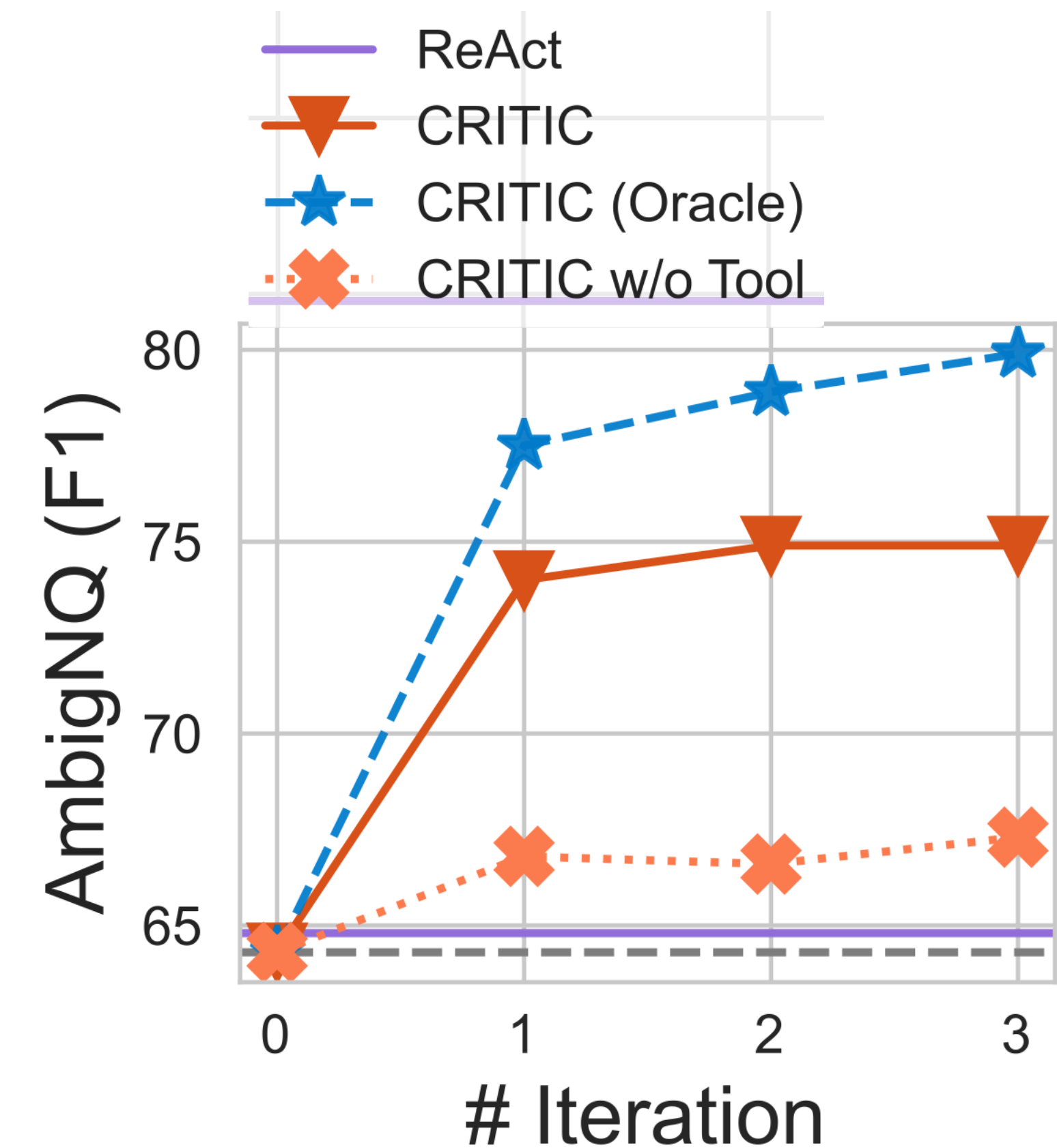
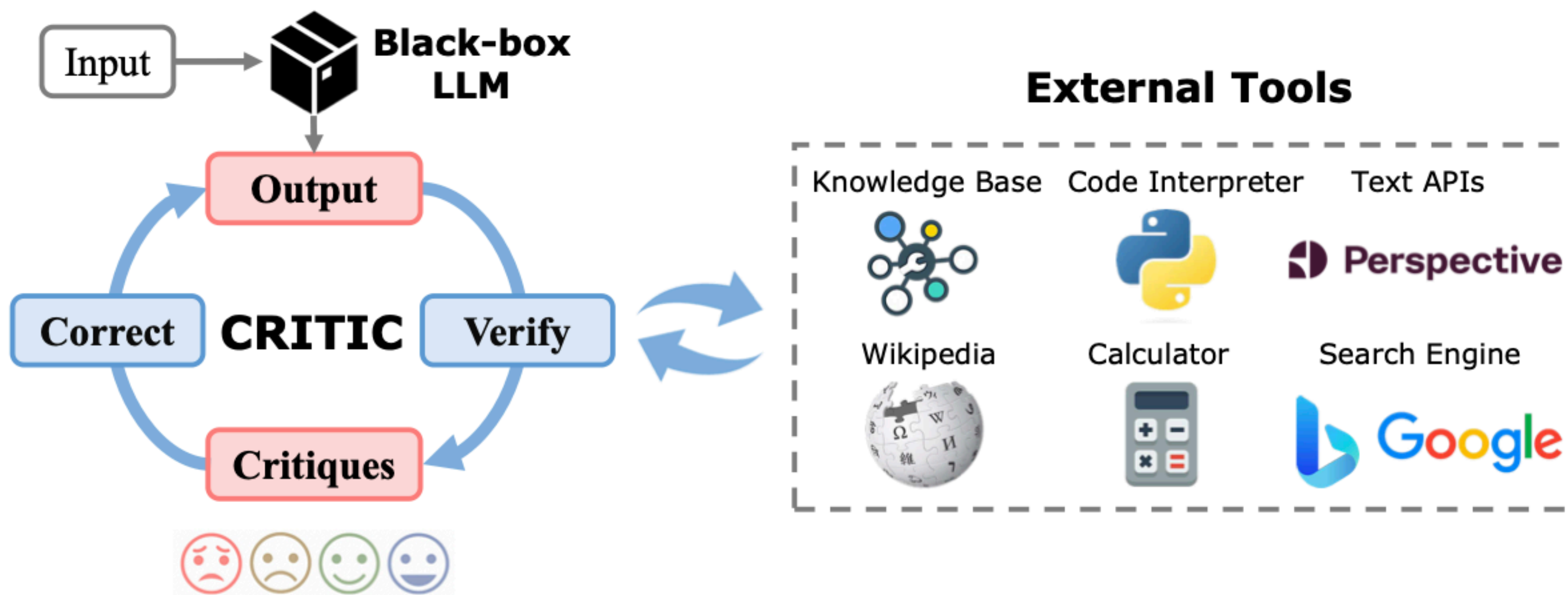
Table 5: Results of GPT-4-Turbo with different feedback prompts.

	# calls	GSM8K	CommonSenseQA
Standard Prompting	1	91.5	84.0
<i>Feedback Prompt:</i> Assume that this answer could be either correct or incorrect. Review the answer carefully and report any serious problems you find.			
Self-Correct (round 1)	3	88.0	81.5
Self-Correct (round 2)	5	90.0	83.0
<i>Feedback Prompt:</i> Review your previous answer and determine whether it's correct. If wrong, find the problems with your answer.			
Self-Correct (round 1)	3	90.0	74.5
Self-Correct (round 2)	5	90.0	81.0
<i>Feedback Prompt:</i> Verify whether your answer is correct, and provide an explanation.			
Self-Correct (round 1)	3	91.0	81.5
Self-Correct (round 2)	5	91.0	83.5

- When asking an LLM to evaluate its own responses it leads to **decreased** performance.
- Reflexion gets around this by using external environment feedback.
- This finding pre-dates reasoning models, which show impressive ability to self-correct during reasoning.
- Today self-reflection is typically built into the LLM reasoning itself rather than explicitly enforced.



CRITIC (2024)



Adding External Tools to the Verifier is useful! (Make an agentic verifier)



CodeAct (2024)

Key Insight: code is the right form factor for actions



JSON Actions

Instruction:

Determine the most cost-effective country to purchase the smartphone model “CodeAct1”. The countries to consider are the USA, Japan, Germany, and India.

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.

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`

Code Actions

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.

