CS 839 · FALL 2026 · LECTURE 2

From Language Models to Agents

Instruction following, reasoning models, and agent loops

Fred Sala
University of Wisconsin–Madison

September 8, 2026

The difference between models and agents?

A model: produces an output from its current context.

An agent uses model outputs to choose actions, observe consequences, and continue.

The model is central in both cases. However, the surrounding loop changes what the system can do.

Questions for today

1. Why did instruction following matter?
2. What does extra reasoning compute buy us?
3. Why connect models to tools?
4. When does the result count as an "agent"?

Outline

Basic Ingredients for Agents

- Pretrained transformers, prompting, instruction-tuning

The Basic Model Call and its Extensions

- A single prompt, its limitations, extending via reasoning, approaches to reasoning, calling tools

Basic Agentic Frameworks

- ReAct, traces, agent loops, examples

Outline

Basic Ingredients for Agents

- Pretrained transformers, prompting, instruction-tuning

The Basic Model Call and its Extensions

- A single prompt, its limitations, extending via reasoning, approaches to reasoning, calling tools

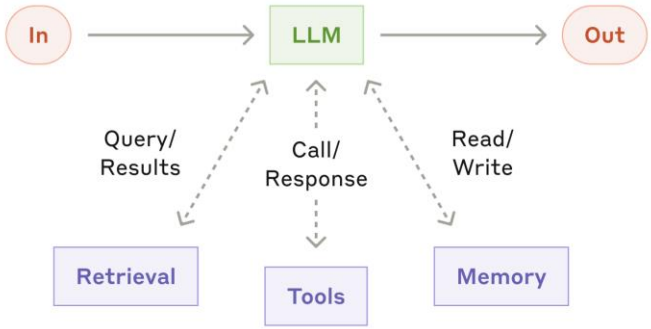
Basic Agentic Frameworks

- ReAct, traces, agent loops, examples

Building blocks: foundation models

- FMs: **pretrain** once on internet-scale broad data
  - Post-train through RL
- **Adapt** through prompting, fine-tuning, or feedback
- **Reuse** language and code representations across tasks

Main improvement: a high level of generality. Same model gets reused.



