Agentic Architectures Tutorial

Wee Sun LEE
National University of Singapore

Yan XIA
MSRA

Primer on LLM and Generative AI

Attention Is All You Need

Ashish Vaswani* Google Brain avaswani@google.com

Noam Shazeer* Google Brain noam@google.com

Niki Parmar* Google Research nikip@google.com

Jakob Uszkoreit* Google Research usz@google.com

Llion Jones* Google Research llion@google.com

Aidan N. Gomez* † University of Toronto aidan@cs.toronto.edu

Łukasz Kaiser* Google Brain lukaszkaiser@google.c

Illia Polosukhin* ‡ illia.polosukhin@gmail.com

Abstract

The dominant sequence transduction models are based on complex recurrent or convolutional neural networks that include an encoder and a decoder. The best performing models also connect the encoder and decoder through an attention mechanism. We propose a new simple network architecture, the Transformer, based solely on attention mechanisms, dispensing with recurrence and convolutions entirely. Experiments on two machine translation tasks show these models to be superior in quality while being more parallelizable and requiring significantly less time to train. Our model achieves 28.4 BLEU on the WMT 2014 English-to-German translation task, improving over the existing best results, including ensembles, by over 2 BLEU. On the WMT 2014 English-to-French translation task, our model establishes a new single-model state-of-the-art BLEU score of 41.8 after training for 3.5 days on eight GPUs, a small fraction of the training costs of the best models from the literature. We show that the Transformer generalizes well to other tasks by applying it successfully to English constituency parsing both with large and limited training data.

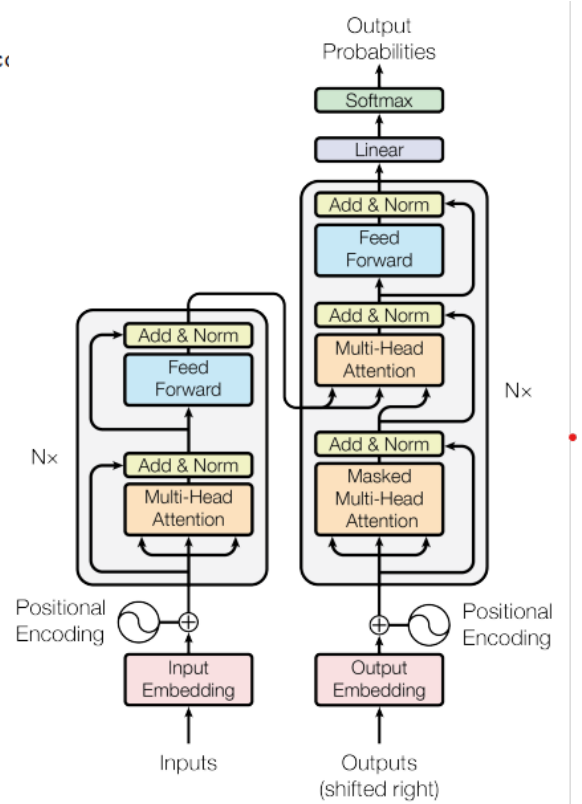


Figure 1: The Transformer - model architecture.

- Usually **transformer**, a deep neural network architecture is used
- The transformer takes as input previously seen tokens to generate the next token
  - Words are tokens in Large Language Models
  - Images can be tokenized by dividing into small subimages

Probabilistic Model

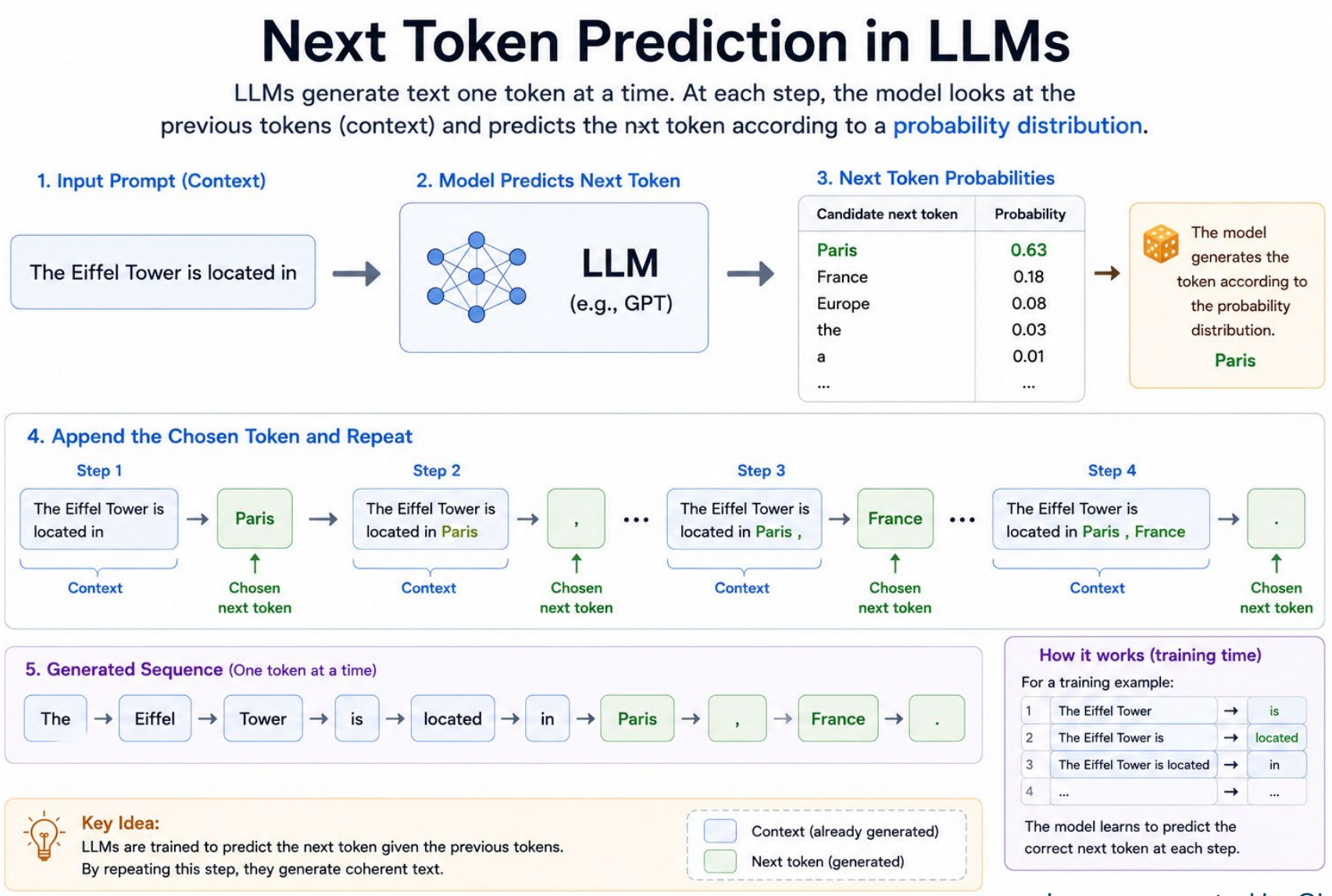
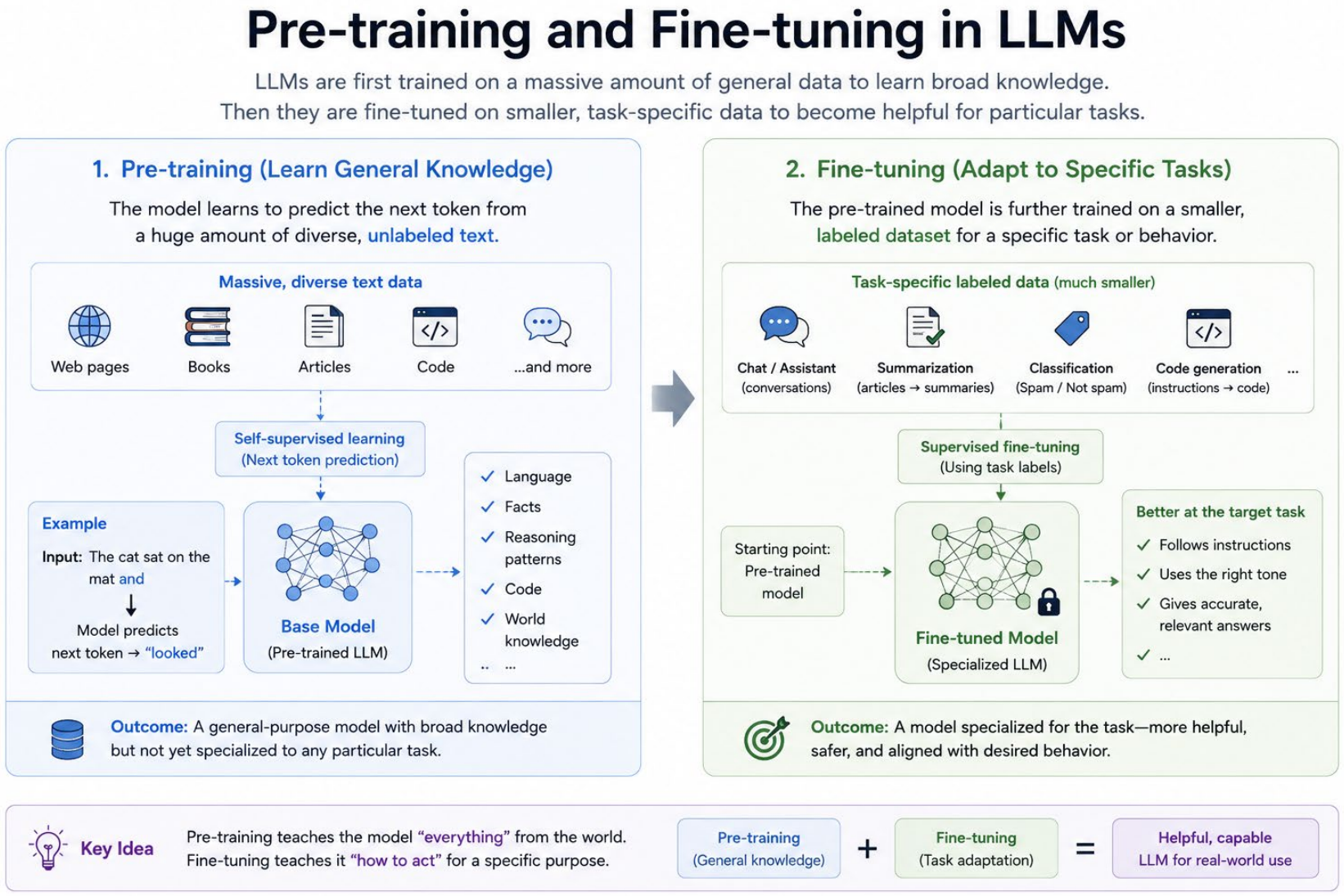