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("CodeAct 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)
```

Control & Data Flow of Code
Simplifies Complex Operations

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



Environment

Japan 904.0



Response

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

Available APIs

[1] lookup_rates(country: str) -> (float, float)

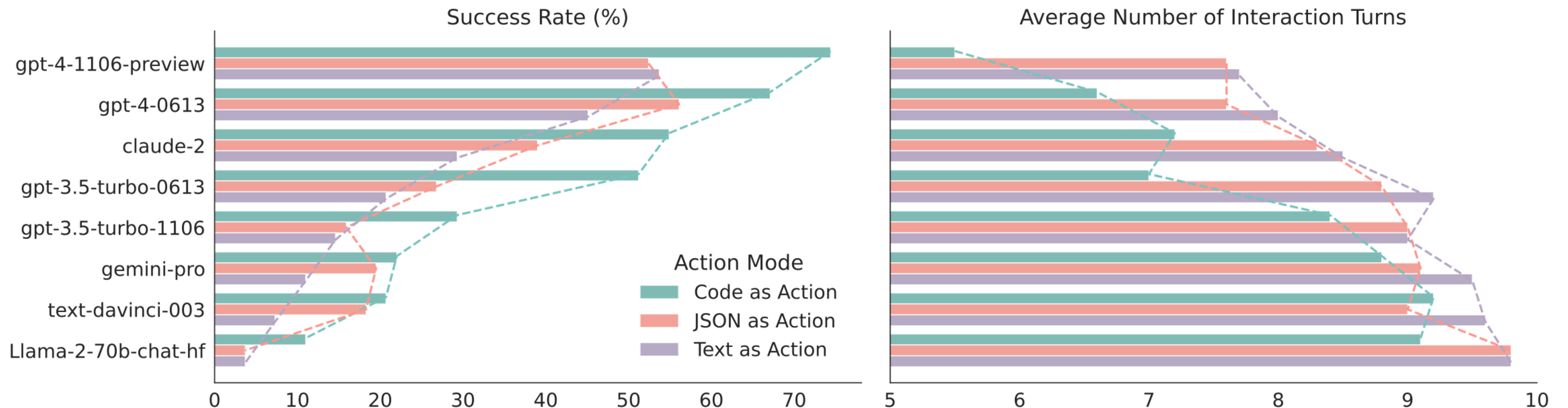
[2] convert_and_tax(price: float, exchange_rate: float, tax_rate: float) -> float

[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

CodeAct: More Accurate and More Efficient



Code as Action has a higher success rate and accomplishes tasks in fewer turns.

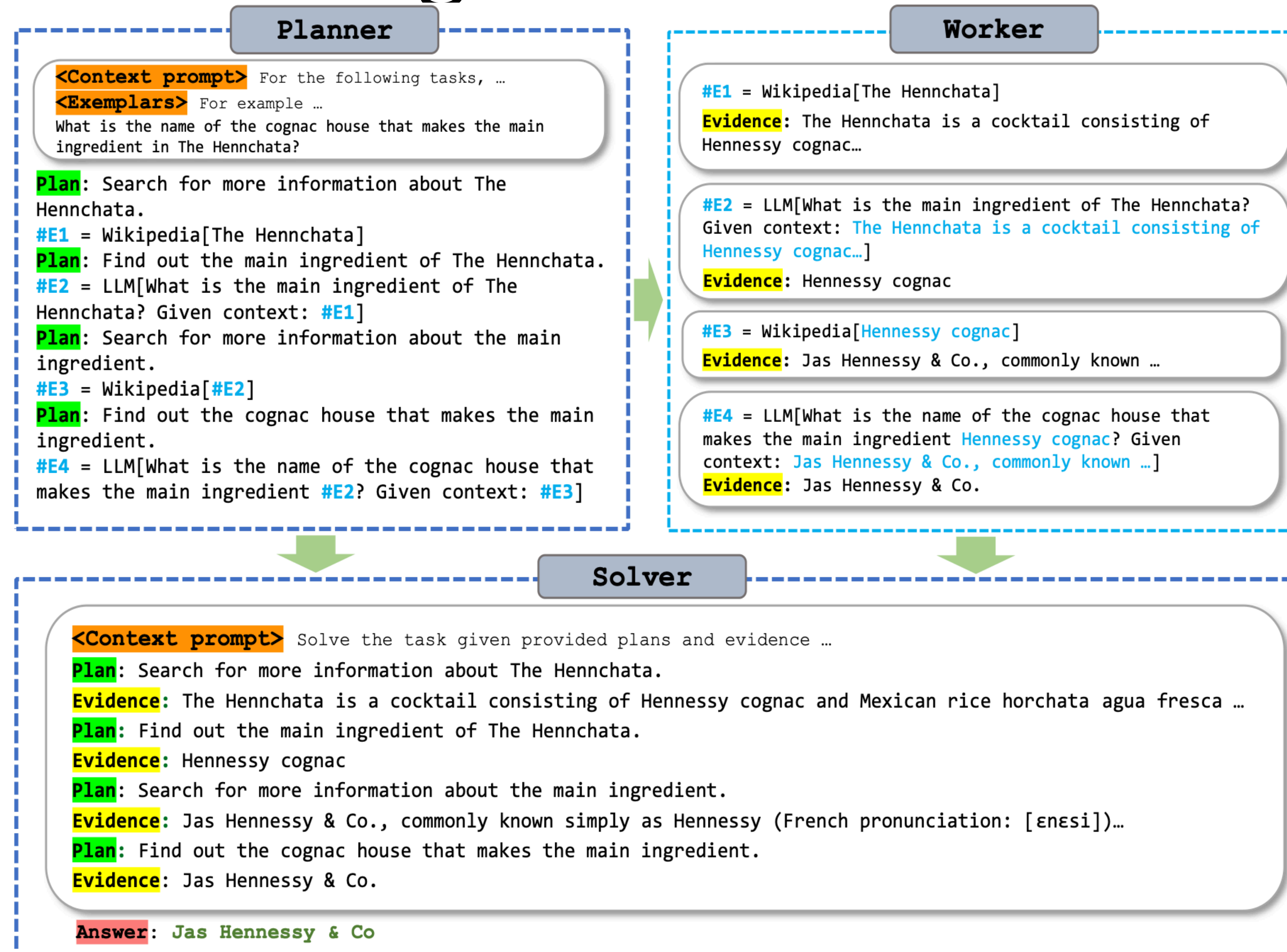


Planning then executing

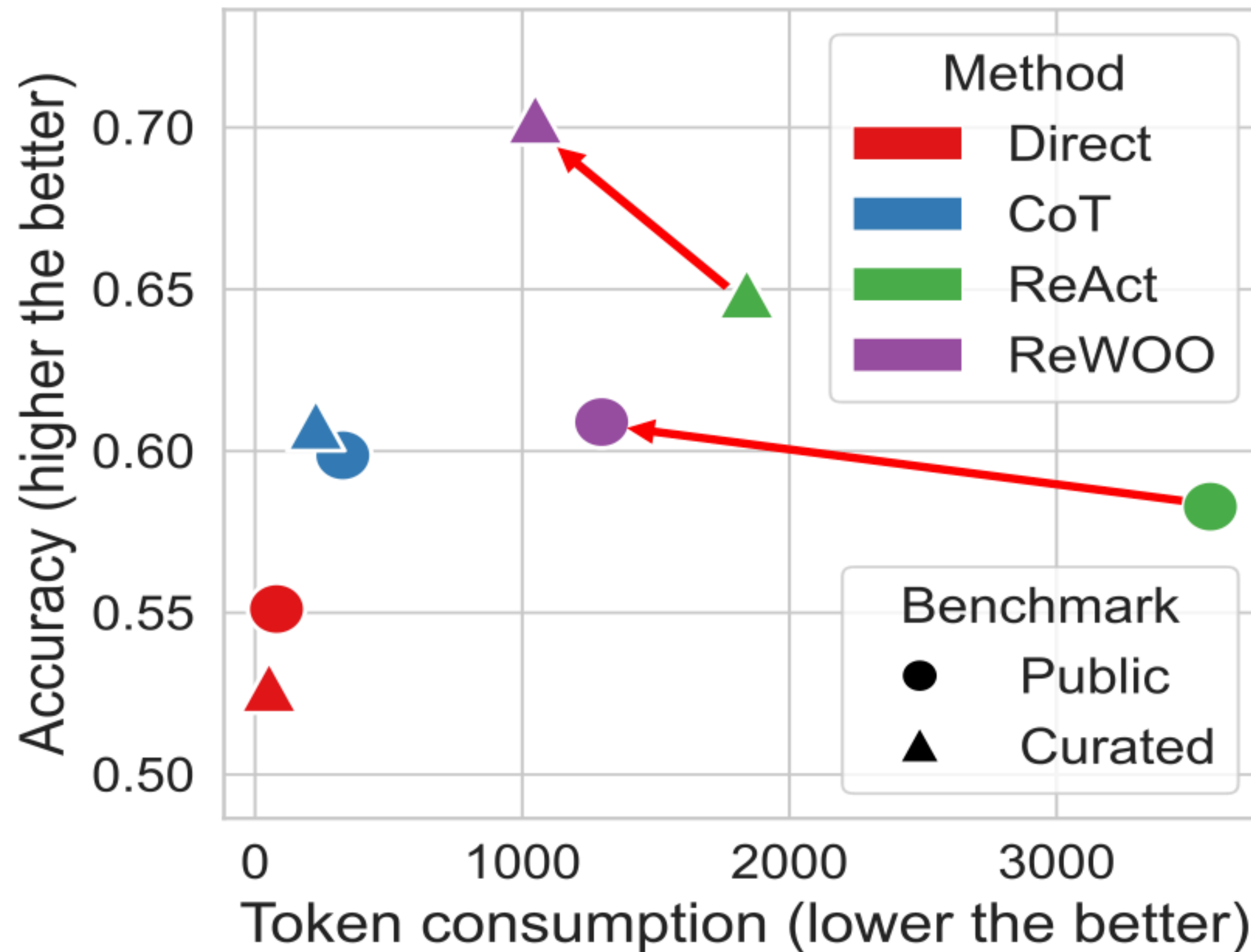
Key Insight: planning enables turning agent action execution back into workflows



ReWOO: Reasoning WithOut Observation (2023)



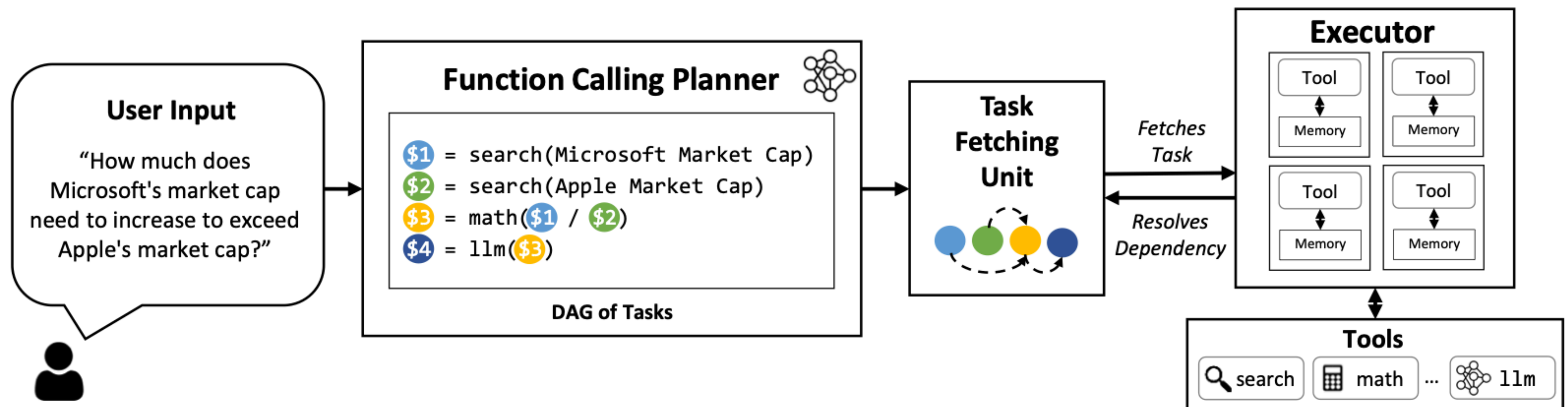
ReWOO



