

香港中文大學(深圳)
The Chinese University of Hong Kong, Shenzhen

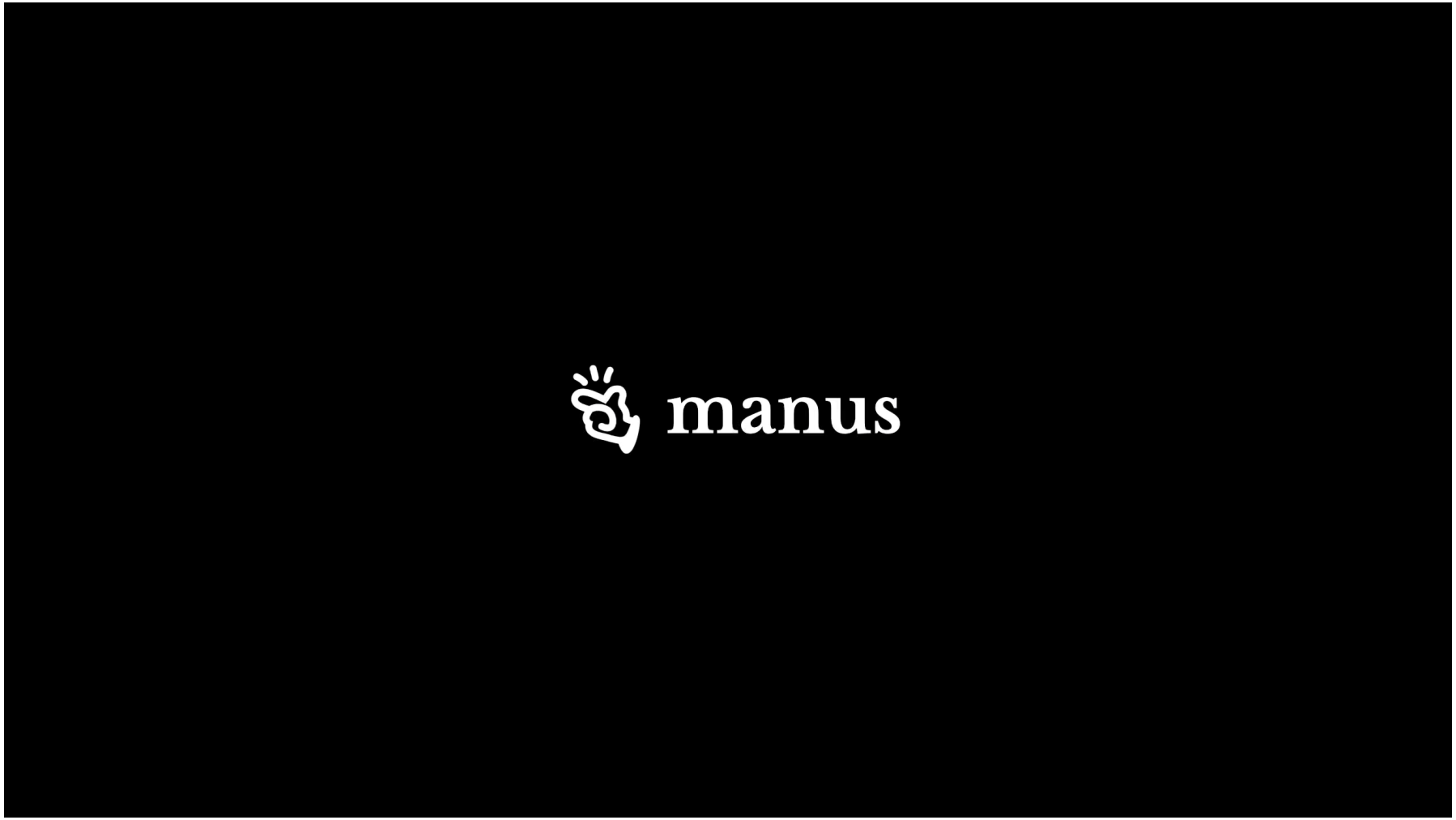
CSC5051/MDS5110/CSC6052 : Natural Language Processing

Lecture 7: LLM Agents and Harness Engineering

Fall 2026

Benyou Wang
School of Data Science

Manus: A General-Purpose Agent



https://manus.im/

Everyone Should Get Used to Working with Agents

- In Silicon Valley, most companies now operate **with fewer than 30 employees**, as AI has dramatically increased coding efficiency.
- They often prefer hiring **new graduates** over experienced engineers, since the former are more **open to leveraging** APIs and AI-assisted tools.
- A new hire is considered unqualified if they cannot effectively solve problems with the help of AI (such as Cursor).
- Today, nearly everyone is a **full-stack engineer** — working alongside AI to focus more on defining applications with a broader vision that truly benefits users.

Credits to a CUHKSZ graduate (now running startups in silicon valley) Casey Zhou

Why? From LLMs to Agents

Why?