Image generated by ChatGPT

- The transformer takes as input previously seen tokens to generate the next token
  - Words are tokens in Large Language Models
  - Images can be tokenized by dividing into small subimages
- The transformer generates next token by computing $P(w_t|w_{t-1}, w_{t-1}, \dots, w_0)$ and sampling from it to get a sample of w_t.
  - Conditional probability is learned from data
  - $P(w_t, w_{t-1}, \dots, w_0) = P(w_0)P(w_1|w_0) \dots P(w_t|w_{t-1}, \dots, w_0)$ from chain rule of probability
  - Trained with supervised learning to predict next word
    - Often called self-supervised learning as label is constructed from sequence itself

Conditional Generation

- How do we get the LLMs to do what we ask (prompt) for?
- Essentially, we want conditional generation given the prompt $P(w_t, w_{t-1}, \dots, w_0 \mid prompt)$.
  - Construct instruction tuning dataset consisting of many pairs of prompts and responses
  - Do fine-tuning with the dataset so that $P(w_t, w_{t-1}, \dots, w_0 \mid prompt)$ more accurately reflect the conditional distribution
    - Learns quickly – knowledge from pre-training transfers well for fine-tuning
  - Can improve further with reinforcement learning.
- Most tasks can be viewed this way: answer questions, compose emails, make travel plans, summarize text, chatbot, …



Is an LLM sufficient? An AI Scientist Agent

Consider what an AI scientist needs to do:

1. Produce an idea
2. Assess impact and search prior works for novelty
3. If impactful and novel
   - Design method
   - Design and run experiments
   - Observe and record outcomes
   - Do analysis
4. If assessed as successful, write paper, otherwise go to 1.

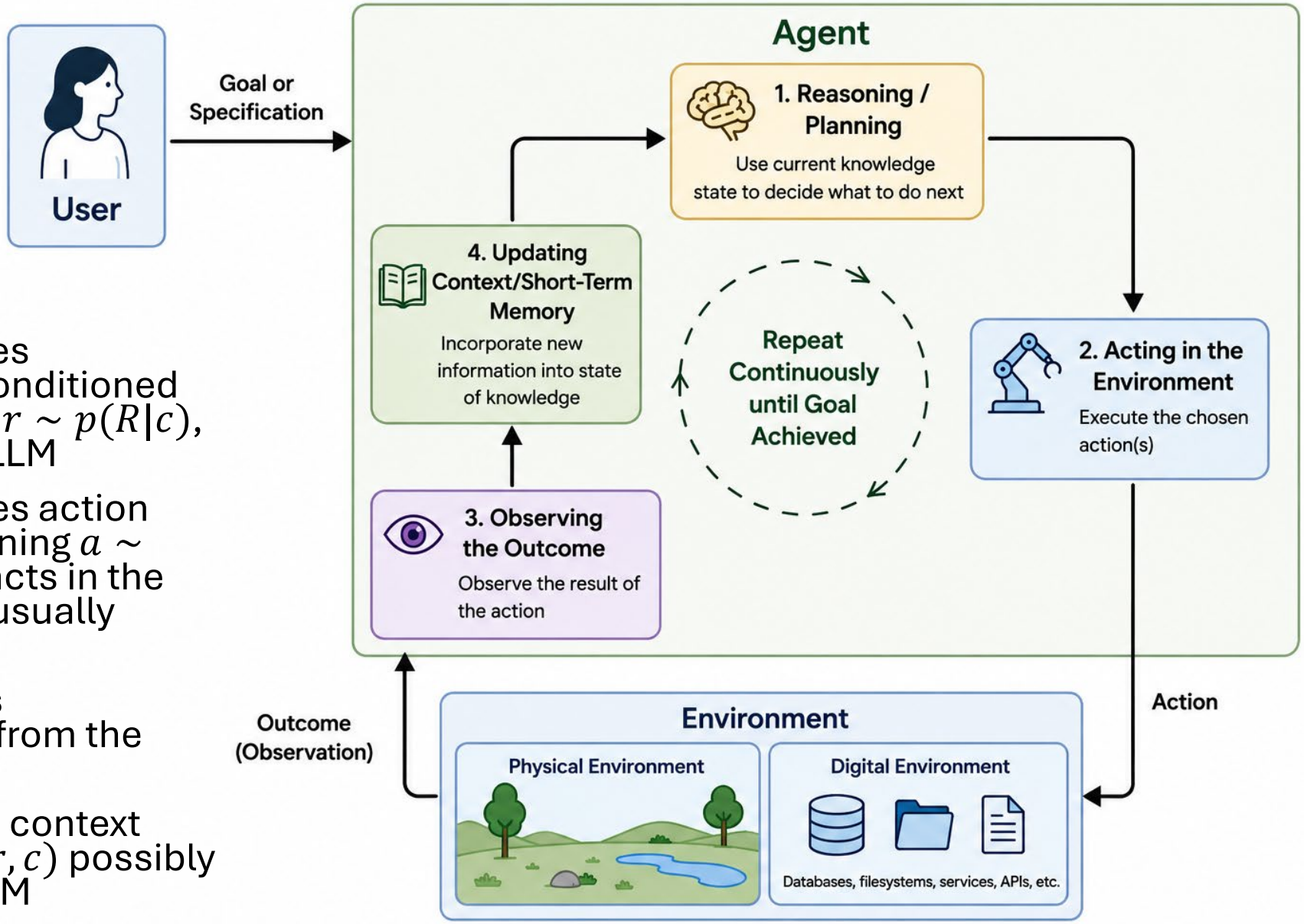
Is an LLM sufficient? An AI Scientist Agent

Can this be done with a standard LLM?

- The task unfolds over time.
- Intermediate state must be retained.
- External systems must be queried.
- Actions may fail or return unexpected results.
- The system must decide what to do next.
- Claims and outputs must be verified.
- Cost, latency, and risk accumulate over multiple steps.

Agentic architecture is the design of these types of loops to enable LLMs to complete these types of tasks successfully.