Planning, then executing enables higher accuracy with fewer tokens



LLMCompiler (2024)



Differences from ReWOO:

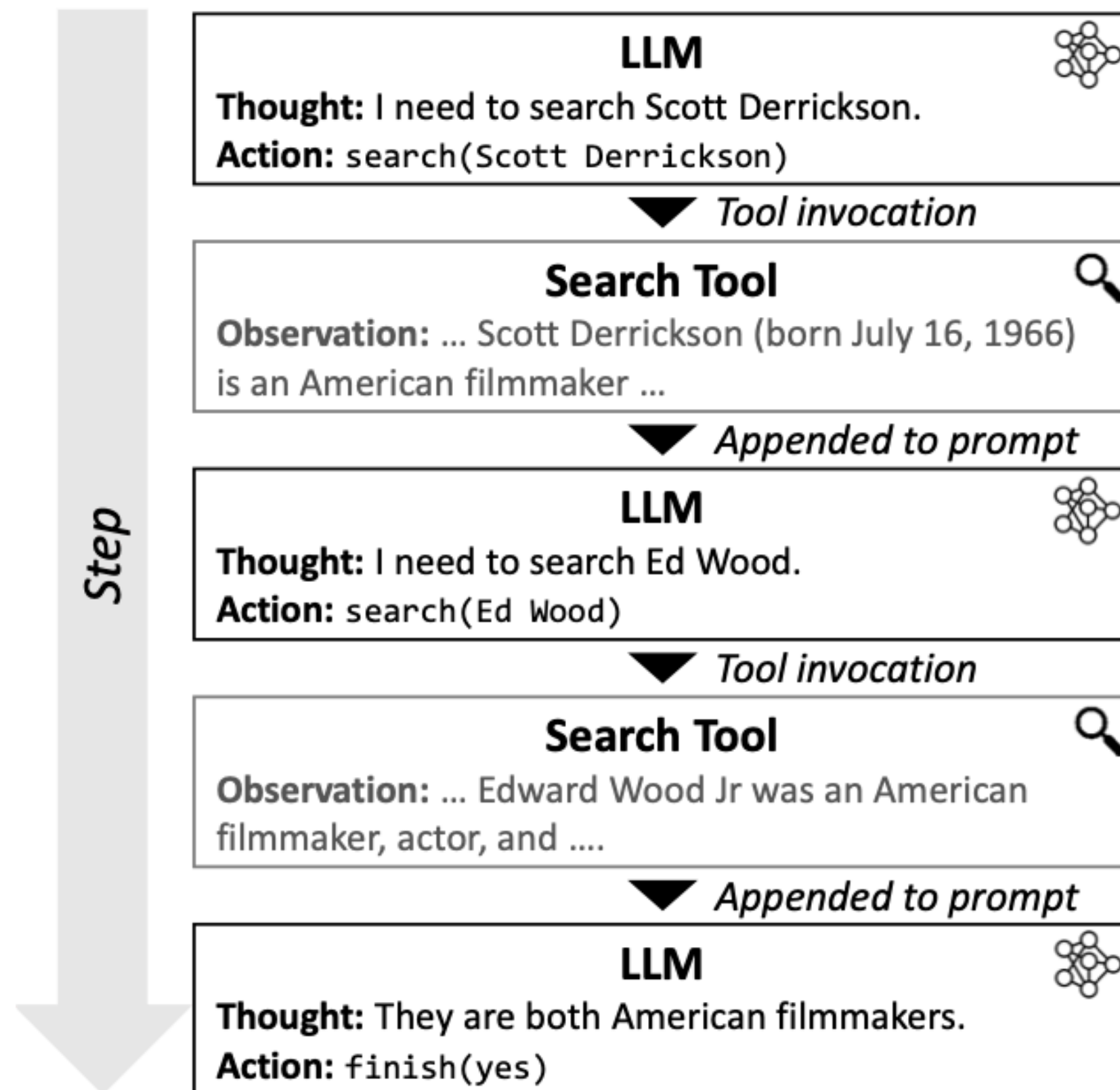
Parallel Execution and Dynamic Replanning



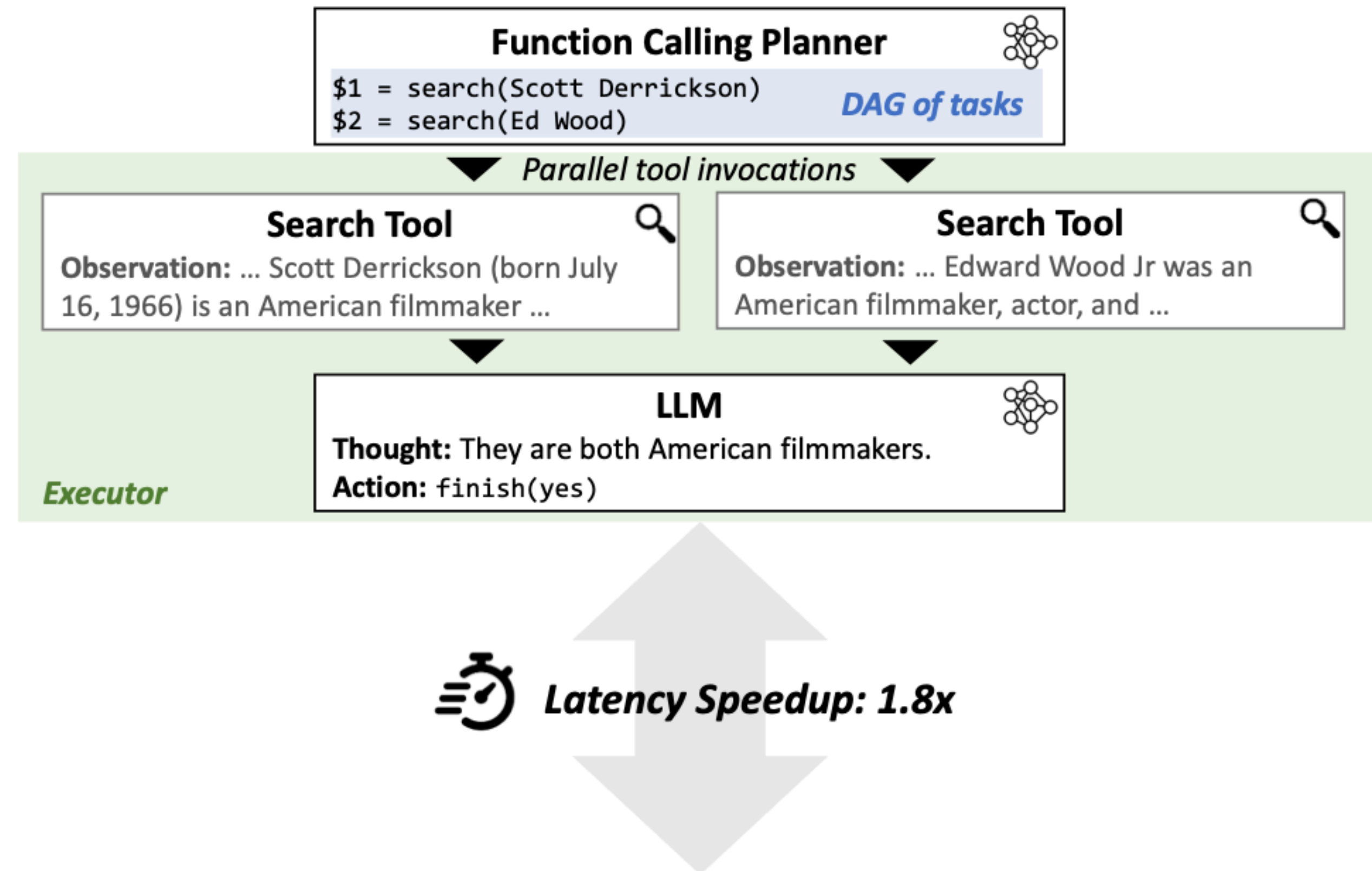
LLMCompiler

Question: Were Scott Derrickson and Ed Wood of the same nationality?

ReAct



LLMCompiler



LLMCompiler

Benchmark	Method	GPT (Closed-source)			LLaMA-2 70B (Open-source)		
		Accuracy (%)	Latency (s)	Speedup	Accuracy (%)	Latency (s)	Speedup
HotpotQA	ReAct	61.52	-	-	54.74	-	-
	ReAct [†]	62.47	7.12	1.00×	54.40	13.44	1.00×
	OAI Parallel Function	62.05	4.42	1.61×	-	-	-
	LLMCompiler	62.00	3.95	1.80×	57.83	9.58	1.40×
Movie Rec.	ReAct	68.60	-	-	70.00	-	-
	ReAct [†]	72.47	20.47	1.00×	70.60	33.37	1.00×
	OAI Parallel Function	77.00	7.42	2.76×	-	-	-
	LLMCompiler	77.13	5.47	3.74×	77.80	11.83	2.82×
ParallelQA	ReAct	89.09	35.90	1.00×	59.59	15.47	1.00×
	OAI Parallel Function	87.32	19.29	1.86×	-	-	-
	LLMCompiler	89.38	16.69	2.15×	68.14	26.20	2.27×