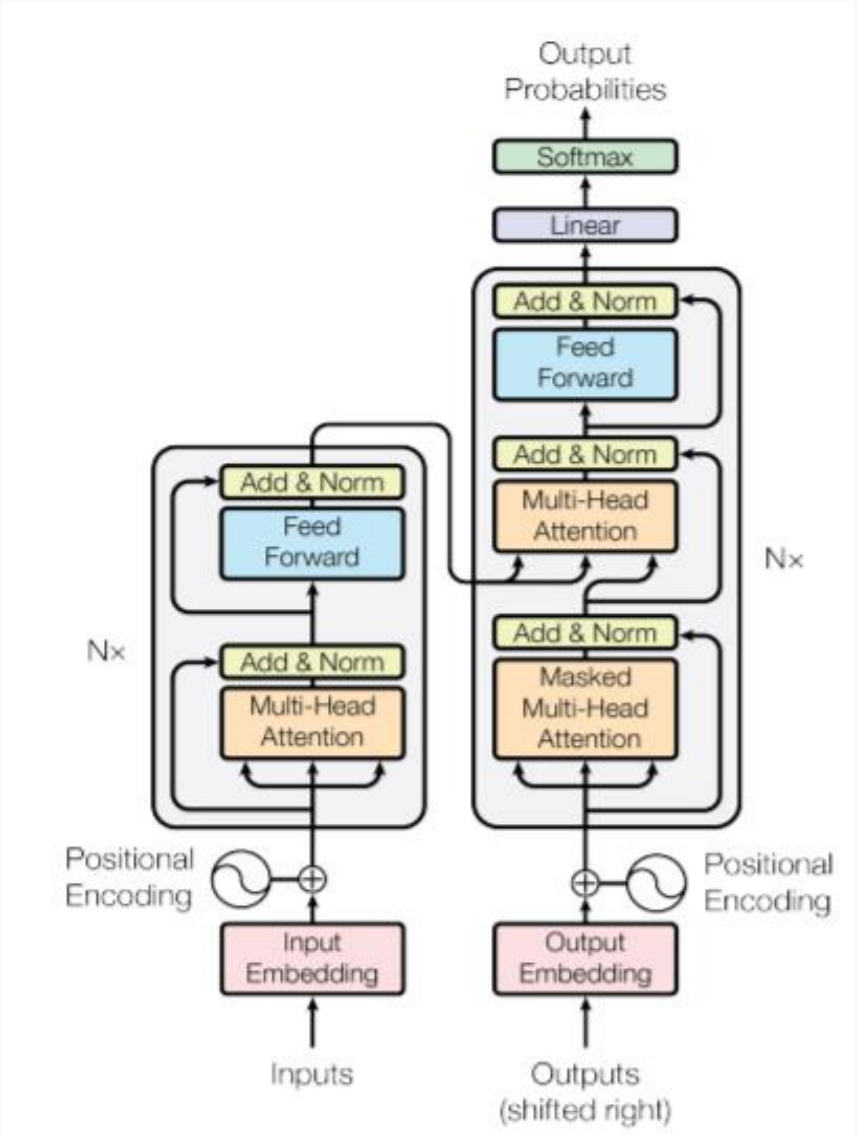
Ingredients (1): Base Model

The first step to building an agent is to obtain a "base" model.

- Training? Learning objective is prediction (next-token, etc.)
- Surprisingly, useful behaviors can emerge from this objective
- Still… difficult to use out-of-the-box.

A base model is a capable predictor: it does not (automatically) behave like a cooperative assistant.

- Example: GPT 3 (2020): an amazing base model.
- Common architecture: the Transformer (right)



Vaswani et al. '17

Interlude: prompting as an interface

How do we use an LLM? Typically by prompting. E.g.,

EXAMPLE INSTRUCTION

Given this abstract, extract the catalyst, solvent, temperature, and reported yield. Return JSON.

Zero-shot State the task.

Few-shot Include demonstrations.

In-context learning Adapt behavior without changing weights.

Ingredients (2): Instruction-following Models

How do we get a base model to follow prompts?

- We add a few steps!

Pretraining: Learn patterns in broad data.

Supervised instruction tuning Practice following tasks written as instructions.

Preference optimization Favor outputs people judge as more useful.

Implication? Natural-language requests can now function as a practical and general interface.

Break & Questions

Outline

Basic Ingredients for Agents

- Pretrained transformers, prompting, instruction-tuning

The Basic Model Call and its Extensions

- A single prompt, its limitations, extending via reasoning, approaches to reasoning, calling tools

Basic Agentic Frameworks

- ReAct, traces, agent loops, examples

A Single Model Call

Inputs Instructions, examples, retrieved text, and tool outputs.

Inference One generated sequence or structured response.

Output An answer, code, proposed action, or tool call.

If nothing outside the call stores state or executes an action, the interaction ends here.

Limits of a single call

Current evidence: Knowledge may be missing or stale.

- Example: knowledge “cut-offs” are often months/years behind present

Exact computation: Arithmetic and code execution may be unreliable.

- Why? Complicated, but related to tokenization and other issues

As a result, text alone does not search, measure, or change an environment.