Agentic Workflow



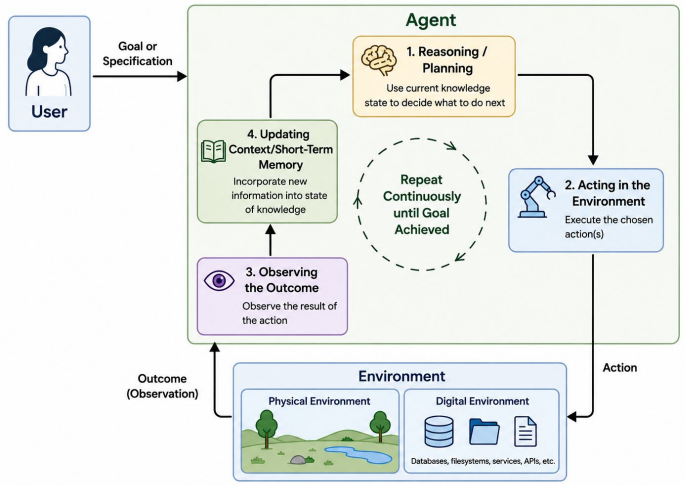
1. Agent generates reason/plan conditioned on its context: $r \sim p(R|c)$, usually using LLM
2. Agent generates action given its reasoning $a \sim p(A|r,c)$ and acts in the environment, usually using LLM
3. Agent receives observation o from the environment
4. Agent updates context $c' \sim p(C|o,a,r,c)$ possibly with help of LLM

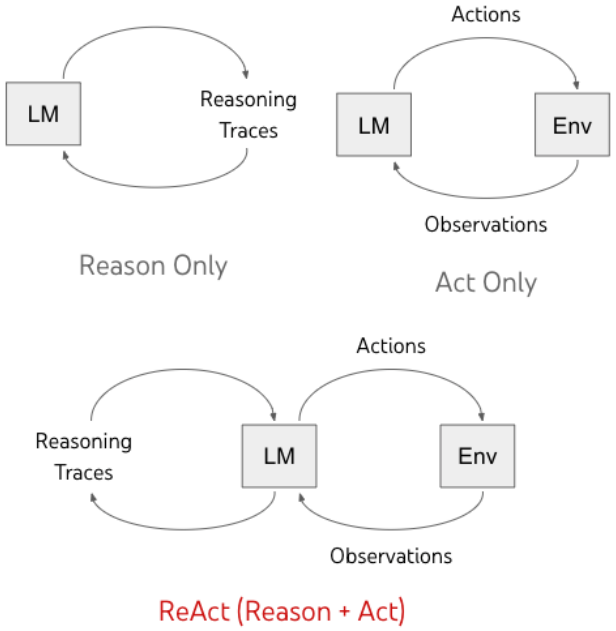
Example: ReAct Agent

Yao et al. ICLR 2023

Early work on LLM agent.

- Added reasoning into loop





(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 …

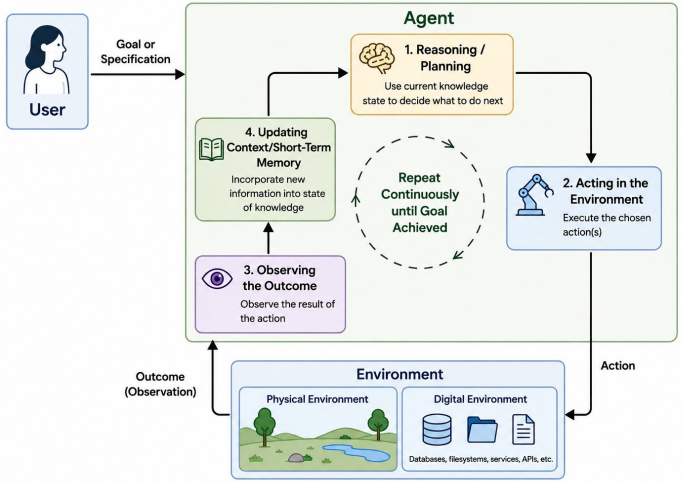
Thought 2: Apple Remote was originally designed to control the Front Row media center program. I need to search Front Row next and find what other device can control it.
Act 2: Search[Front Row]
Obs 2: Could not find [Front Row]. Similar: ['Front Row Seat to Earth', 'Front Row Motorsports',' Front Row (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] ✓

Figures from paper and https://research.google/blog/react-synergizing-reasoning-and-acting-in-language-models/

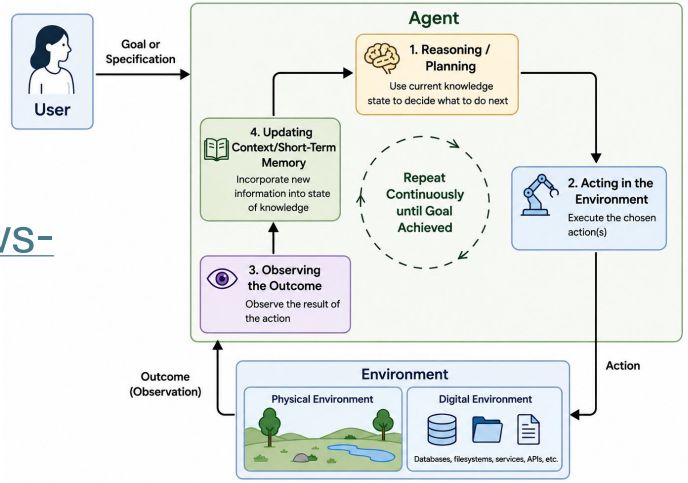
Decomposing Agentic Workflow Further



- Instead of asking LLM to do everything, decompose workflow into predetermined code paths.
  - Often called harness, scaffold, framework, etc.
- Aim to
  - Improve reliability through determinism, structured outputs, verification, error handling, retries, etc., improve latency through parallelism, etc.
  - Provide access to tools, e.g. command-line interface, coding, retrieval, web search, APIs, etc., also sandboxes to execute and verify.
  - Handle short-term memory/context/state and long-term memory (e.g. file-system, git, database), etc.
- Subtasks sometimes done by sub-agents with its own context and can run independently.