Game of 24	Tree-of-Thoughts	74.00	241.2	1.00×	30.00	952.06	1.00×
	LLMCompiler	75.33	83.6	2.89×	32.00	456.02	2.09×

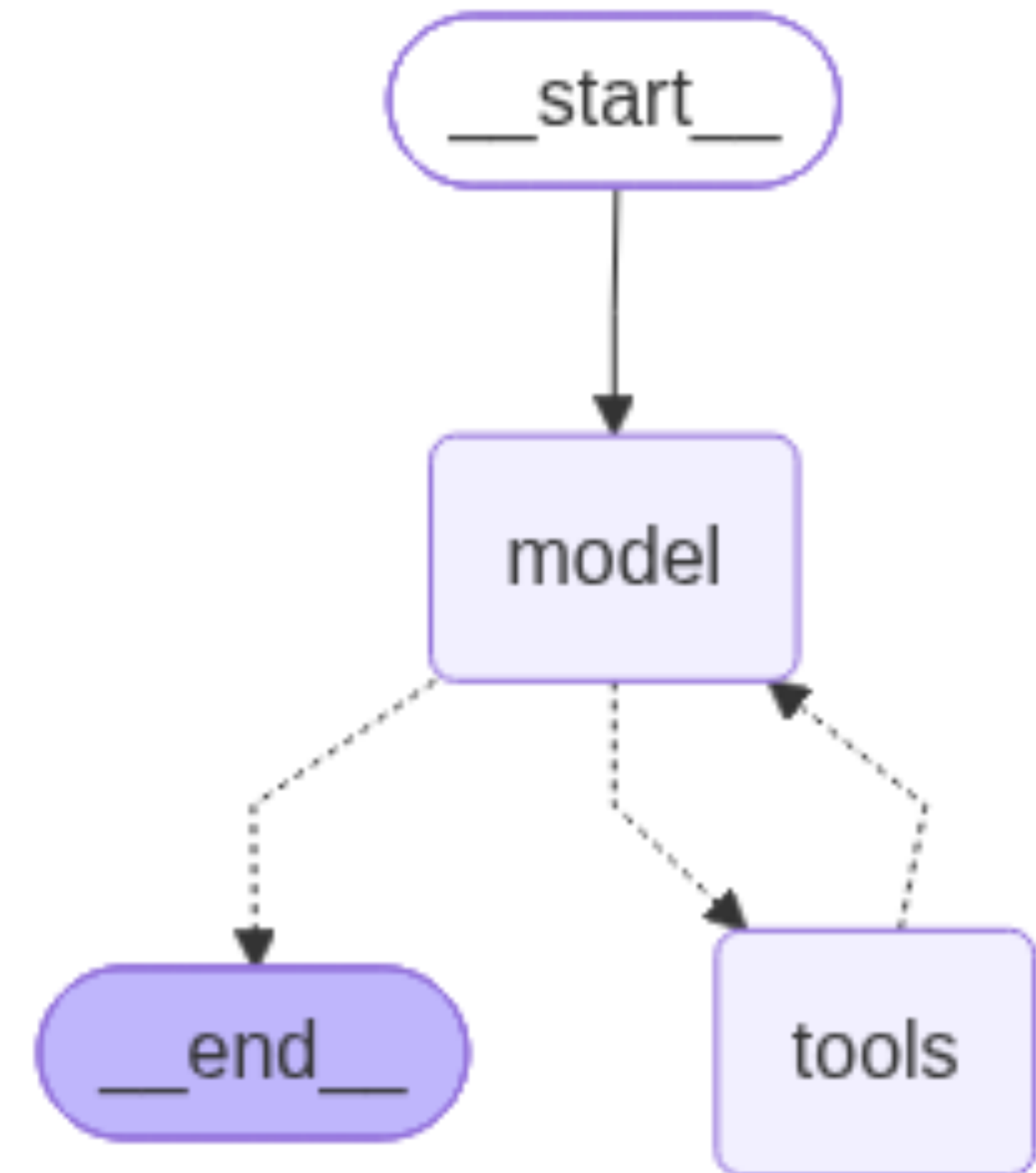


Recap



Recap

- **ReAct**: Reasoning before acting is useful.
- **Reflexion**: Self-reflection is incorporated into the reasoning today.
- **CodeAct**: Code is a good form factor for actions.
- **ReWOO/LLMCompiler**: Compiling workflows is primarily done with parallel tool calls today. Planning is still important.
- Sometimes workflows are still used for “sub-agent” orchestration.



Agent Design Patterns: Context Management

Workflows vs Agents

Workflow Designs

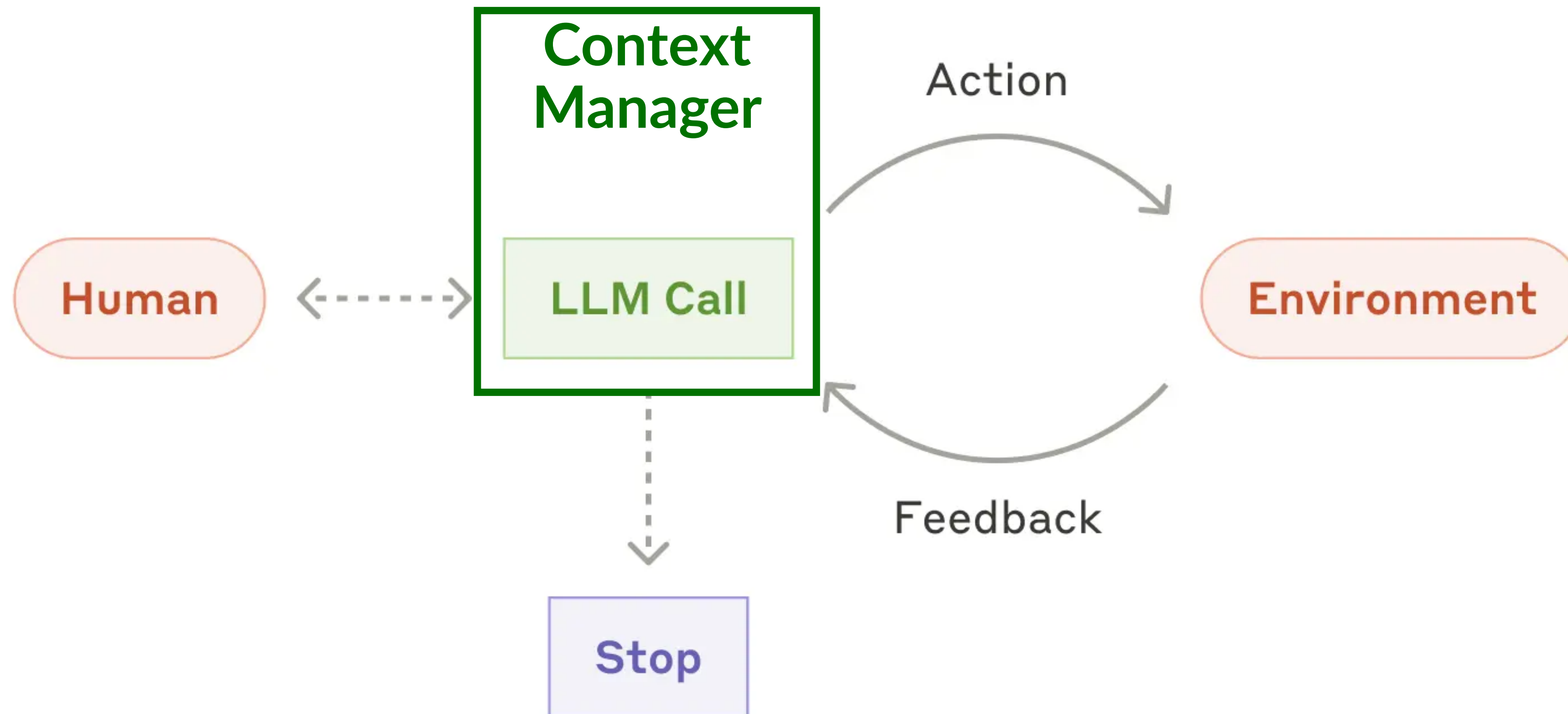
Reasoning Designs

Context Designs

Case Study



Agents: Context

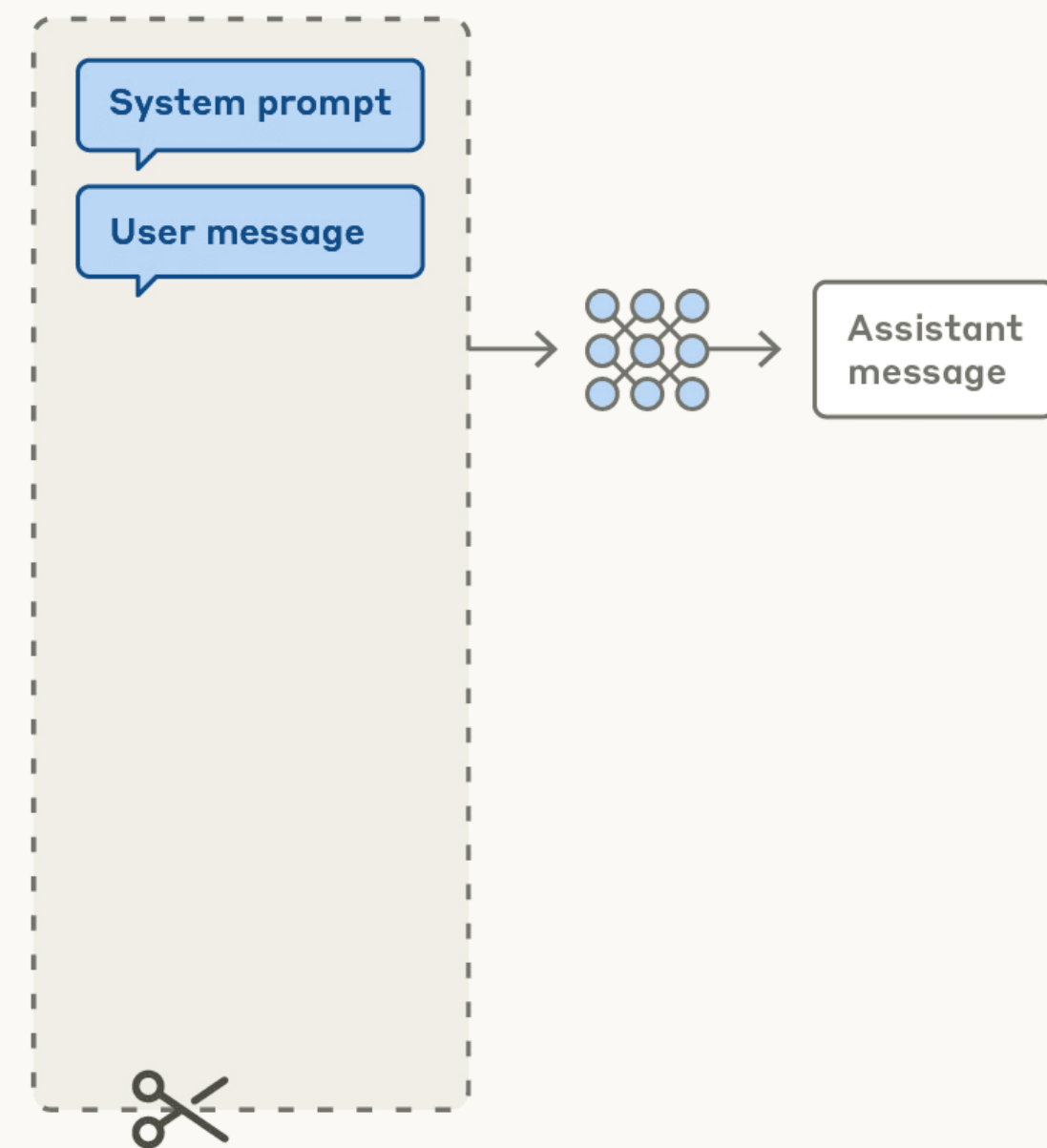


Context Engineering

Prompt engineering vs. context engineering

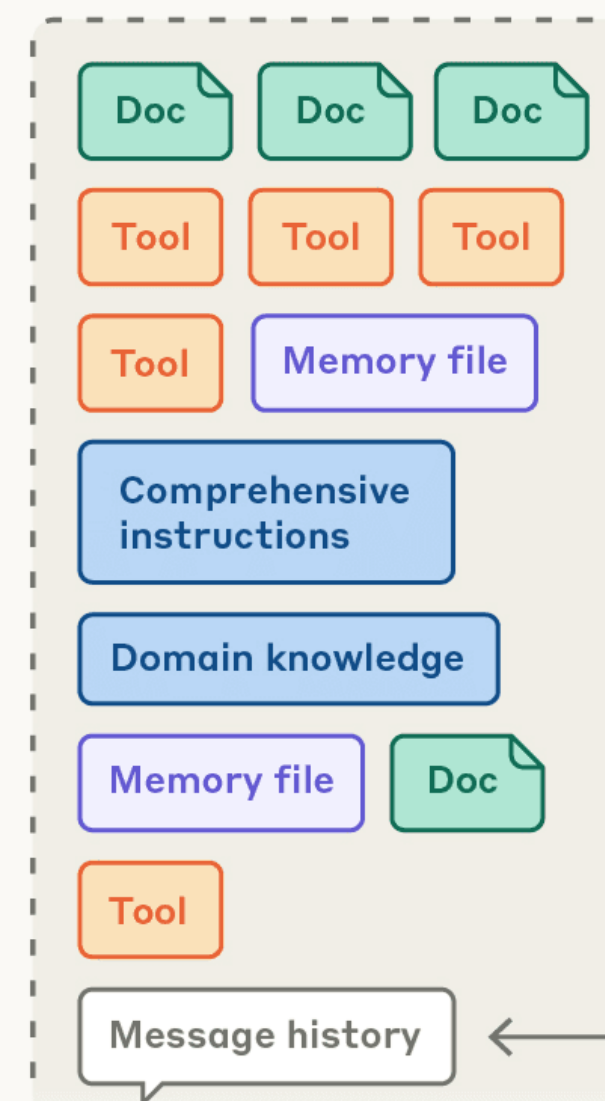
Prompt engineering for single turn queries

Context window



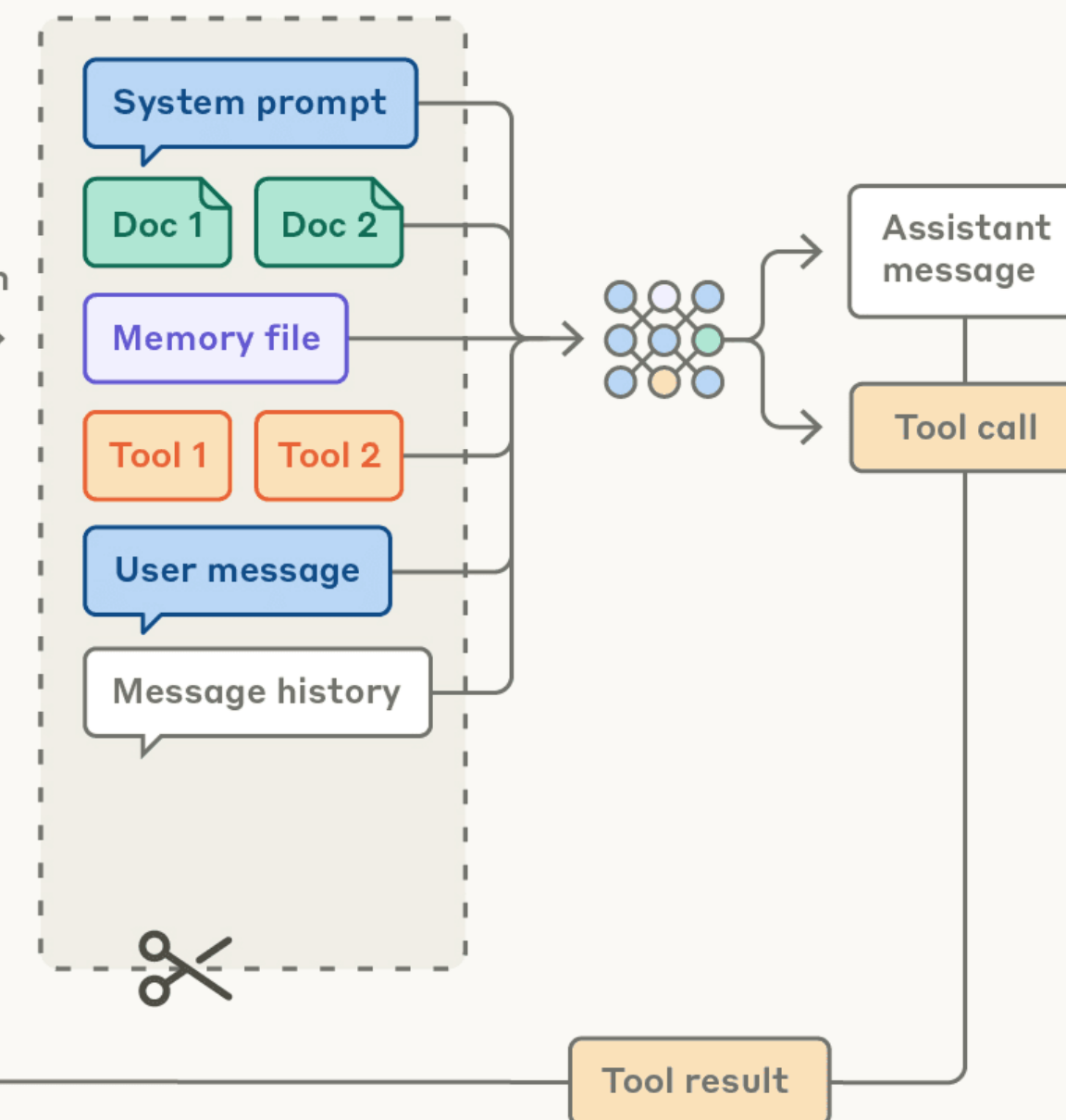