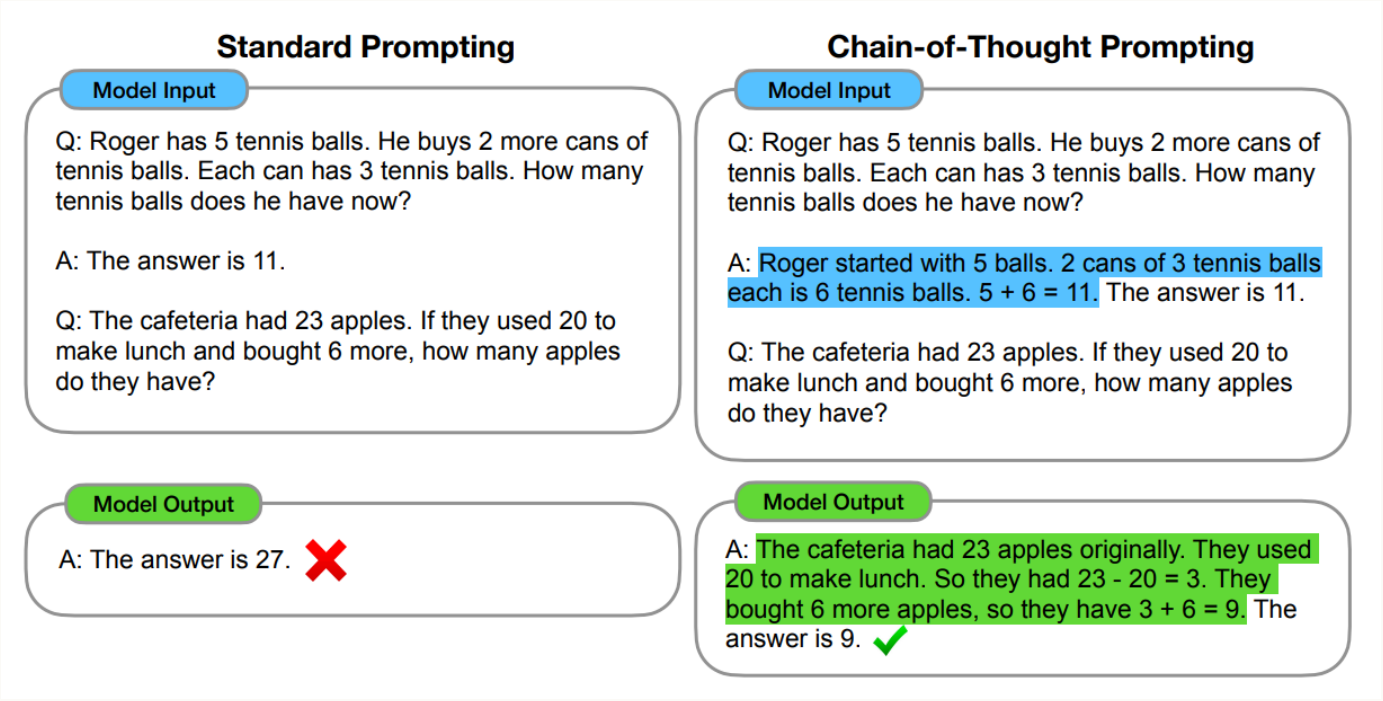
In other words, no **progress exists** unless the system records it.

Next, we’ll see ways to improve what can be done in a single call.

Explicit intermediate reasoning

Chain-of-thought prompting asks the model to produce intermediate steps before the answer.

- Helps decompose some multistep tasks
- Makes more computation available at inference time
- Creates a trace that can be inspected or sampled



Reasoning methods

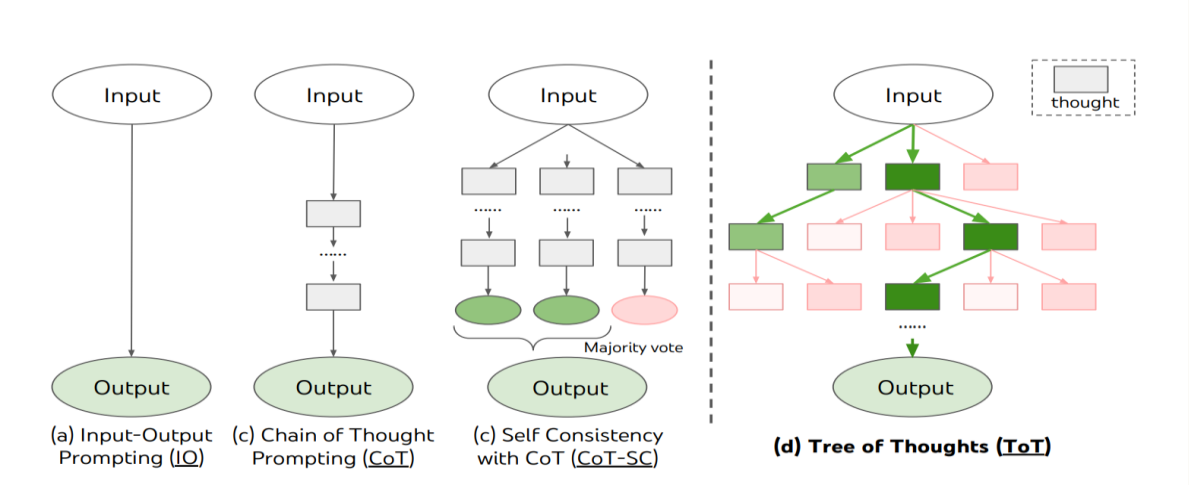
Few-shot CoT Show worked reasoning examples.