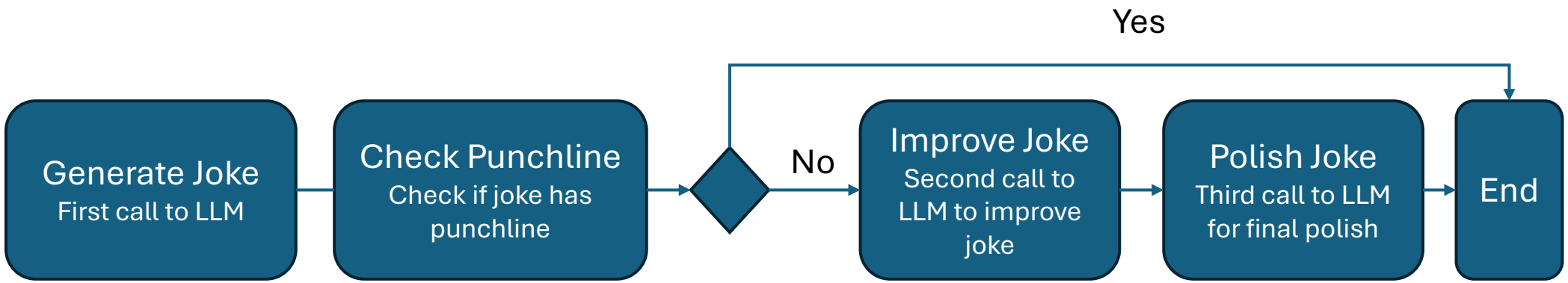
Common Workflows/Patterns

See examples at https://docs.langchain.com/oss/python/langgraph/workflows-agents



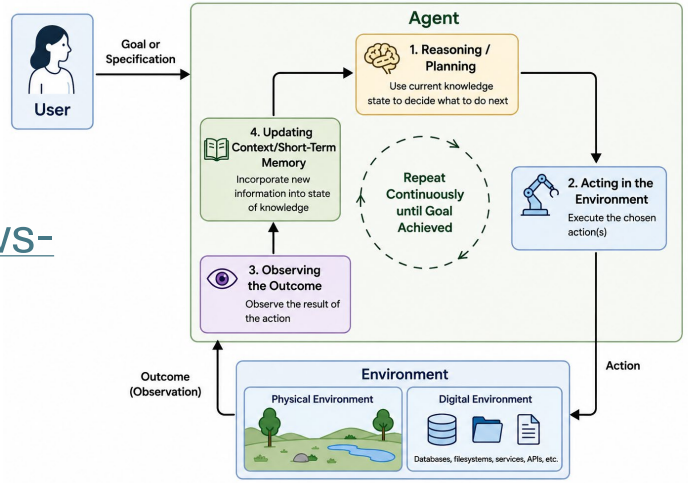
Prompt Chaining

- Each LLM call processes output of previous call



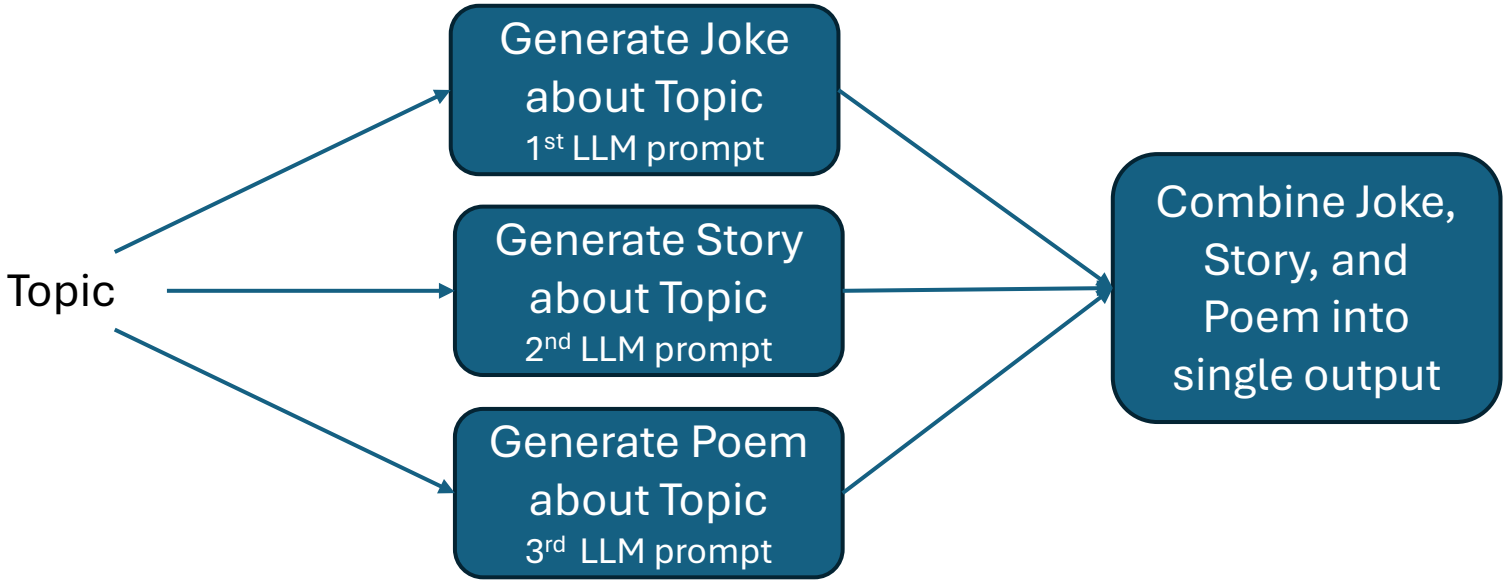
Common Workflows/Patterns

See examples at https://docs.langchain.com/oss/python/langgraph/workflows-agents



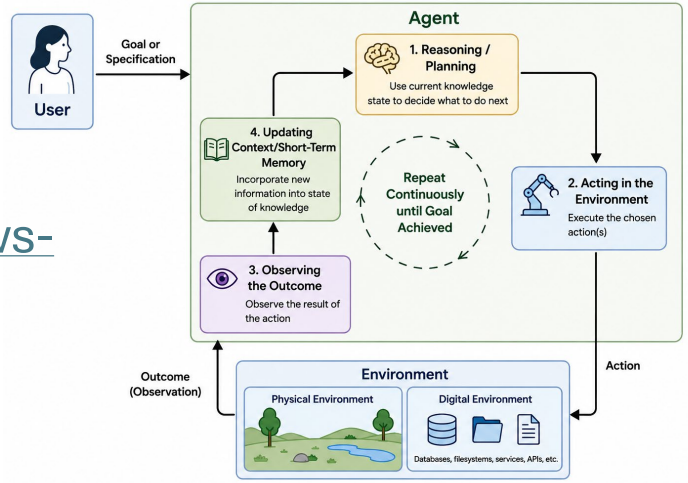
Parallelization

- Each LLM work simultaneously on a task.
  - Run in parallel to decrease latency
  - Run multiple times to increase confidence



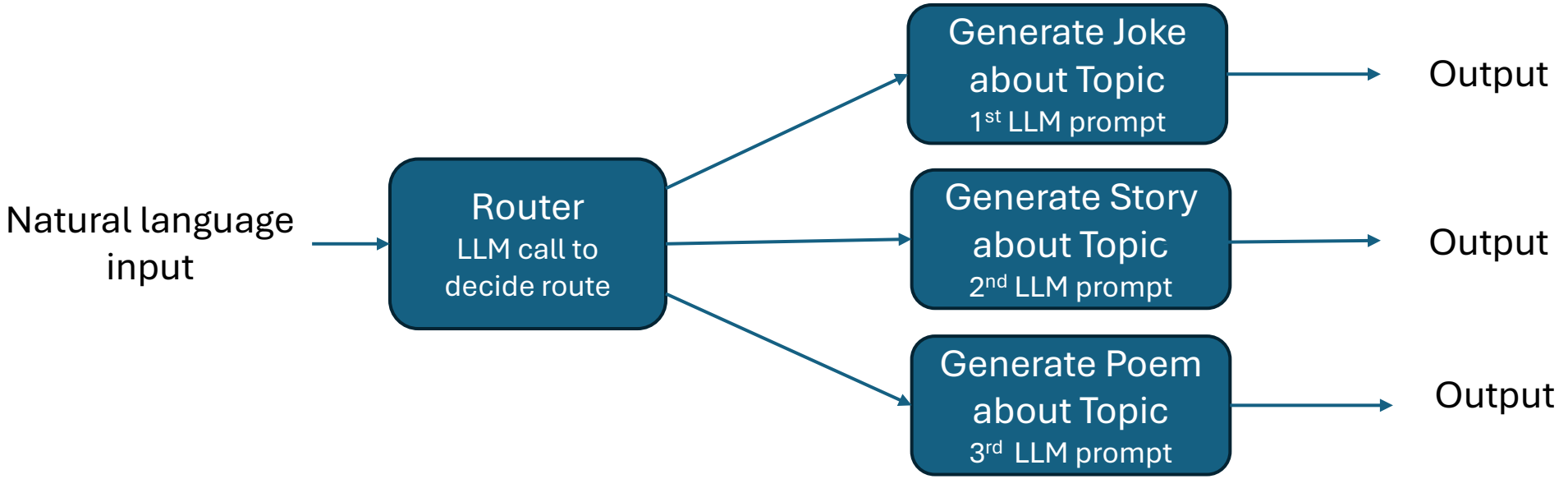
Common Workflows/Patterns

See examples at https://docs.langchain.com/oss/python/langgraph/workflows-agents



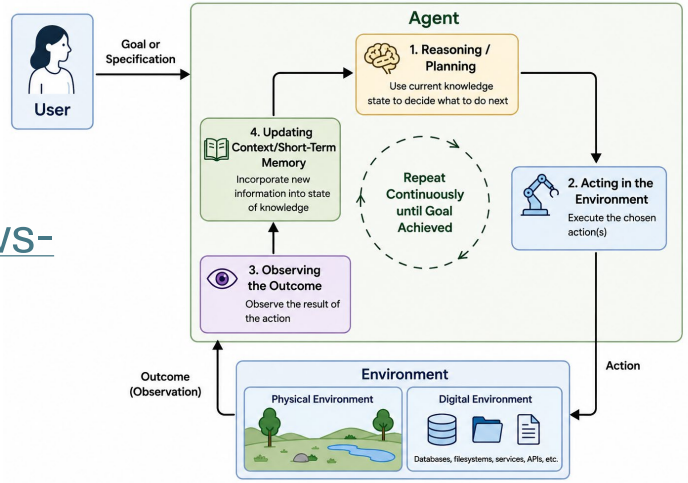
Routing

- Decide on which path to take, then route through the path



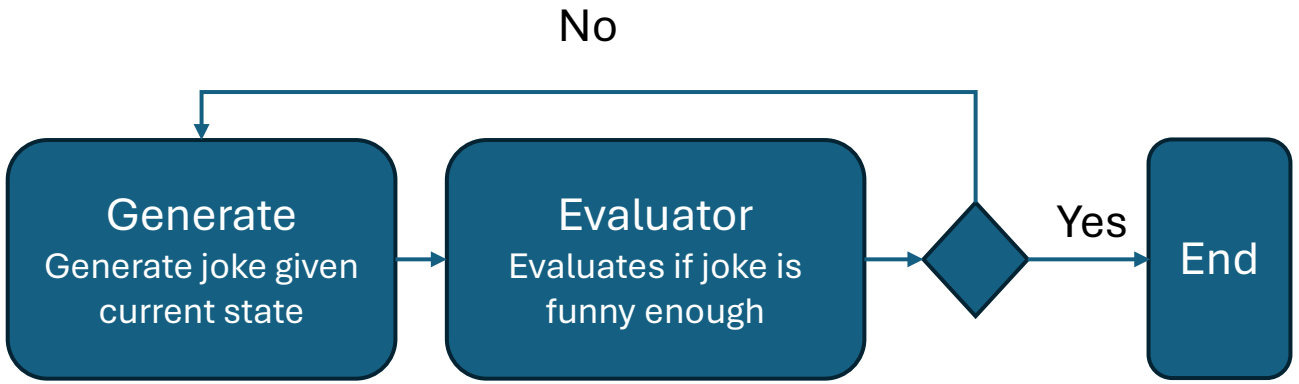
Common Workflows/Patterns

See examples at https://docs.langchain.com/oss/python/langgraph/workflows-agents



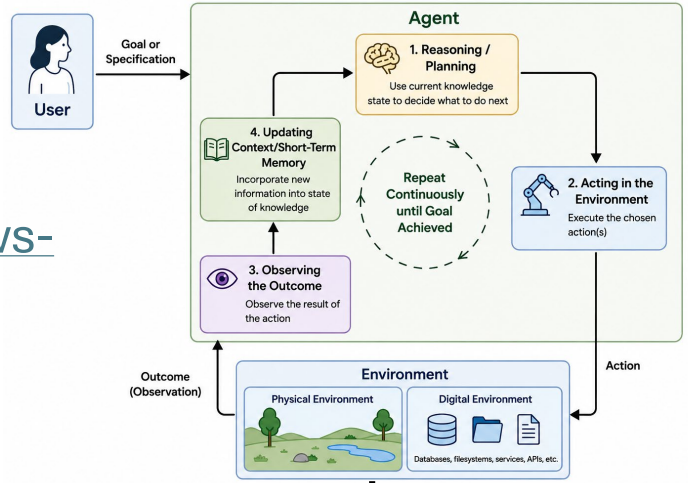
Evaluator-Optimizer

- One LLM call generates, another evaluates (can be human-in-the-loop.
- If refinement needed, feedback provided and new version generated