Context engineering for agents

Possible context to give model



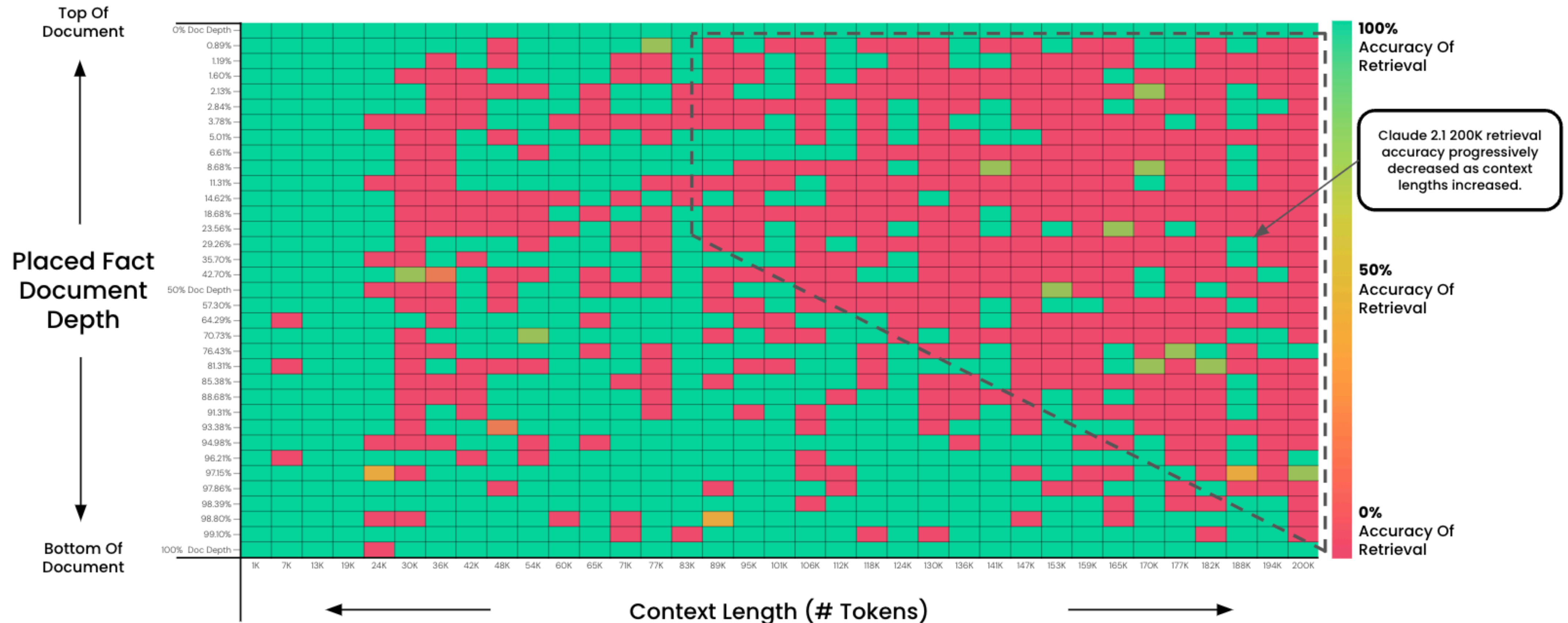
Curation

Context window



Pressure Testing Claude-2.1 200K via "Needle In A HayStack"

Asking Claude 2.1 To Do Fact Retrieval Across Context Lengths & Document Depth



Goal: Test Claude 2.1 Ability To Retrieve Information From Large Context Windows

A fact was placed within a document. Claude 2.1 (200K) was then asked to retrieve it. The output was evaluated (with GPT-4) for accuracy.

This test was run at 35 different document depths (top > bottom) and 35 different context lengths (1K > 200K tokens).

Document Depths followed a sigmoid distribution



Context Engineering: Principles

Focus the Context

- Keep only what you need for the task at hand in the context window.
- The entire history does not need to be in the context.
- Excessive information is context-bloat.

Preserve Continuity

- Manipulate the context to preserve requirements, goals, constraints, progress, etc.