Credit: a CUHK-SZ graduate (now running startups in silicon valley) Casey Zhou

Contents

Foundations — *from prompts to agents*

1 From Prompts to Agents
ICL · CoT · least-to-most · ToT · PoT · self-consistency · self-refine

2 What Is an Agent?
From ELIZA and RL agents to LLM agents · the agent loop

3 The Four Elements of an Agent
ReAct · LLM+P · function calling · RAG · embodied agents

Harness engineering — *everything except the model*

4 Harness Engineering: Principles
Agent = Model + Harness · harness tax · context · skills & MCP · guides & sensors

5 Harness Engineering in Practice
Claude Code · Codex · OpenClaw · A2A · fault tolerance · evals · bitter lesson

Applications — *agents in the world*

6 Agents in the Wild
Coding, games, markets · WebShop · WebArena · SWE-bench

7 Agent Societies and Economic World Models
Individual and social simulation · twin market · twin hospital · policy sandbox

The equation of this lecture

Agent = Model + Harness

What we engineer

Prompt engineering
what we say

↓ Context engineering
what the model sees

↓ Harness engineering
the system it runs in

1 From Prompts to Agents

Every prompting trick — decomposition, tools, self-checking — later becomes a component of an agent.

What Is In-Context Learning?

Circulation revenue has increased by 5% in Finland. // Positive

Panostaja did not disclose the purchase price. // Neutral

Paying off the national debt will be extremely painful. // Negative

The company anticipated its operating profit to improve. // _____



Circulation revenue has increased by 5% in Finland. // Finance

They defeated … in the NFC Championship Game. // Sports

Apple … development of in-house chips. // Tech

The company anticipated its operating profit to improve. // _____

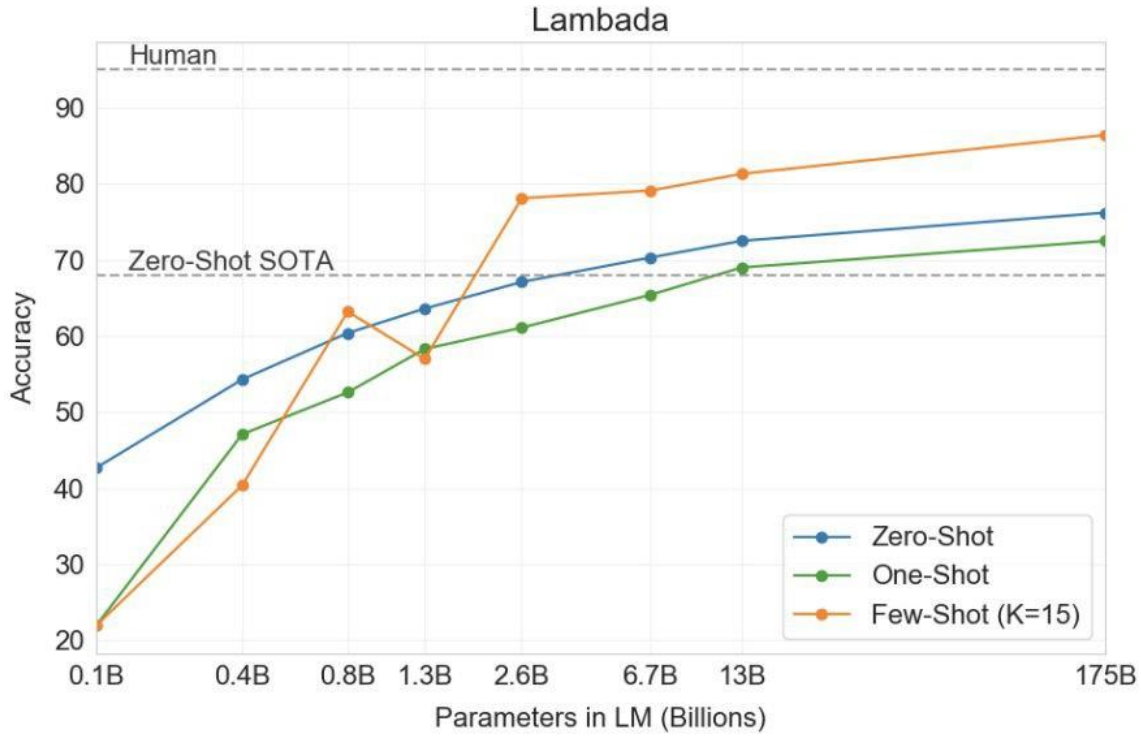


2

https://ai.stanford.edu/blog/understanding-incontext/

What Can In-Context Learning Do?

- No parameter tuning need
- Only need few examples for downstream tasks
- GPT-3 improved SOTA on LAMBADA(last word prediction task) by 18%!



Works like magic!