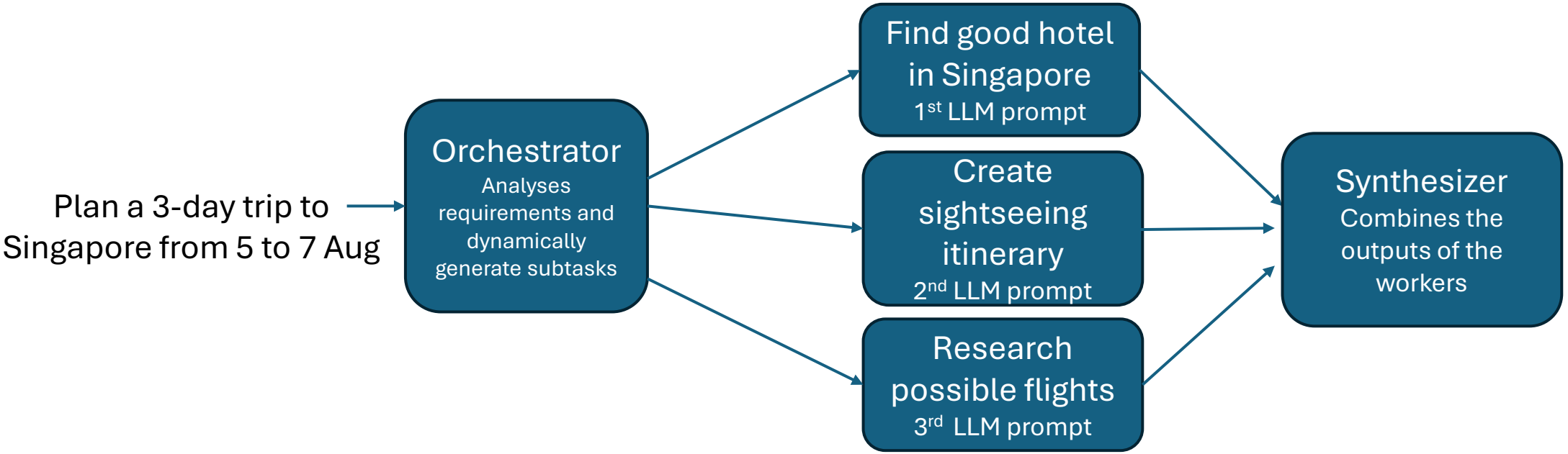
Common Workflows/Patterns

See examples at https://docs.langchain.com/oss/python/langgraph/workflows-agents



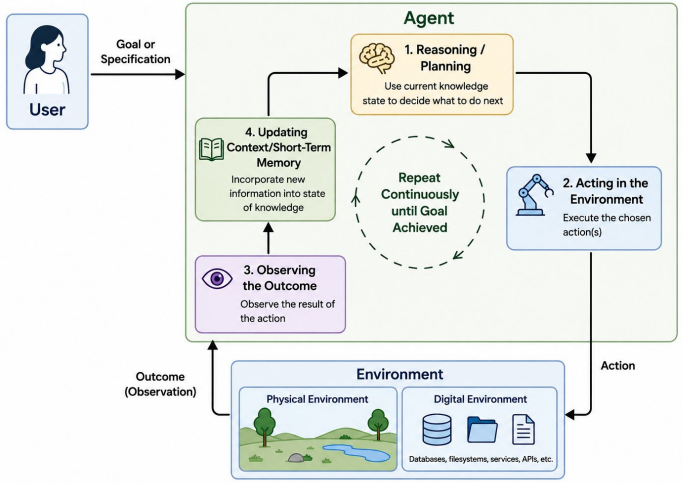
Orchestrator-Worker

- Orchestrator analyses the task and dynamically creates workers (human in the loop if necessary).
- Workers generate outputs that is combined by the synthesizer



Tools

Example from: https://huggingface.co/learn/agents-course/en/unit1/tools



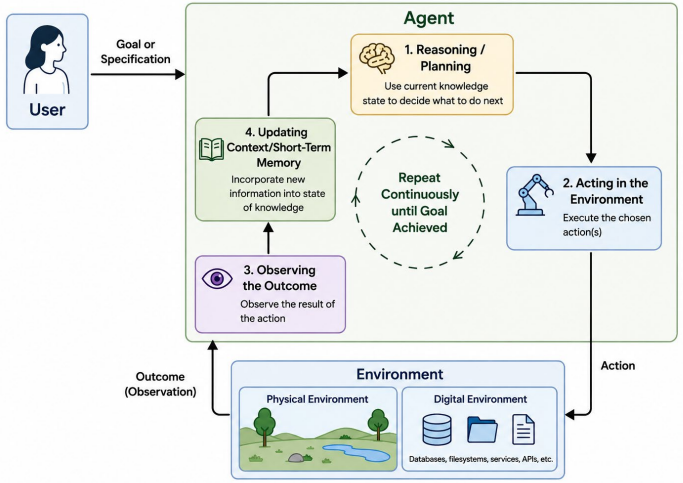
- AI agents take actions in the environment though use of tools.
  - LLM generate text that represent the tool call
  - LLM needs to know what the tools do and what exact inputs they expect
    - Usually placed in the context by the system at the start
    - Example with a calculator tool:

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

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

Tools

Reference: https://huggingface.co/learn/agents-course/en/unit1/tools

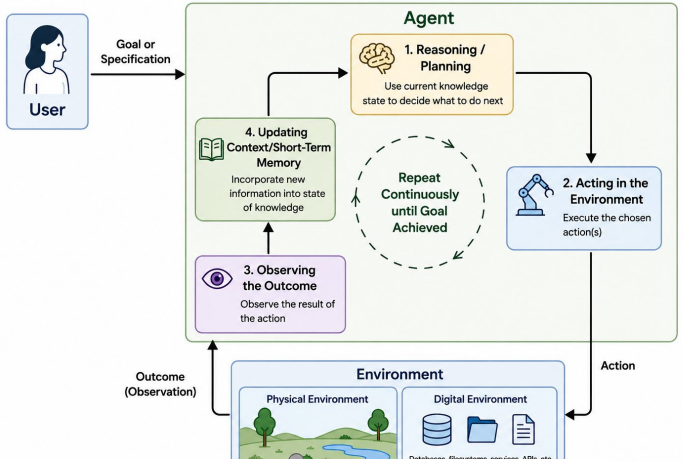


- Some commonly used tools
  - Command line interface
    - Create/list/change directories, run programs through command line, etc.
  - Retrieval from external source.
    - Retrieval Augmented Generation (RAG) is a commonly used method.
    - Index document collection, do similarity search through embedding
  - Database queries, e.g. through SQL
  - Search the web and retrieve web pages
  - Model Context Protocol (MCP) provides a standardized interface for LLM tools
    - Pre-built integration for the LLM
    - Fast growing list of MCP servers available: Github, Notion, Slack, …

Tools

Example from https://huggingface.co/learn/agents-course/en/unit1/actions

One highly expressive way for generating actions is to use code as action



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

Available APIs

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

LLM Agent using [**Text/JSON**] as Action

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

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

Environment 1.1, 0.19

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

Environment 700

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

Environment 916.3

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

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

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

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

Fewer Actions Required!