- Practically this also means preserving the KV Cache



Context Management Patterns

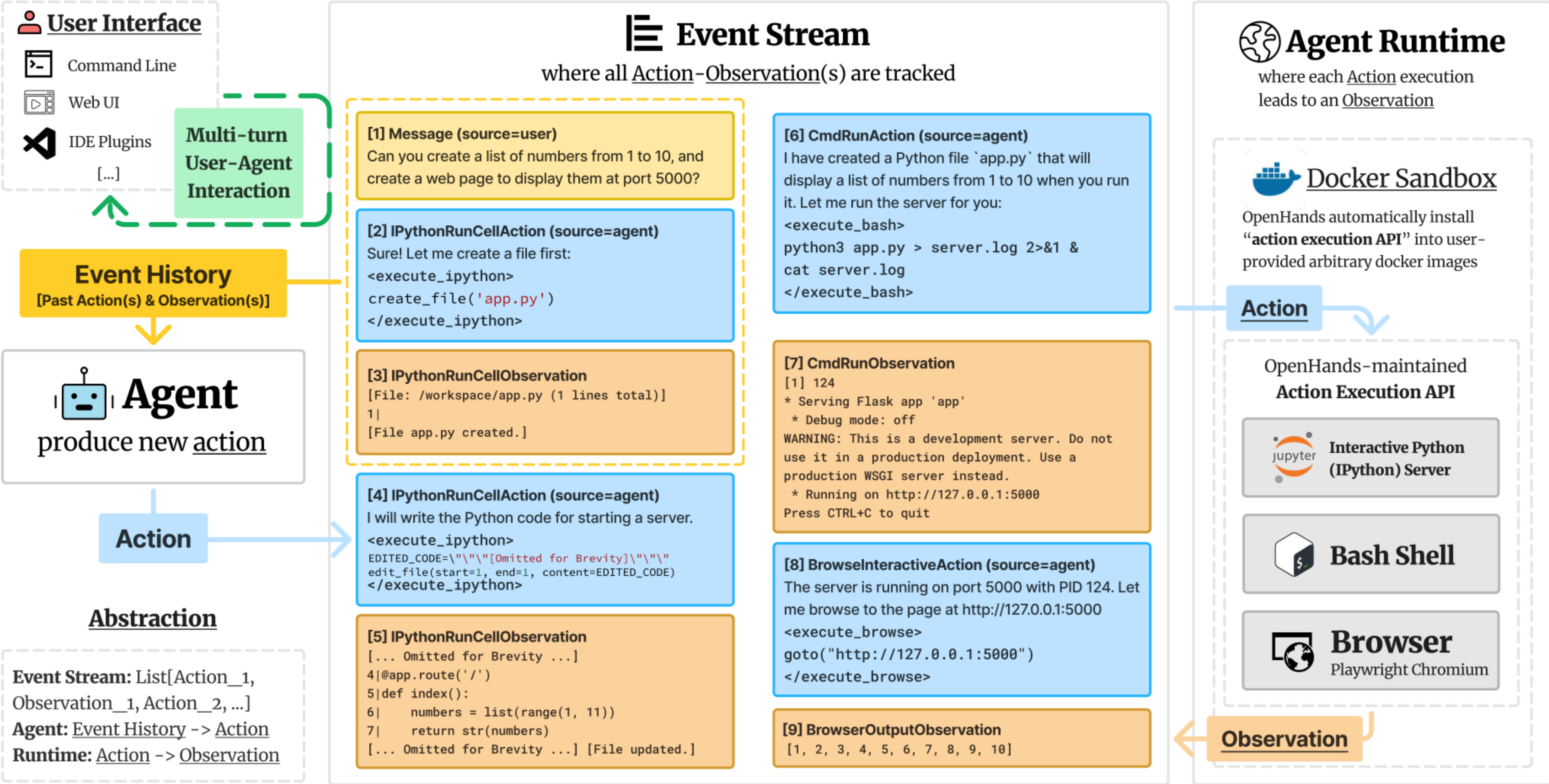


Context Patterns Overview

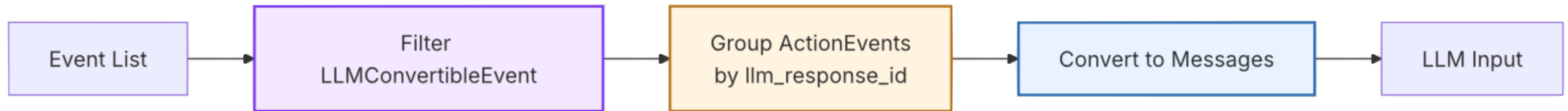
- **Persistent Event Log:** Rebuild the context from the system events
- **Memory Offloading:** Store knowledge to memory to enable pruning
- **Compaction:** Summarizing prior context into a compressed description
- **Observation Masking:** Hiding/truncating old outputs from tool calls
- **Context Isolation:** Use subagents for specific tasks, only include in the context what you actually need for the task at hand.



Persistent Event Log: OpenHands



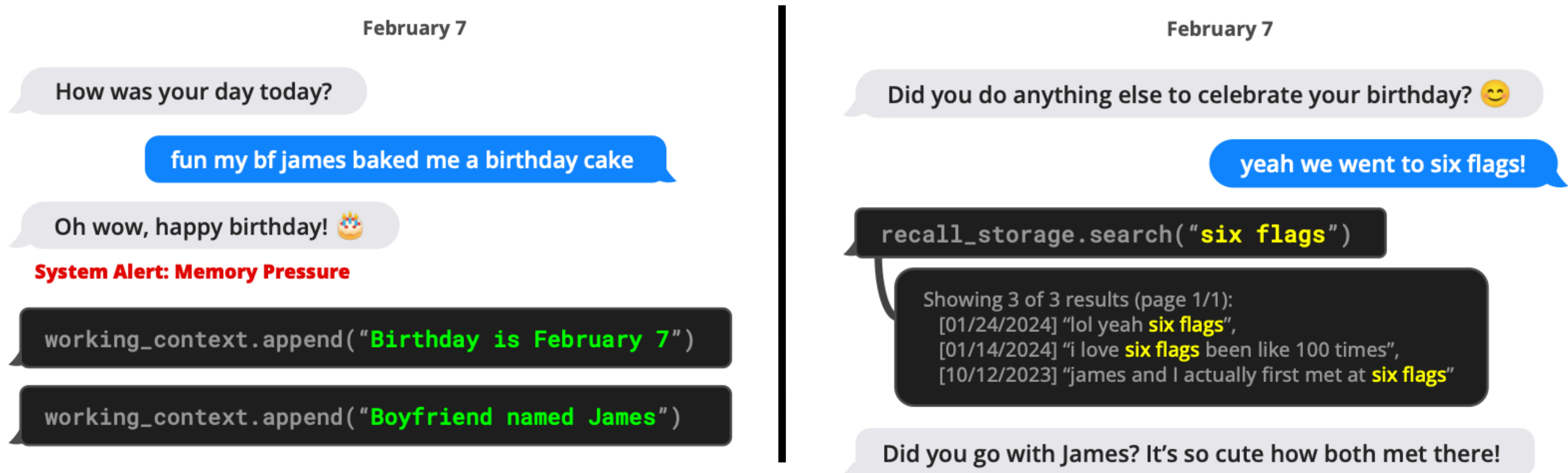
Persistent Event Log: OpenHands



- Everything gets preserved exactly as it happened even if the LLM input must be changed. The event log history can be immutable (append only).
- For every call to the LLM the input is reconstructed from the event log.
- An event log also enables hooks for external observers to monitor the system state.