Chain-of-Thought (CoT) Prompting

Standard Prompting

Input

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The answer is 27. ✗

Chain of Thought Prompting

Input

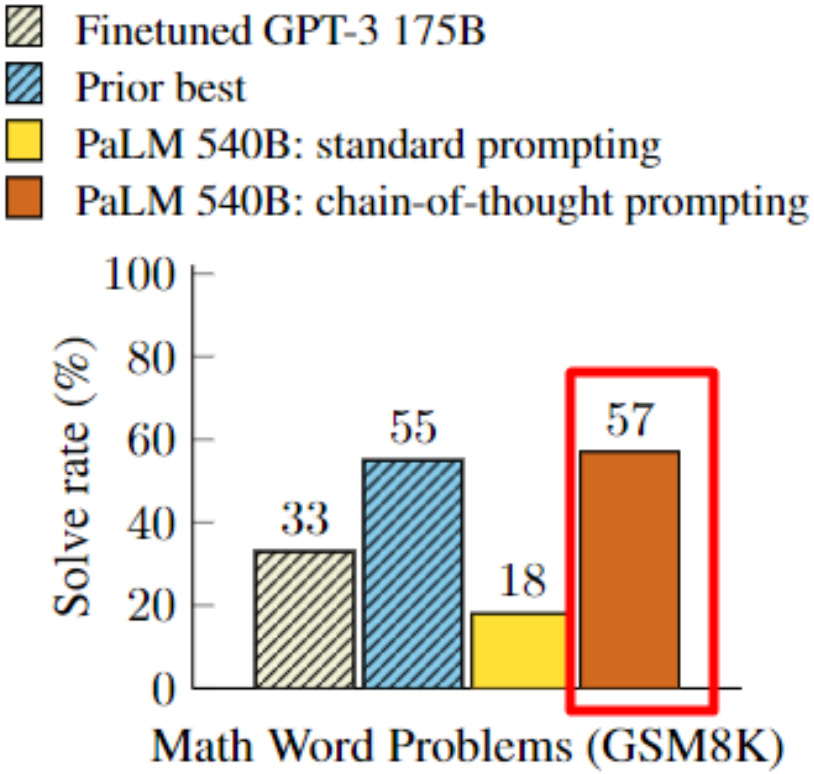
Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?

A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: The cafeteria had 23 apples. If they used 20 to make lunch and bought 6 more, how many apples do they have?

Model Output

A: The cafeteria had 23 apples originally. They used 20 to make lunch. So they had 23 - 20 = 3. They bought 6 more apples, so they have 3 + 6 = 9. The answer is 9. ✓



<input, intermediate results, output>

- decompose into easier intermediate steps
- interpretable

Zero-Shot CoT: “Let's Think Step by Step”

Examples

(a) Few-shot

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The answer is 8. ✗

(b) Few-shot-CoT (Wei et al., 2022)

Q: Roger has 5 tennis balls. He buys 2 more cans of tennis balls. Each can has 3 tennis balls. How many tennis balls does he have now?
A: Roger started with 5 balls. 2 cans of 3 tennis balls each is 6 tennis balls. 5 + 6 = 11. The answer is 11.

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A:

(Output) The juggler can juggle 16 balls. Half of the balls are golf balls. So there are 16 / 2 = 8 golf balls. Half of the golf balls are blue. So there are 8 / 2 = 4 blue golf balls. The answer is 4. ✓

CoT Examples

Step-by-step Answer

(c) Zero-shot

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: The answer (arabic numerals) is

(Output) 8 ✗

(d) Zero-shot-CoT (Kojima et al., 2022)

Q: A juggler can juggle 16 balls. Half of the balls are golf balls, and half of the golf balls are blue. How many blue golf balls are there?
A: ***Let's think step by step.***

(Output) There are 16 balls in total. Half of the balls are golf balls. That means that there are 8 golf balls. Half of the golf balls are blue. That means that there are 4 blue golf balls. ✓