Zero-shot CoT Add a simple reasoning ”cue”, e.g., “let’s think step by step”

Self-consistency Sample several traces and aggregate answers.

Search Explore alternatives and backtrack.

These methods change inference. But… they do not give the model access to the world.



Yao et al ‘23

Limits of reasoning traces

Can help: math, symbolic manipulation, decomposition, and structured coding tasks.

Can fail: The trace may sound plausible while the conclusion is wrong.

Can mislead: A readable explanation does not prove that the model used a faithful process.

- A current major issue in the research space; CoT monitoring may be insufficient to provide evidence of safe operation.

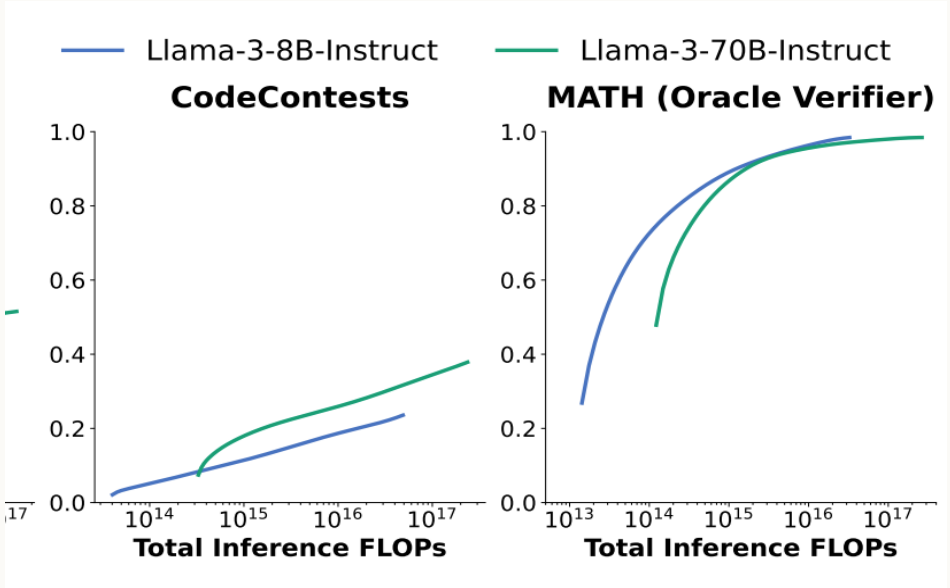
For us: a reasoning trace is a something to check but not fully believe in.

Test-time compute

Standard call: One attempt with a fixed decoding budget.

Reasoning model: More inference work through longer traces, revision, or internal search.

- **Traditional scaling** spends more compute during training
- **Test-time scaling** spends more compute on a particular problem
- The useful amount depends on the task and the quality of feedback



Brown et al ‘24

Best-of-N

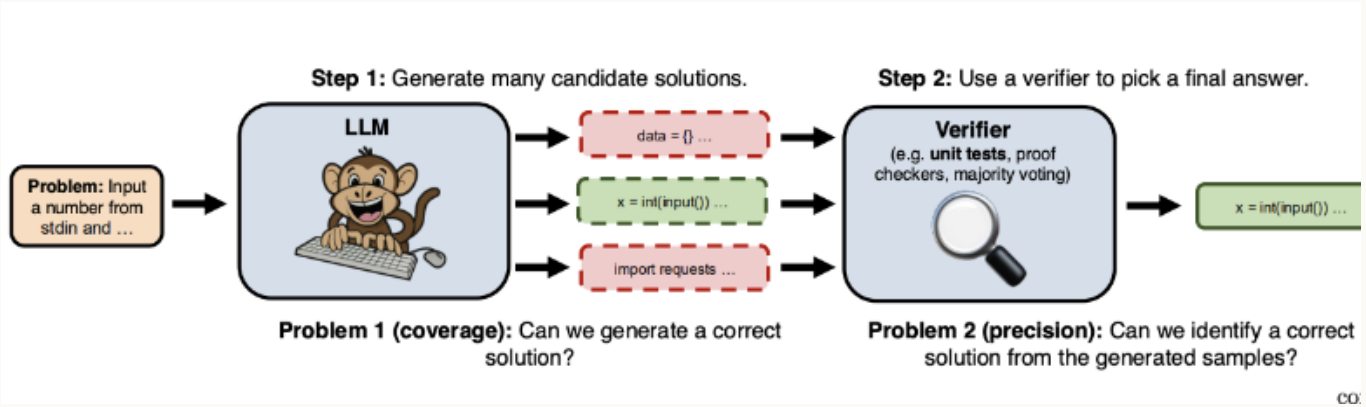
Repeated sampling increases the chance that one candidate is correct.