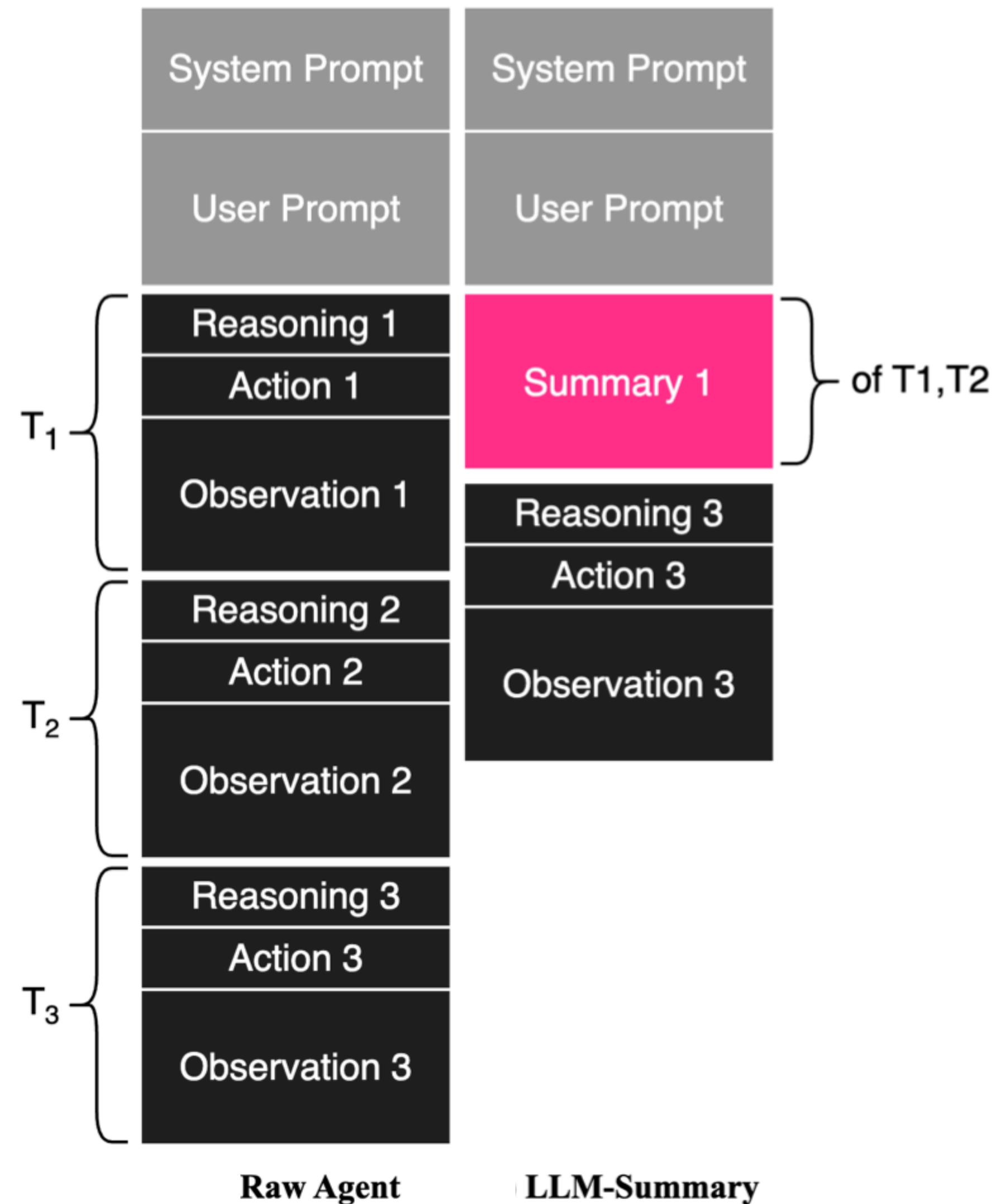
Memory Offloading: MemGPT



Storing things to memory or files frees up parts of the context to prune



Compaction

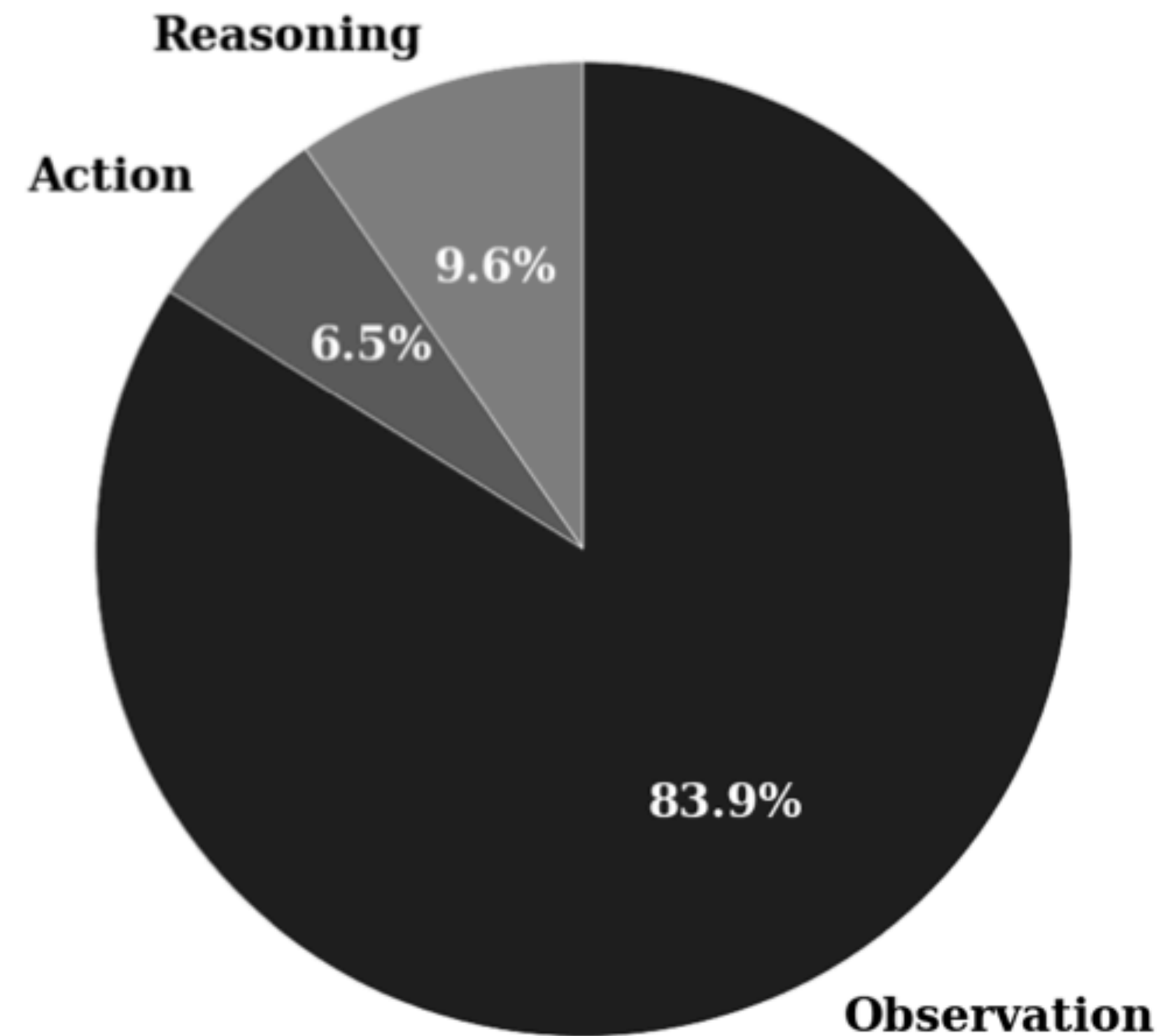


- Have the LLM summarize prior reasoning, action, observation instances.
- Compress the history into fewer tokens.
- Can be helpful for the compaction itself to be agentic.

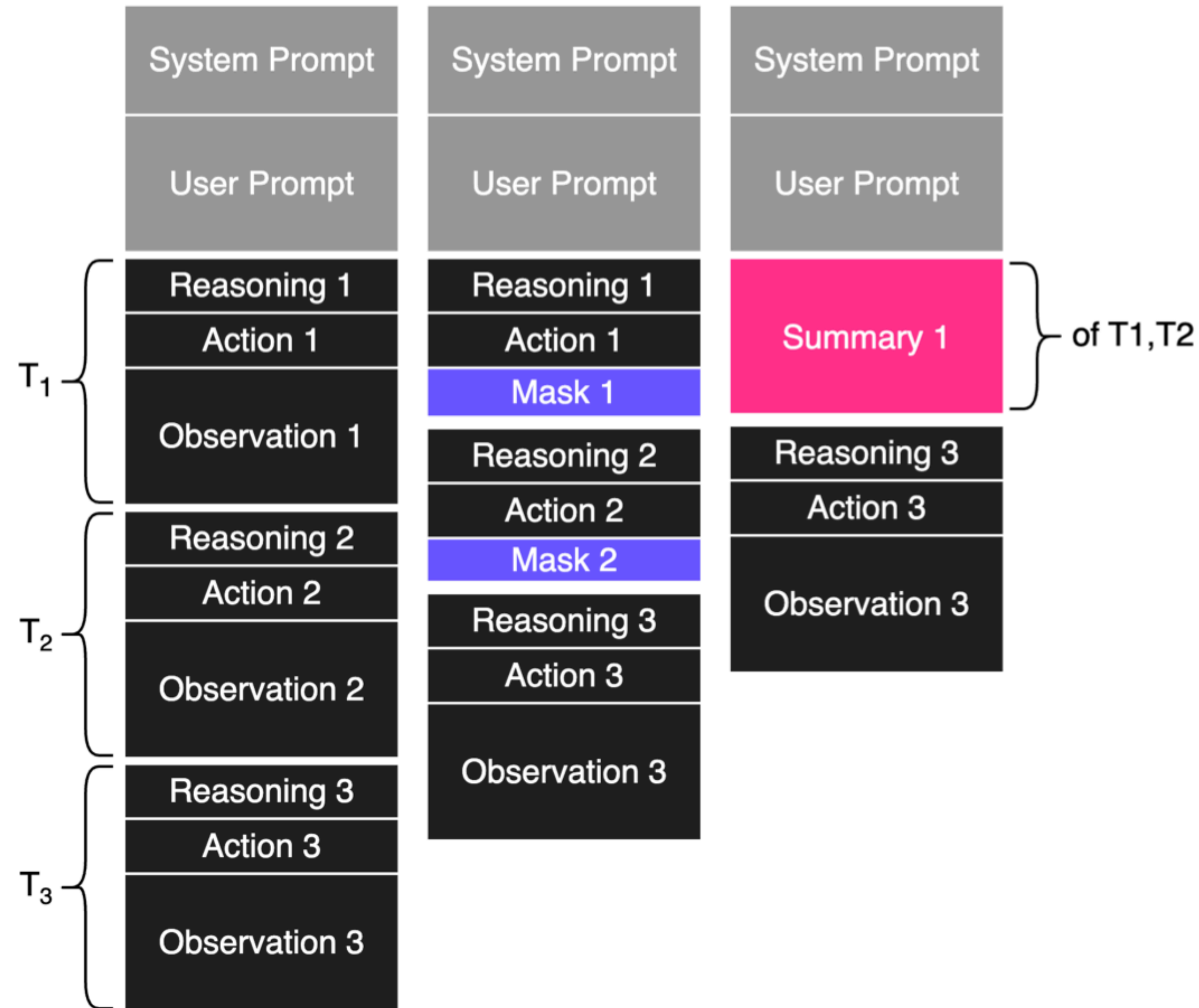


Observation Masking

Token Type Distribution - Raw Agent



Observation Masking

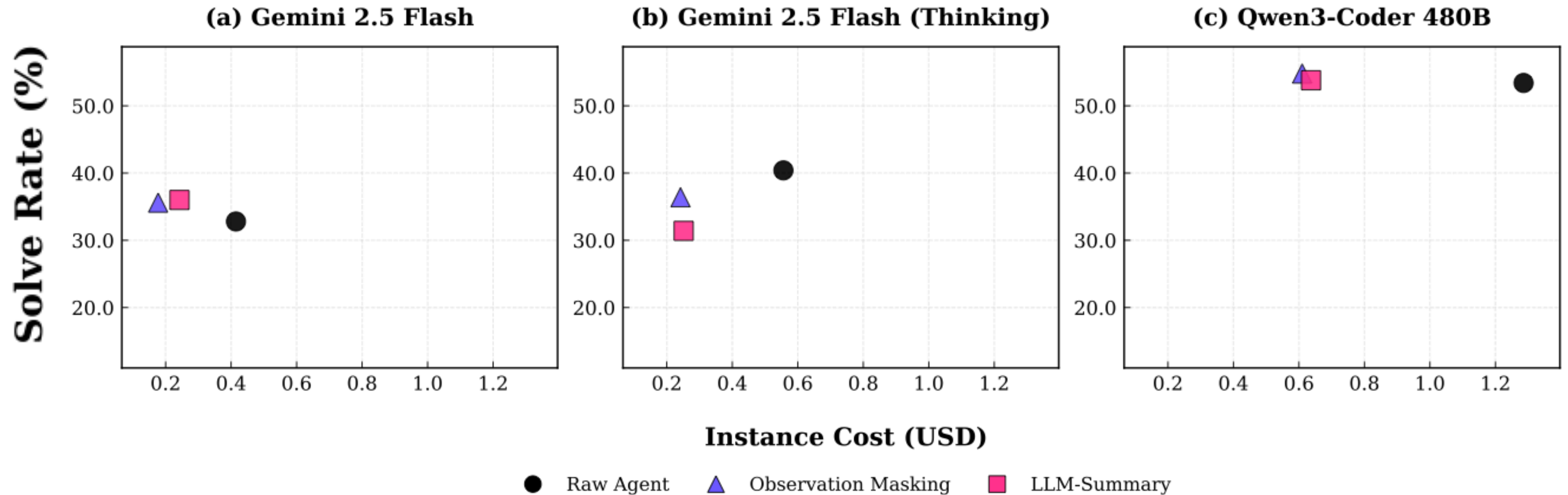


- Instead of having an LLM compact the context into a summary you can take old observations and truncate them.
- Especially useful for coding agents where the observations can be quite long.

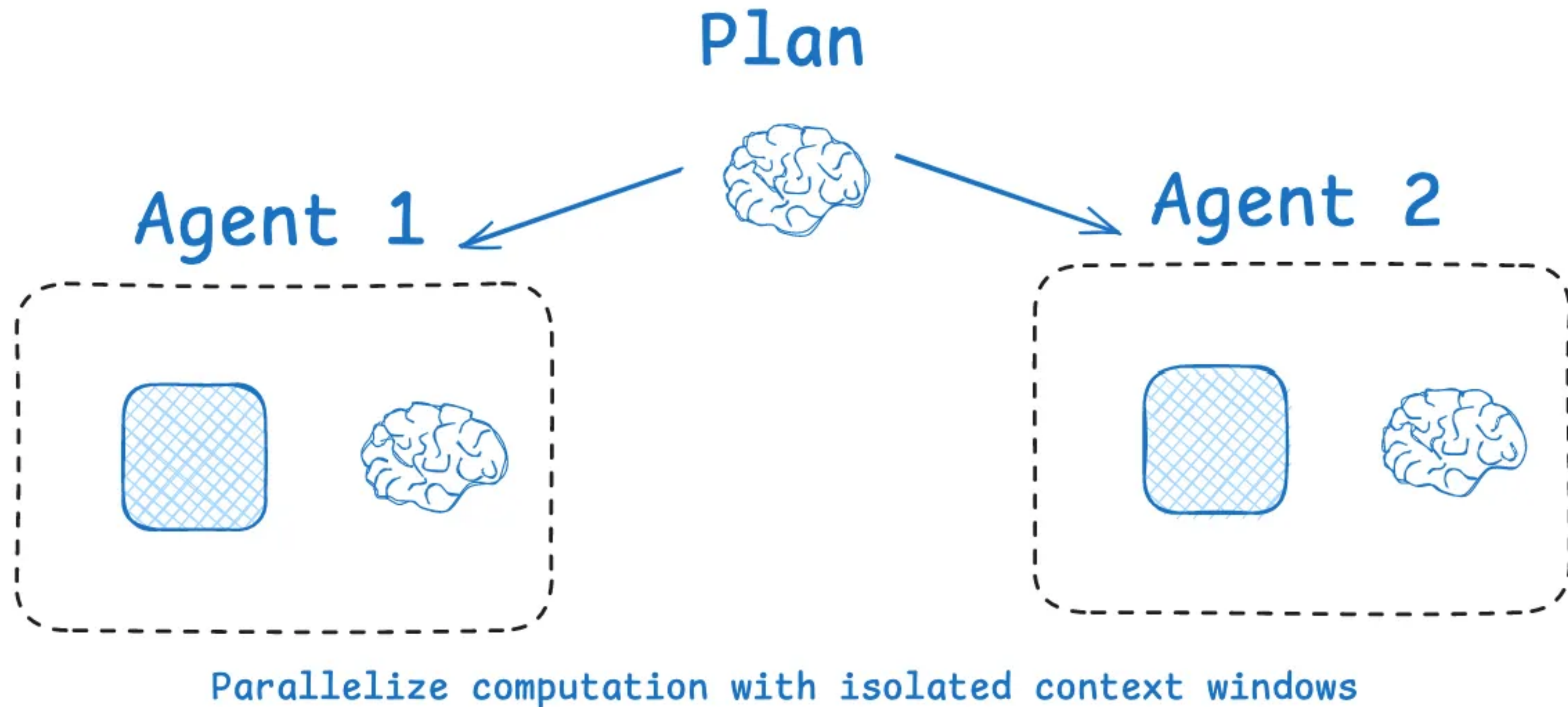
(a) Raw Agent (b) Observation Masking (c) LLM-Summary



Observation Masking



Context Isolation



Context Isolation: Progressive Disclosure

pdf/SKILL.md

YAML Frontmatter