Step-by-step Answer

Least-to-Most Prompting: Decompose, then Solve

Explicitly decompose into subquestions

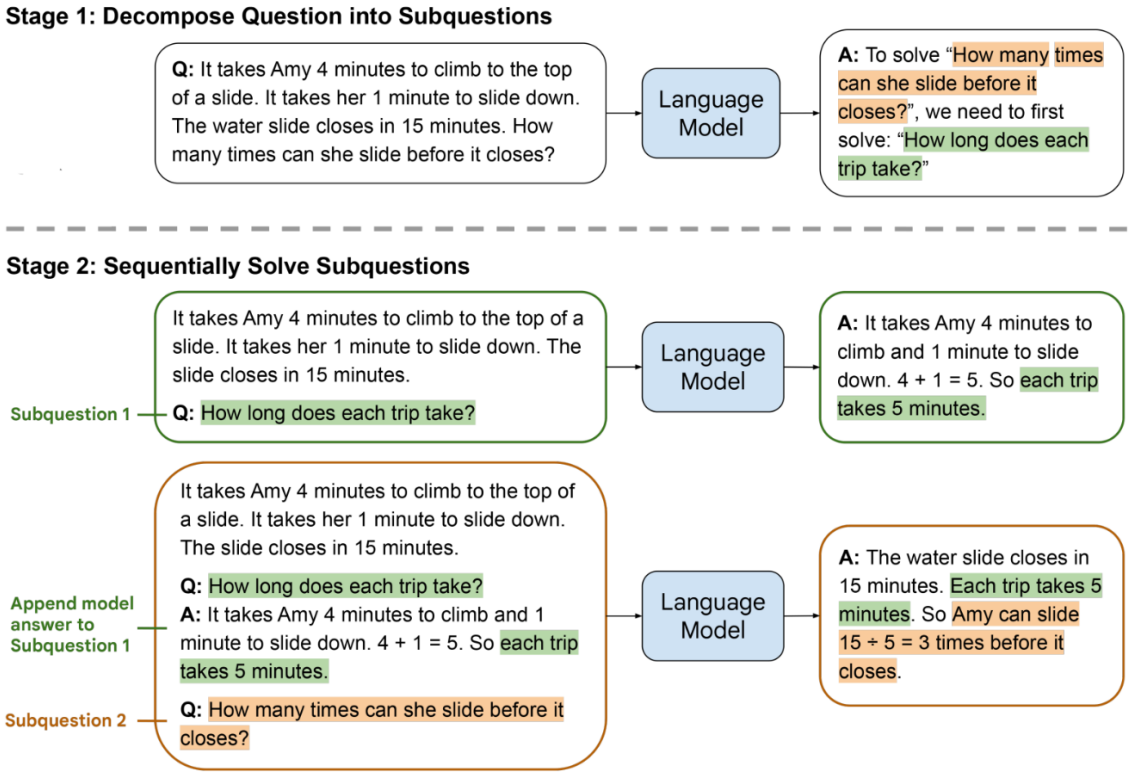
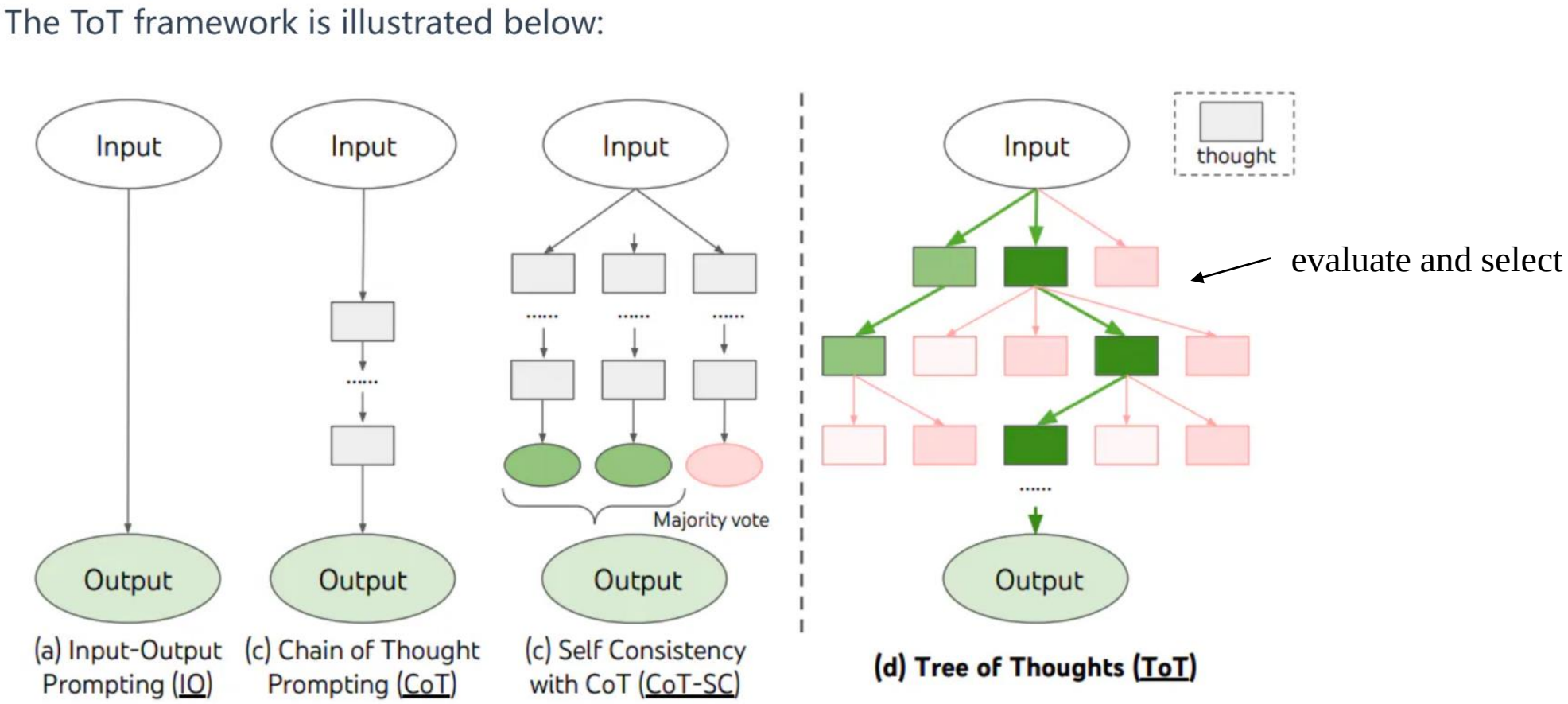