CodeAct: LLM Agent using [**Code**] as Action

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

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

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

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

Control & Data Flow of Code Simplifies Complex Operations

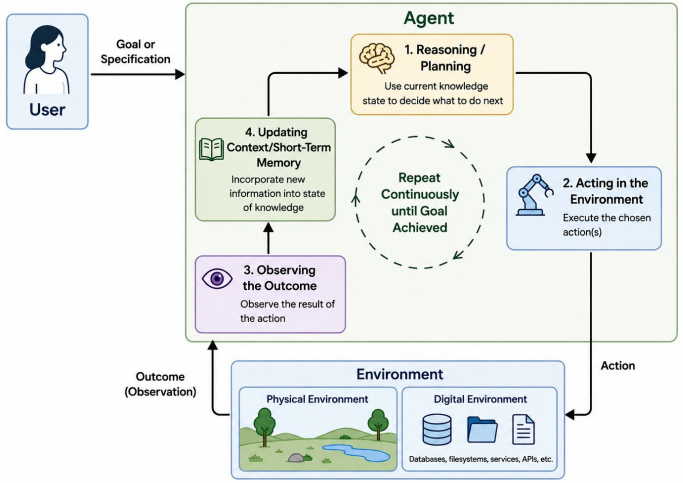
Re-use `min` Function from Existing Software Infrastructures (Python library)

Environment 1.1, 0.19

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

Memory

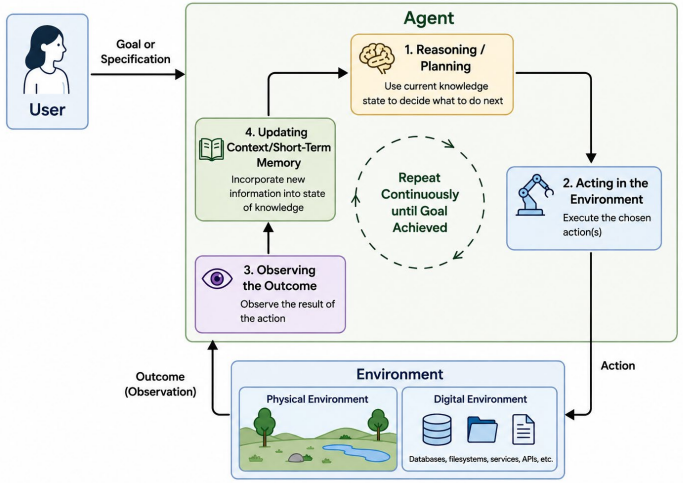
Reference: https://docs.langchain.com/oss/python/concepts/memory



- Memory is often divided into
  - Short-term memory that tracks what has happened in the current session
  - Long-term memory that stores data across sessions
- Simplest method of managing short-term memory is to append the entire history of what is generated and observed into the LLM context. Multiple issues:
  - LLM performance is poorer over long context, sometimes called *context rot*.
  - Long context is computationally expensive
    - Quadratic time, although exact prefix can be cached
    - System message with instructions and tool description can be long even before processing starts
  - Maximum context length is fixed, depending on model.
  - When context too long, often trim messages, delete some messages, summarize messages (using LLM), etc.

Memory

Example from https://docs.langchain.com/oss/python/concepts/memory



- Long-term memory is often classified into semantic, episodic, and procedural

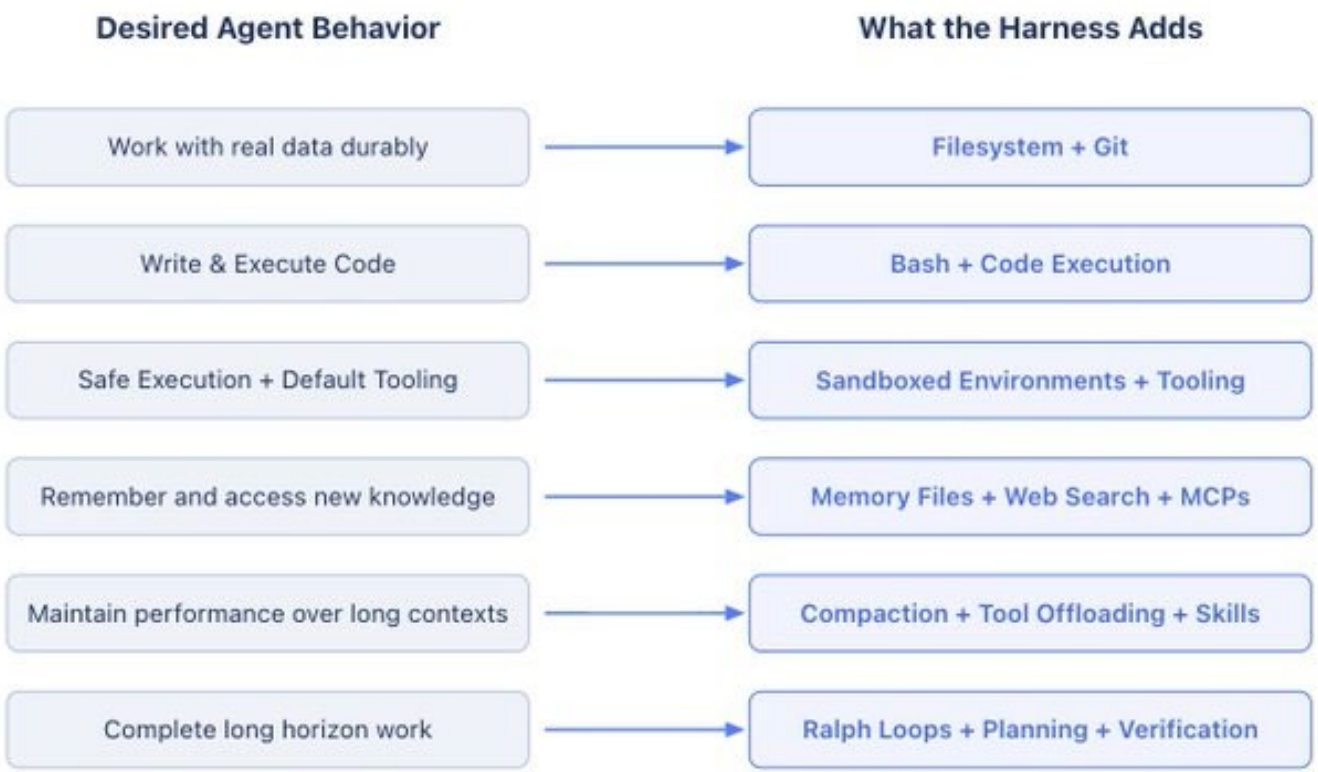
Memory Type	What is Stored	Human Example	Agent Example
Semantic	Facts	Things I learned in school	Facts about a user
Episodic	Experiences	Things I did	Past agent actions
Procedural	Instructions	Instincts or motor skills	Agent system prompt

- Long-term memory is often stored in files, databases, etc.
  - Can be retrieved with tools such as RAG (retrieval augmented generation), database queries, reading files, etc.
  - Some agentic systems allow procedural knowledge to be stored as *skills* and loaded with the system message.

Example: Codex Harness and Coding

Reference and images: https://openai.com/index/unrolling-the-codex-agent-loop/ and https://www.langchain.com/blog/the-anatomy-of-an-agent-harness

- Codex is OpenAI's coding agent
- Initial LLM prompt consists of
  - System message from config.toml
  - Tools including Command Line Interface (CLI), web search, MCP servers
  - User instructions in AGENTS.md (README for agents)
  - Skills (procedural knowledge) from SKILL.md
- Uses prompt caching for exact prompt prefix for performance
- Compact the conversation using summarization once number of tokens exceeds some threshold
- Post-trained with models and harness in the loop



Each harness feature derives from a behavior the model can't deliver on its own

Examples from Agents4Academia Hackathon

https://github.com/Agents4Academia-AI. Two week hackathon (14-26 June 2026) in Oxford, NUS, NTU.

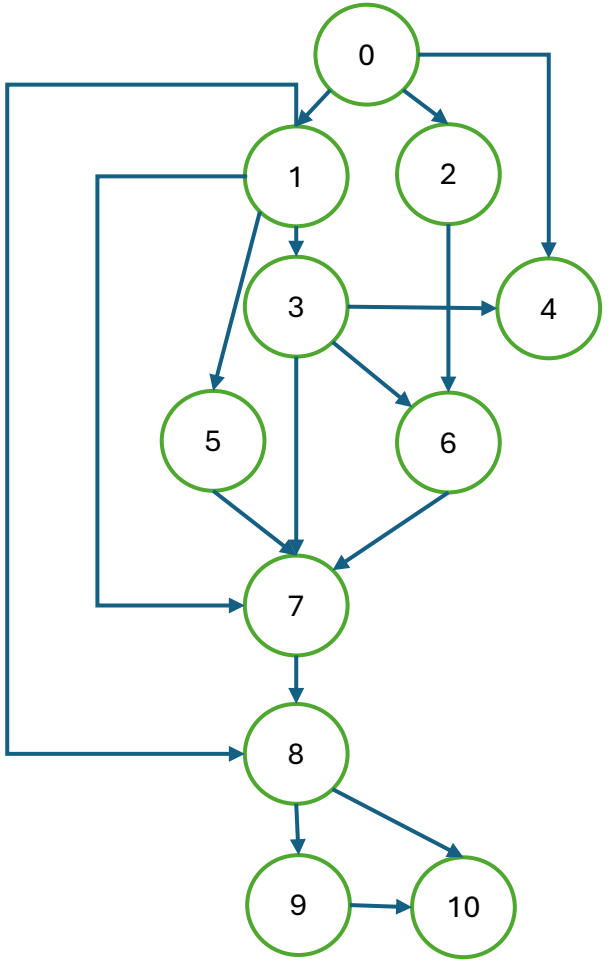
Stage	Project	What it does	
🔍	**Discover**	Prior	Builds an auditable knowledge graph of claims and contributions from primary literature, helping researchers understand the state of a field, contradictions, and open questions.
🗂️	**Organise**	UReKA	Unifies Zotero, Notion, Obsidian, and arXiv into a linked, queryable research knowledge base.
🔁	**Reproduce**	Benchmark-Replicator	Produces clean, minimal, runnable implementations of prior empirical papers.
✅	**Verify**	RefWarden	Checks whether references exist, whether their metadata is correct, and whether they actually support the claims they are cited for.
📝	**Review**	Auto-Reviewer	Turns a paper PDF into claim–evidence maps, novelty and rigor checks, and a self-critique pass for hallucinations.