Concepts:

Coverage. Did we generate a good candidate?

Precision, Can we identify it?

Verifier: a tool to perform this identification.



What's a verifier in our setting?

Depends on the data modality:

Code: Tests, type checks, and reproducible outputs.

Data analysis: Held-out data, uncertainty estimates, and invariants.

Literature: Source retrieval, quotation checks, and citation support.

Experiments: Measurements, controls, and physical constraints.

Major challenge: deciding what evidence **genuinely distinguishes between candidates**.

External tools

Let's further extend our model's abilities via **tools.** We can:

- **Search** for information not in context
- **Run** exact computations or simulations
- **Read** and **write** structured artifacts
- **Interact** with services or instruments (in our setting, scientific instruments)

Your task is to add calls to a Question Answering API to a piece of text. The questions should help you get information required to complete the text. You can call the API by writing "[QA(question)]" where "question" is the question you want to ask. Here are some examples of API calls:

Input: Joe Biden was born in Scranton, Pennsylvania.

Output: Joe Biden was born in **[QA("Where was Joe Biden born?")]** Scranton, **[QA("In which state is Scranton?")]** Pennsylvania.

Input: Coca-Cola, or Coke, is a carbonated soft drink manufactured by the Coca-Cola Company.

Output: Coca-Cola, or **[QA("What other name is Coca-Cola known by?")]** Coke, is a carbonated soft drink manufactured by **[QA("Who manufactures Coca-Cola?")]** the Coca-Cola Company.

Input: x

Output:

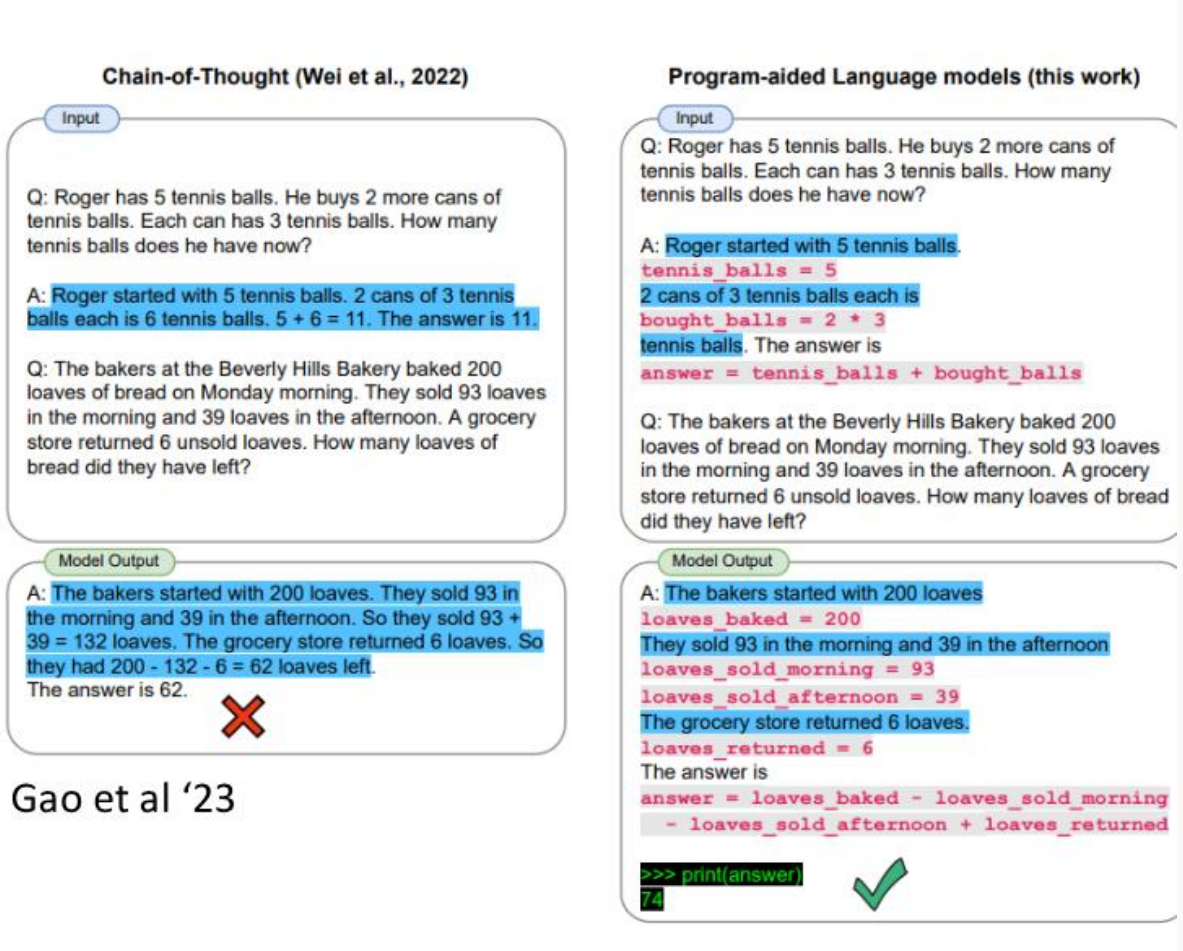
Example: program-aided language models

Let the model express part of the reasoning as code, then let an interpreter compute the result.

Language decomposes the problem

Interpreter executes exactly

Outputs become new context



Four decisions in a tool interaction

1. **Choose:** Is a tool needed?
2. **Construct:** Are the arguments valid?
3. **Interpret:** What does the returned observation mean?
4. **Decide:** What should happen next?

Correct execution does not imply that the tool choice, query, or interpretation was correct.

- Many mistakes of this type made in agentic operation!

Tool failures

Common failure points:

- Call the wrong tool
- Use invalid arguments or a malformed query
- Misread an error as evidence
- Chain valid calls into an invalid conclusion

The system must reason about both the task and the interface.

```
================================================ tool_calls =================================================
------------------------------------------------- tool -------------------------------------------------------
read_query

------------------------------------------------- input ------------------------------------------------------
{"query": "SELECT DISTINCT NAICS_Code, Description FROM naics WHERE Description LIKE '%flour milling%' OR Description
LIKE '%wheat flour%' OR Description LIKE '%corn flour%'"}

================================================ read_query =================================================
Error: ToolException('Error executing tool read_query: SQLite error: no such column: NAICS_Code')  Please fix your
mistakes.
```

Break & Questions

Outline

Basic Ingredients for Agents

- Pretrained transformers, prompting, instruction-tuning

The Basic Model Call and its Extensions

- A single prompt, its limitations, extending via reasoning, approaches to reasoning, calling tools