Figure 1: Least-to-most prompting solving a math word problem in two stages: (1) query the language model to decompose the problem into subproblems; (2) query the language model to sequentially solve the subproblems. The answer to the second subproblem is built on the answer to the first subproblem. The demonstration examples for each stage's prompt are omitted in this illustration.

Wei, Jason, et al. "Chain-of-thought prompting elicits reasoning in large language models." Advances in Neural Information Processing Systems 35 (2022): 24824-24837.

Tree of Thoughts (ToT): Search over Thoughts

Explore over units of text that serve as intermediate steps



Yao, Shunyu, et al. "Tree of thoughts: Deliberate problem solving with large language models." arXiv preprint arXiv:2305.10601 (2023).

Program of Thoughts (PoT): Let Code Do the Math

Output Python programs and call Python interpreter to calculate the answers

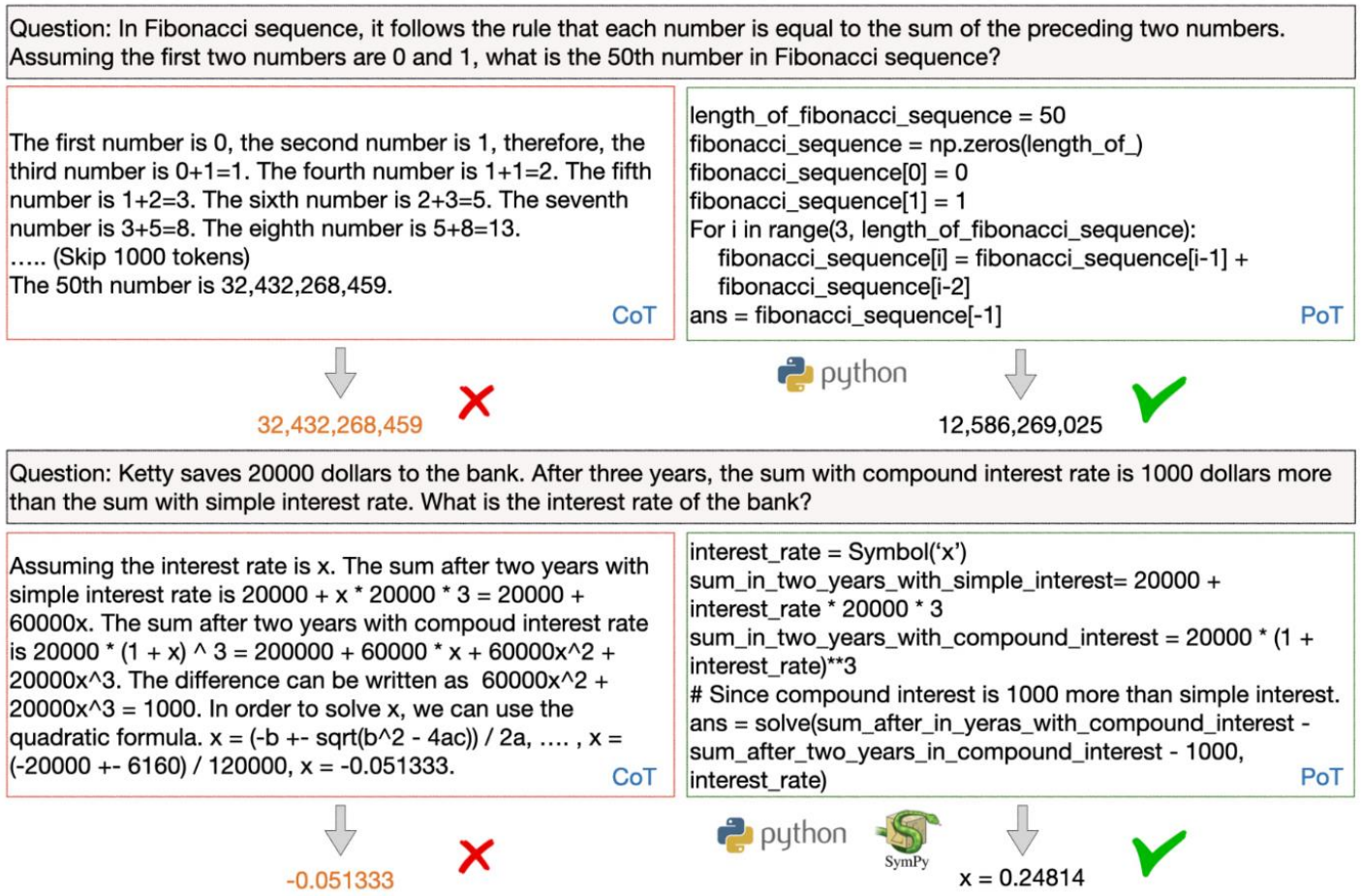


Figure 1: Comparison between Chain of Thoughts and Program of Thoughts.