Examples from Agents4Academia Hackathon

https://github.com/Agents4Academia-AI. Two week hackathon (14-26 June 2026) in Oxford, NUS, NTU.

Stage	Goal
0 Directly read the PDF	Directly read the PDF → structured paper representation: title, claims, contributions, figures/tables, etc.
1 Overall understanding	One-paragraph summary + claim-evidence map
2 Section-by-section analysis	Issues / missing information / ambiguous claims
3 Claim extraction and evidence mapping	Claim extraction and evidence mapping: categorized by novelty / correctness / empirical evidence, etc.
4 Novelty check	Use Claude's built-in web_search tool to find similar prior work
5 Significance / impact analysis from multiple perspectives	Significance / impact analysis from multiple perspectives: 5 personas
6 Rigor check	Internal correctness, claim support, experimental rigor
7 Review planning	Strengths / weaknesses / recommendation
8 Draft review	Author-facing review text
9 Self-critique	Identify hallucinations / overly strong claims / inconsistencies
10 Finalize	Apply the critique and output the final review + improvement checklist

Dependency pattern for stages in **Auto-Reviewer**



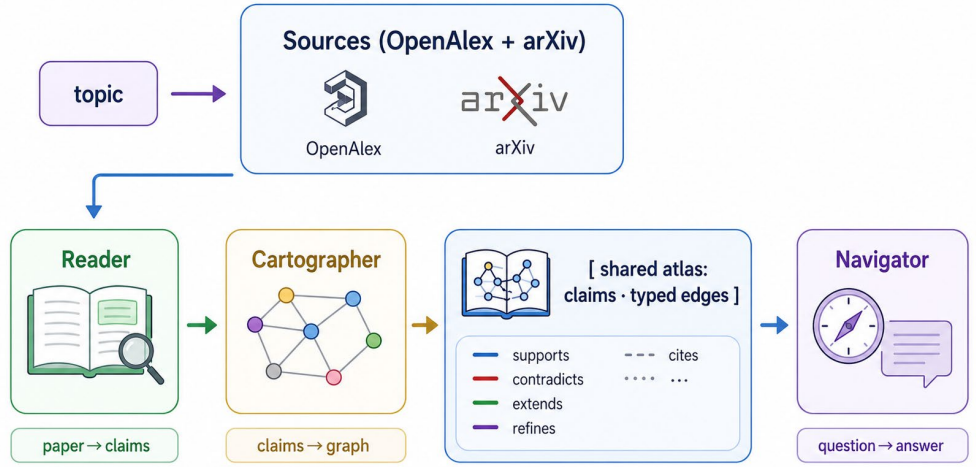
Examples from Agents4Academia Hackathon

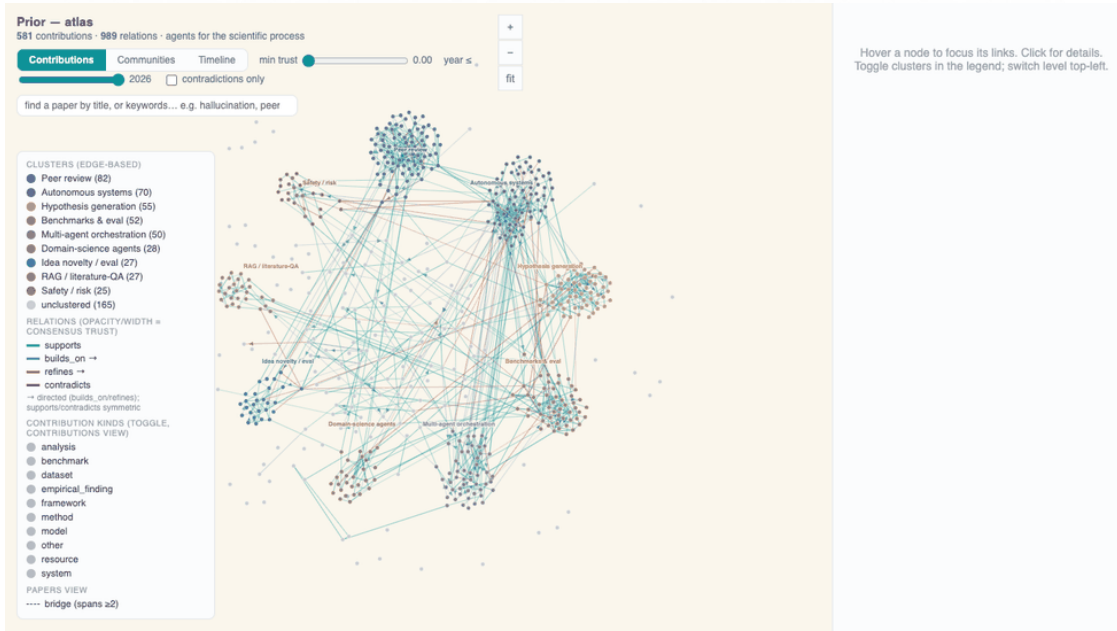
https://github.com/Agents4Academia-AI. Two week hackathon (14-26 June 2026) in Oxford, NUS, NTU.

Prior turns primary literature into a queryable atlas of claims (a typed graph) and answers questions from it — with citations, surfaced contradictions, and an honest "not found".

https://github.com/Agents4Academia-AI/prior/blob/main/docs/architecture.md

Agent	File	In → Out	How
Reader	reader.py	one paper → atomic, typed claims (+ evidence span, confidence)	one structured call per paper
Cartographer	cartographer.py	claims → typed relations (supports/contradicts/refines/extends)	BM25 proposes candidate pairs; one structured call per claim labels them (avoids O(n²))
Navigator	navigator.py	question + atlas → grounded answer	BM25 retrieves relevant claims; one call returns verdict + supporting/contradicting/open, or not_found





Multi-agent Architectures
(ChatGPT output in slide notes, use if helpful)