```
----
name: pdf
description: Comprehensive PDF toolkit for extracting text and tables,
merging/splitting documents, and filling-out forms.
----
```

Markdown

Overview

This guide covers essential PDF processing operations using Python libraries and command-line tools. For advanced features, JavaScript libraries, and detailed examples, see `./reference.md`. If you need to fill out a PDF form, read `./forms.md` and follow its instructions.

Quick Start

```
```python
from pypdf import PdfReader, PdfWriter
```

#### # Read a PDF

```
reader = PdfReader("document.pdf")
print(f"Pages: {len(reader.pages)}")
```

...

You don't need all tools and context in the initial prompt, you can disclose things later as needed (e.g. skill files)



# Context Management

**NOTE:** These strategies are all compatible. You can first offload knowledge to memory then compact the context. You can both use an event log to build the context and truncate the observations. Real systems are combinations of many or all of these strategies.



# Recap



# Context Management Taxonomy

## Write Context

Long-term memories  
(across agent sessions)



Scratchpad  
(within agent session)



State  
(within agent session)



## Select Context

Retrieve relevant tools



Retrieve from scratchpad



Retrieve long-term memory

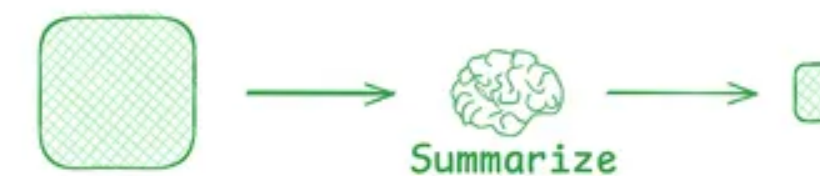


Retrieve relevant knowledge



## Compress Context

Summarize context  
to retain relevant tokens



Trim context to  
remove irrelevant tokens



## Isolate Context

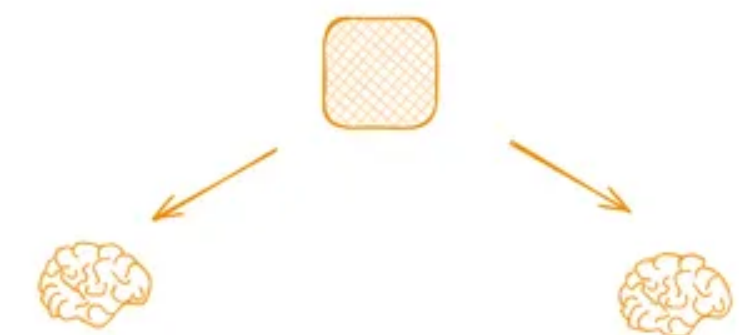
Partition context in state



Hold in environment/sandbox



Partition across multi-agent



 = LLM Context



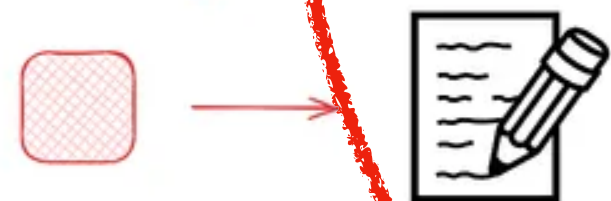
# Context Management Taxonomy

## Write Context

Long-term memories  
(across agent sessions)



Scratchpad  
(within agent session)



State  
(within agent session)



## Select Context

Retrieve relevant tools



Retrieve from scratchpad



Retrieve long-term memory



Retrieve relevant knowledge



## Compress Context

Summarize context  
to retain relevant tokens



Trim context to  
remove irrelevant tokens

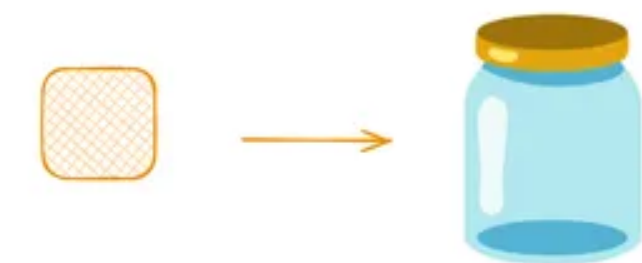


## Isolate Context

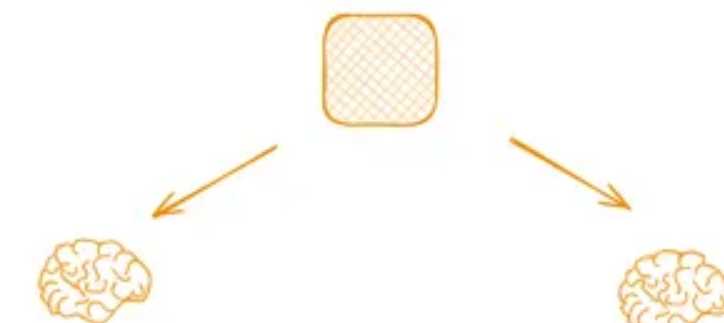
Partition context in state



Hold in environment/sandbox



Partition across multi-agent



 = LLM Context **This Wednesday**

**Next Monday**



# Recap

- Event Logs, Memory, Compaction, Observation Masking, and Context Isolation make up a few key design patterns for context management.
- Real systems combine these patterns together to manage the context.
- Context engineering is driven by two core principles: **keep the context focused** and **preserve continuity**.



# Case Study: Pi

---

Workflows vs Agents

---

Workflow Designs

---

Reasoning Designs

---

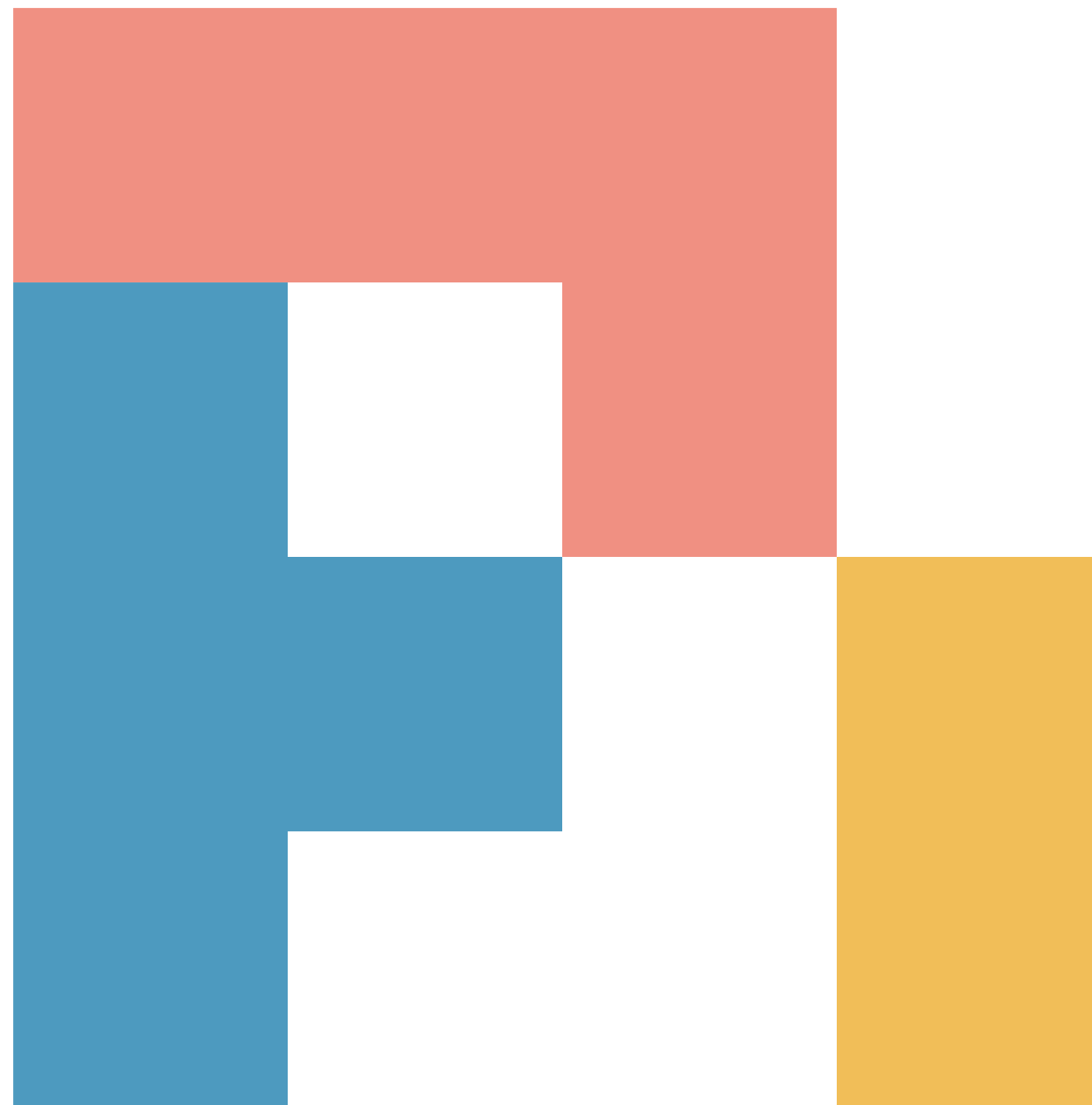
Context Designs

---

Case Study



# Pi Minimalist Agent Harness



# Pi Design Philosophy

“Pi is a minimal agent harness. Adapt Pi to your workflows, not the other way around.”

Pi is an extremely basic harness and people build tons of extensions on top to add all the features of other mainstream harnesses.

Things Pi is lacking:

- No Sub-Agents
- No Permission Checks
- No Plan Mode
- No agent to-do lists
- No bash in the background



# Pi Tools

Built-in	Purpose
<code>read</code>	Read text files and supported images
<code>bash</code>	Run shell commands
<code>powershell</code>	Run PowerShell commands on Windows
<code>edit</code>	Apply exact text replacements to an existing file
<code>write</code>	Create or overwrite a file
<code>grep</code>	Search file contents
<code>find</code>	Find paths using glob patterns
<code>ls</code>	List directory contents

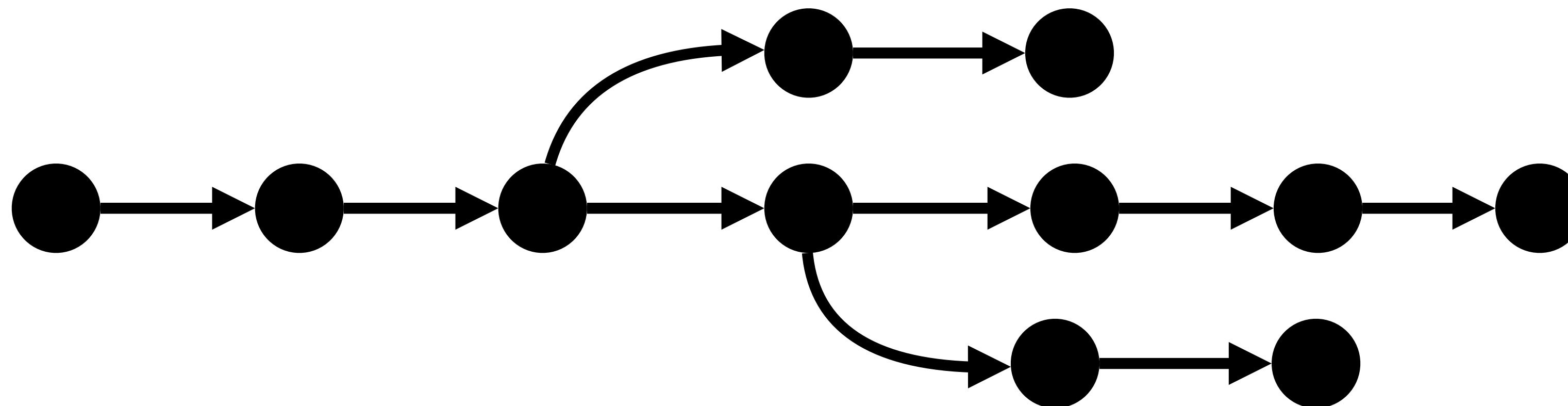
- Only read, bash, edit, and write are enabled by default.
- Minimal tool bloat.
- All other tools are subsets of bash.



# Pi Agent Loop

Pi has a ReAct style loop of reasoning before acting on JSON tool calls.

Context is built as a tree, and you can return to a prior branch to continue a different thread of the conversation.



# Pi Context Management

1. Event Log to LLM Input Conversion
2. Progressive Disclosure through Skill Files
3. Compaction



# Pi Safety



Pi runs at the same permissions as the process, and has no popups to ask for permission, so consider it always on auto-mode with no protections.



# Recap

- Pi is designed to be a minimalist harness.
- The community has built several extensions.
- The default tools are bash, read, write, and edit.
- Pi is a ReAct style agent.
- Context is managed through event logs, progressive disclosure, and compaction.
- Pi has no safeguards built in.



# Summary

---

Workflows vs Agents

---

Workflow Designs

---

Reasoning Designs

---

Context Designs

---

Case Study



# Summary

- ✓ Review: Workflows vs Agents
- ✓ Workflow Design Patterns
- ✓ Agent Design Patterns: Reasoning/Acting
- ✓ Agent Design Patterns: Context
- ✓ Case Study: Pi



***Remember*** to come to class Wednesday  
where we will talk about **Memory**!