Chen, Wenhu, et al. "Program of thoughts prompting: Disentangling computation from reasoning for numerical reasoning tasks." arXiv preprint arXiv:2211.12588 (2022).

Self-Consistency: Sample Many, Vote Once

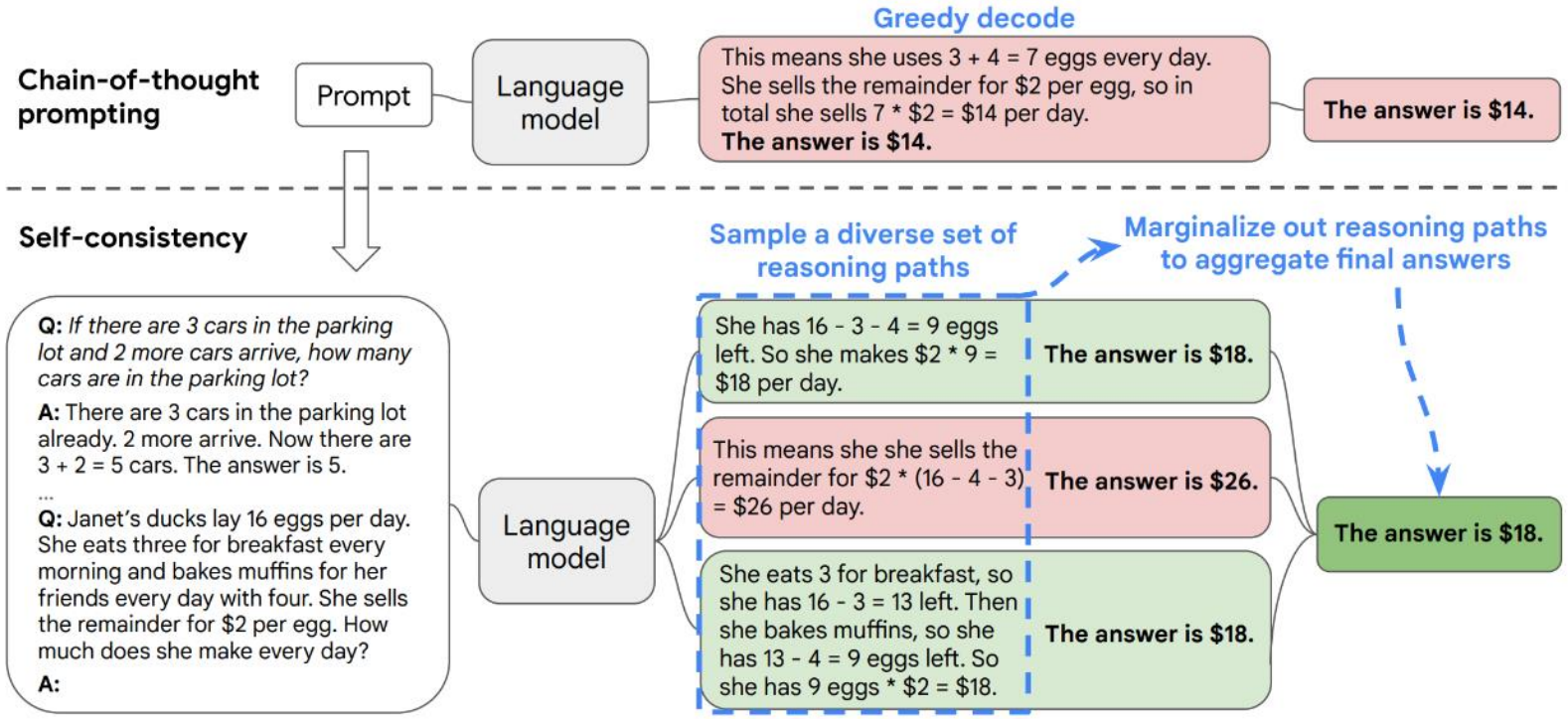


Figure 1: The self-consistency method contains three steps: (1) prompt a language model using chain-of-thought (CoT) prompting; (2) replace the "greedy decode" in CoT prompting by sampling from the language model's decoder to generate a diverse set of reasoning paths; and (3) marginalize out the reasoning paths and aggregate by choosing the most consistent answer in the final answer set.

Wang, Xuezhi, et al. "Self-consistency improves chain of thought reasoning in language models." arXiv preprint arXiv:2203.11171 (2022).