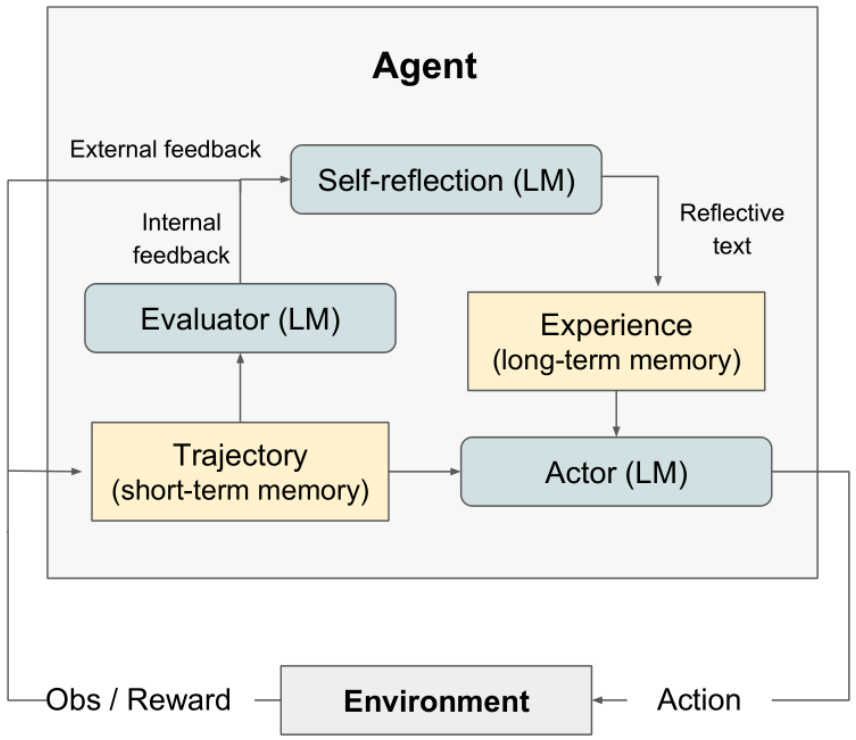
Example multi-agent scrum team

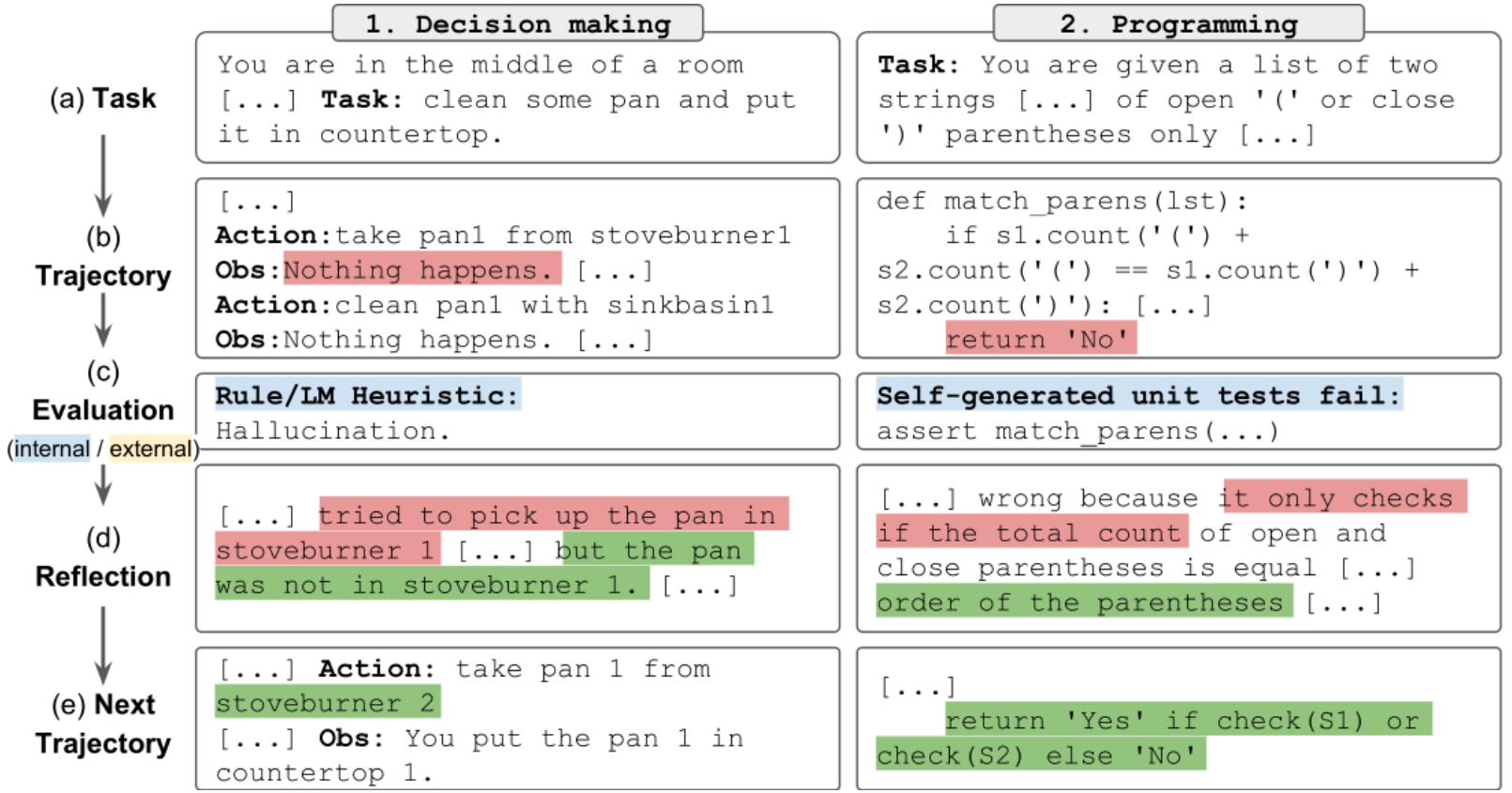
Observability and Learning

- Execution traces provide information that can be used to improve agents.
- Tracking and evaluating the execution traces often called agent **observability**
  - Include reasoning failures as well as in system issues, e.g. latency, cost of tokens, etc.
- AI tools are also used to assist evaluation
  - E.g. LLM-as-judge, often used with defined rubrics
- The system should learn and improve from the execution data
  - LLM also used to diagnose and summarize

Example: Reflexion

Shinn et al. NeurIPS 2023

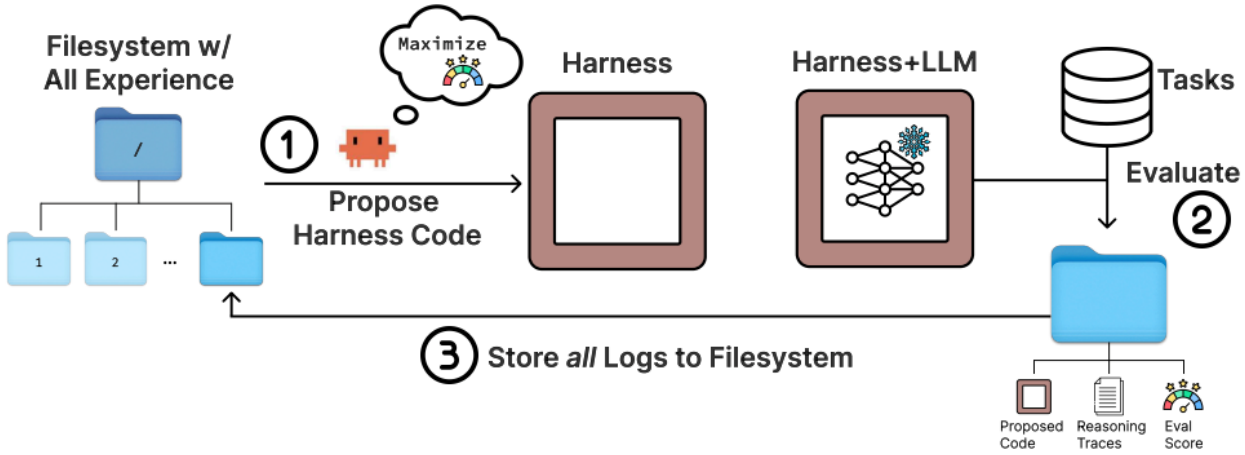


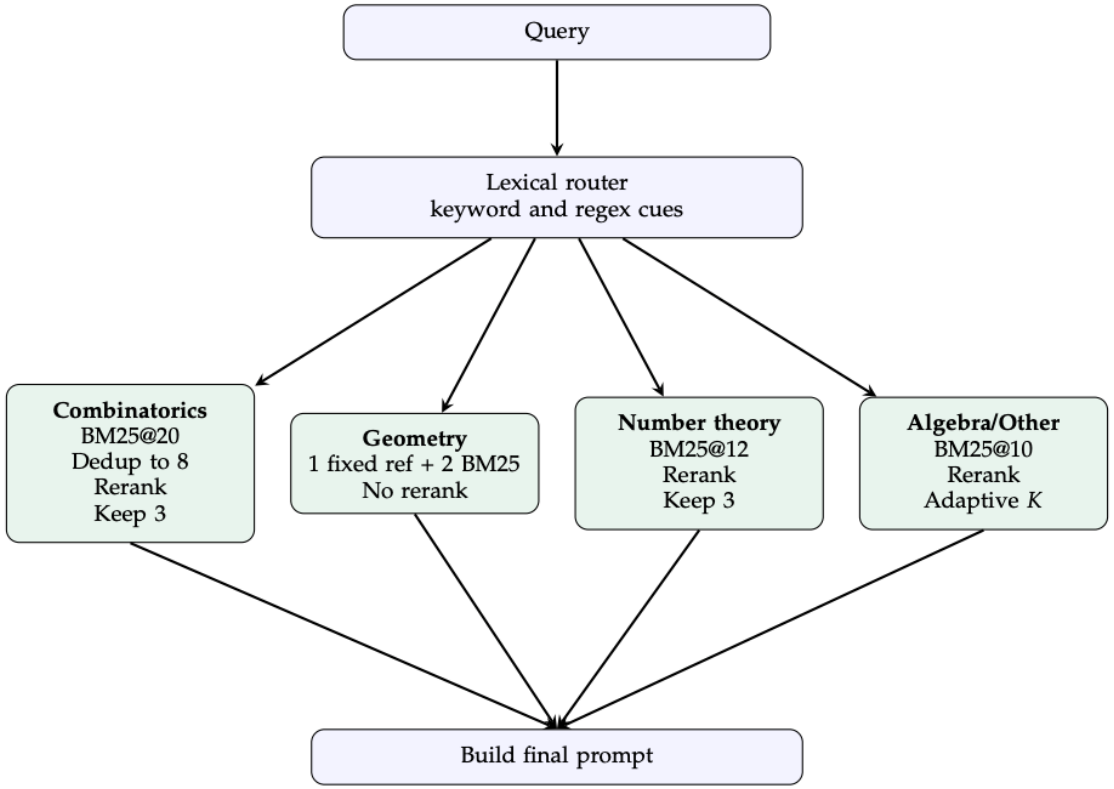


Example: Meta-Harness

Lee et al. 2026
https://arxiv.org/abs/2603.28052

- Search over harness using Claude Code
  1. Read file system containing all prior candidates' source code, execution traces, and scores.
  2. Propose new harnesses and evaluate on evaluation tasks.
  3. Store all logs in new directory and loop
- Improve IMO level problems by 4.7 points





Summary