Basic Agentic Frameworks

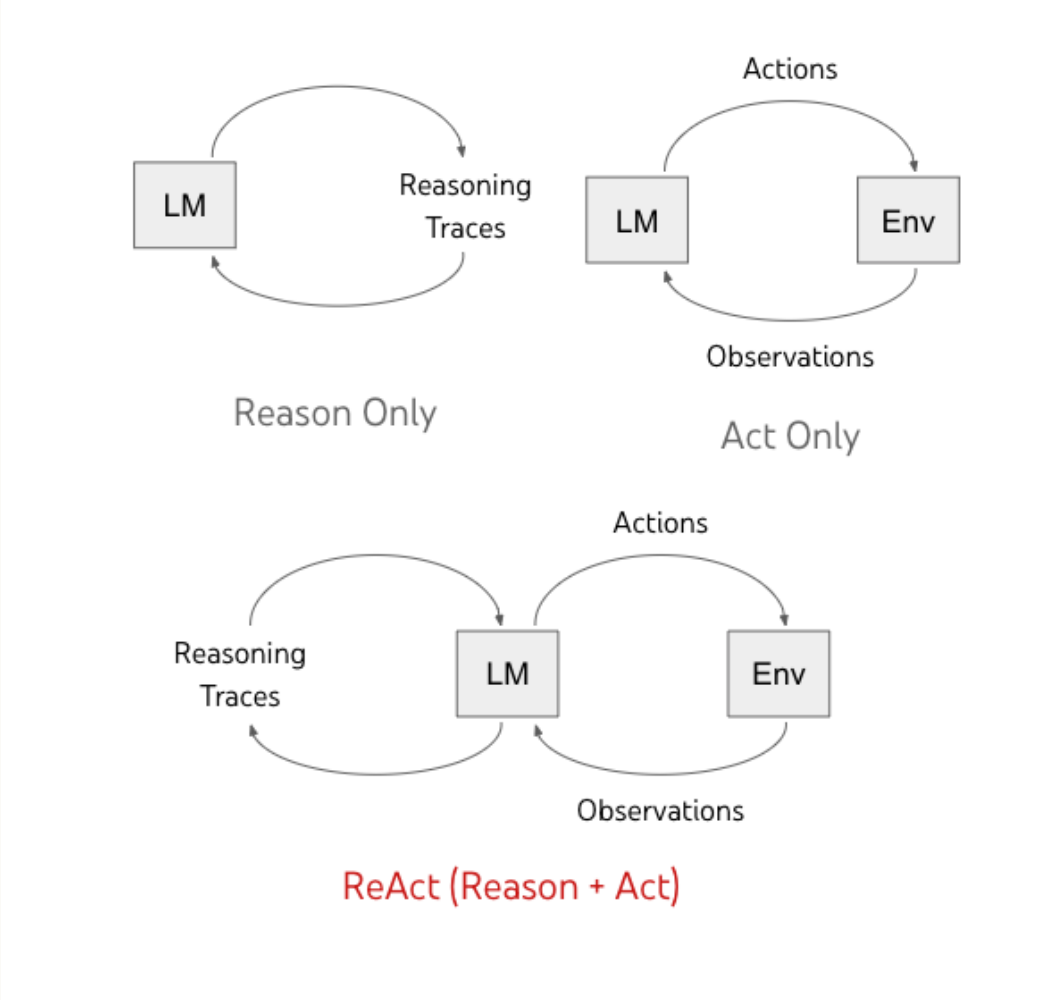
- ReAct, traces, agent loops, examples

ReAct

ReAct interleaves reasoning with actions and observations.

- Reason about the next information need
- Act through a tool
- Condition the next step on the observation

The environment can correct, surprise, or constrain the model.



Example of a scientific ReAct trace

Question: Does paper X report improved stability over baseline Y?

Reason: I need the experimental section and the metric definition.

Act: Search the paper, then retrieve the relevant table and caption.

Observe: The table reports a different metric and a shorter evaluation horizon.

Revise: The claimed comparison lacks support. Search for a matched evaluation.

Model call and agent system: key differences

	Model call	Agent system
Goal	Encoded in one prompt	Maintained across steps
State	Current context	Context plus workflow state
Actions	Generate output	Select and execute operations
Feedback	Usually none during generation	Observations affect later choices
Termination	End of generation	Success, stop rule, budget, or human decision

The minimal agent loop

Goal: What counts as progress?

State: What is known now?

Action: What can the system do?

Observation: What happened?

Decision: Should the system continue, revise, or stop?

An agent implements a policy through repeated model calls plus external state and actions.

Stop rules

Success: A verifiable condition has been met.

Budget: Time, tokens, money, or tool calls are exhausted.

Uncertainty: Evidence is insufficient or contradictory.