Self-Refine: Feedback, then Refine

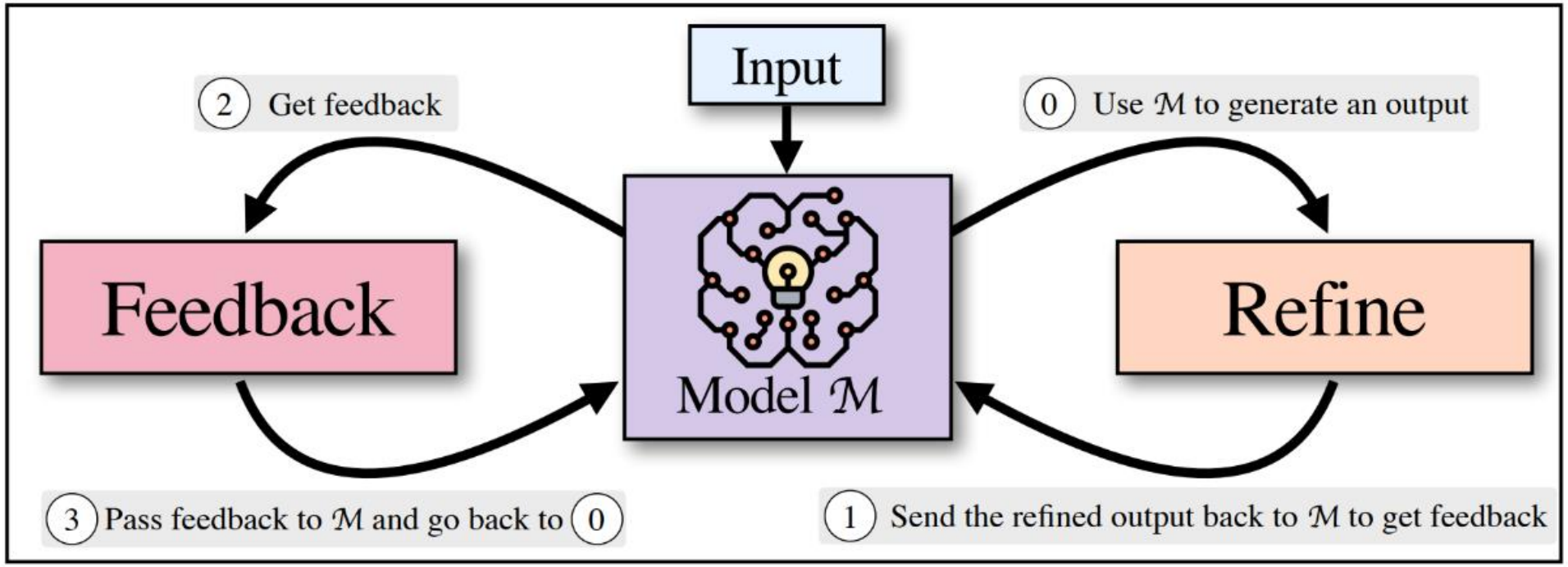


Figure 1: SELF-REFINE starts by taking an initially generated output (⓪), and passing it back to the same model $\mathcal{M}$ (①) to get feedback (②); feedback on the initial output is passed back to the model (③), to iteratively refine (⓪) the previously generated output. SELF-REFINE is instantiated with a powerful language model such as GPT-3.5 and does not involve human assistance.

Madaan, Aman, et al. "Self-refine: Iterative refinement with self-feedback." arXiv preprint arXiv:2303.17651 (2023).

Should We Use All These Techniques?

- Usually CoT can perform well under many situations
- Accuracy vs **Cost**:
  - Additional techniques need more computational sources (self-consistency) or additional data processing(PoT) although these techniques can usually improve the performance.
  - the trade off depends on the real application.

From Prompting Tricks to Agent Components

Prompting trick (one model call)	Agent component (many calls)
Least-to-most, ToT — decompose and search	**Planning**
PoT, PAL — let code compute	**Tools**
In-context examples — show what good looks like	**Memory & skills**
Self-consistency, verifiers — check before answering	**Verification (sensors)**
Self-refine — feedback, then fix	**The agent loop**

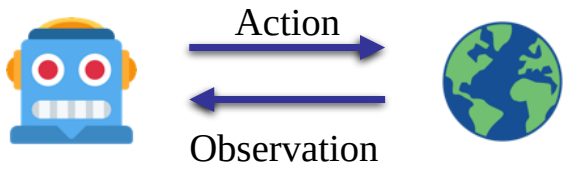
An agent is these tricks running in a loop — with real tools and real feedback instead of imagined ones.

Wei et al. (2022) · Zhou et al. (2023) · Yao et al. (2023) · Chen et al. (2022) · Wang et al. (2023) · Madaan et al. (2023).

2 What Is an Agent?

An agent is a model that acts in an environment and learns from feedback — an old idea that LLMs finally made general.