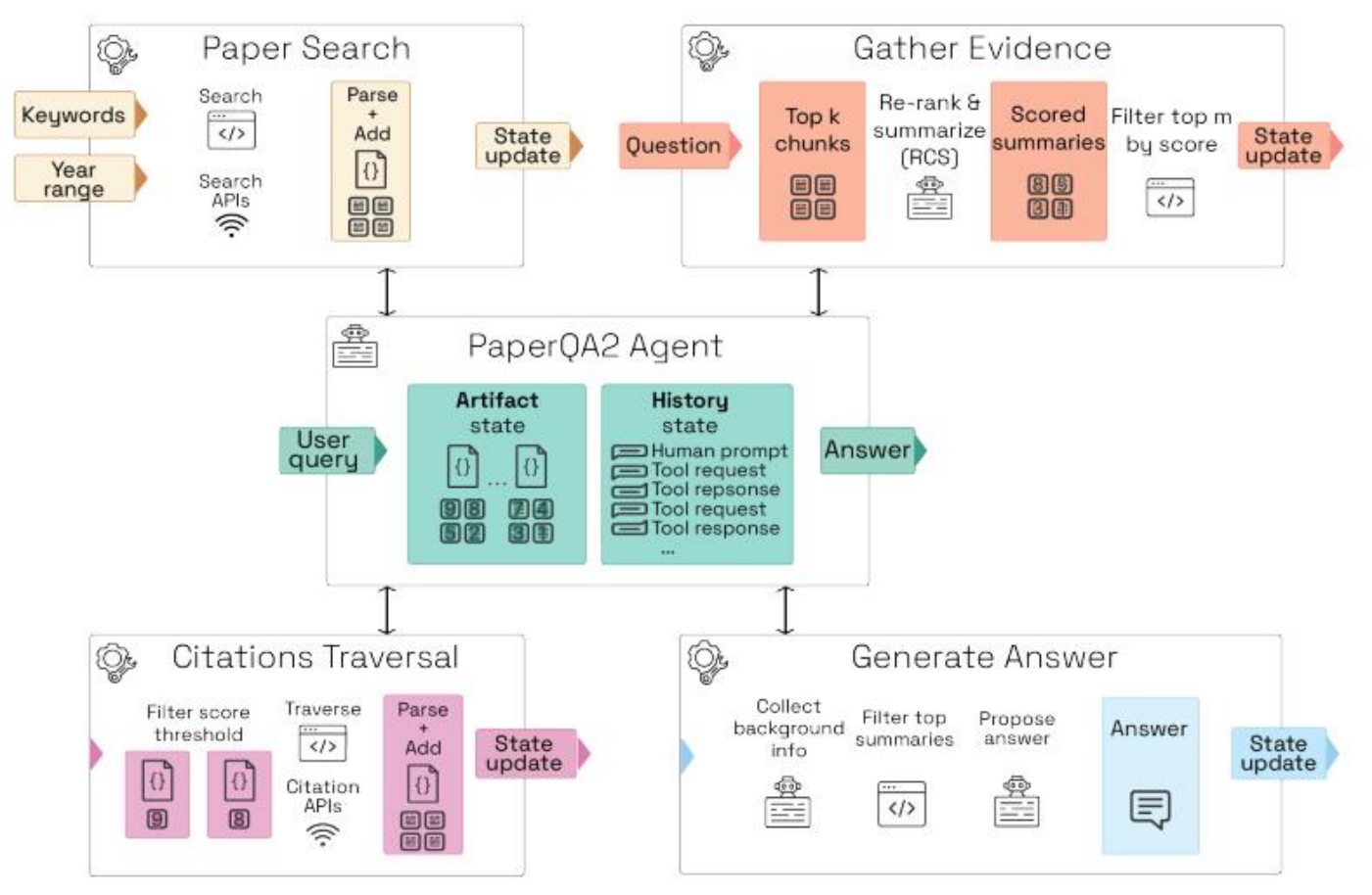
Permission: The next action requires human approval.

Without a stop rule, the agent will go on forever!

Evidence-synthesis agent

Goal: Answer a scientific question with traceable evidence.

1. Form search queries
2. Retrieve candidate papers
3. Extract passages and metadata
4. Compare claims and evidence
5. Draft an answer with citations
6. Verify that citations support the claims



Control surfaces for scientific agents

Allowed Read papers, run sandboxed code, and analyze provided data.

Approval required Spend money, access credentials, change shared artifacts, or operate equipment.

Forbidden Fabricate evidence, conceal provenance, or bypass safety constraints.

Capabilities and permissions are separate design choices.

Takeaways

- **Instruction following** turns natural language into a practical interface
- **Reasoning methods** spend more inference compute on a problem, but do not create more evidence (because they can't interact!)
- Tools provide external information, exact computation, and more
- An agent repeatedly chooses actions using state, observations, and a stopping rule

Key design question: What should remain inside the model call, and what should become an explicit system component?

Next lecture

LECTURE 3

Agent Architectures and Frameworks

Orchestration, state, control flow, and debugging

How should the loop be represented?
Where should state live?
How do we inspect failures?