What Is an "Agent"?



- An "**intelligent**" system that interacts with some "**environment**"
  - Physical environments: robot, autonomous car, …
  - Digital environments: DQN for Atari, Siri, AlphaGo, …
  - Humans as environments: chatbot
  -
- Define "agent" by defining "intelligent" and "environment"
  - It changes over time!
  - Exercise question: how would you define "intelligent"?

Agents Are Everywhere

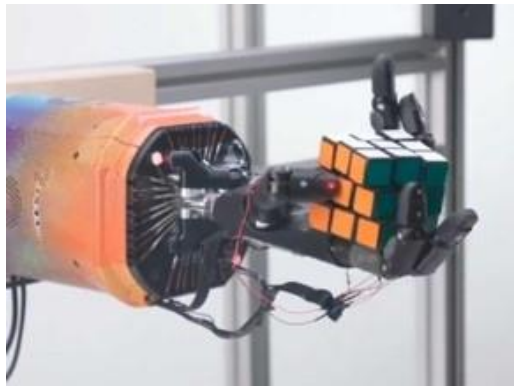








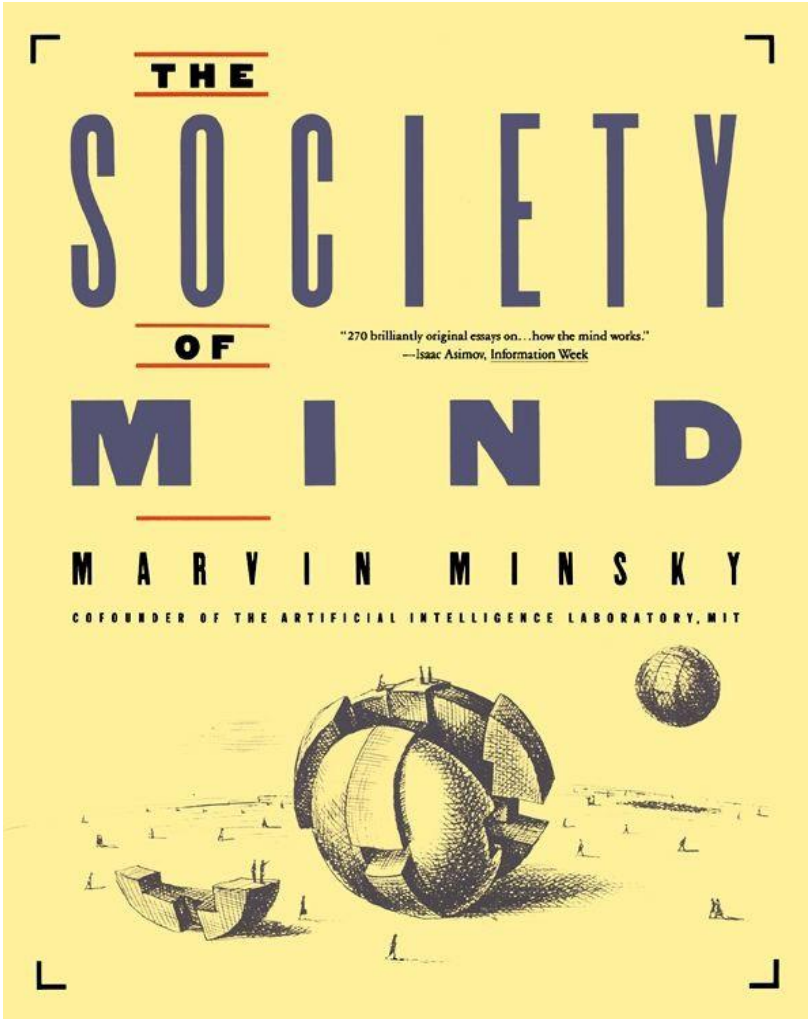




Agents Are Not New: The Society of Mind (1986)

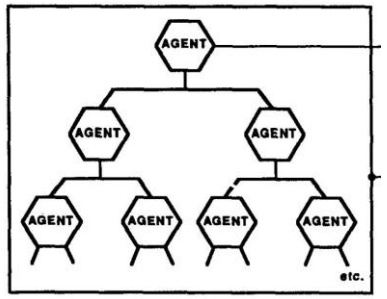
Function:	*How do agents work?*
Embodiment:	*What are they made of?*
Interaction:	*How do they communicate?*
Origins:	*Where do the first agents come from?*
Heredity:	*Are we all born with the same agents?*
Learning:	*How do we make new agents and change old ones?*
Character:	*What are the most important kinds of agents?*
Authority:	*What happens when agents disagree?*
Intention:	*How could such networks want or wish?*
Competence:	*How can groups of agents do what separate agents cannot do?*
Selfness:	*What gives them unity or personality?*
Meaning:	*How could they understand anything?*
Sensibility:	*How could they have feelings and emotions?*
Awareness:	*How could they be conscious or self-aware?*

Agent from 1986



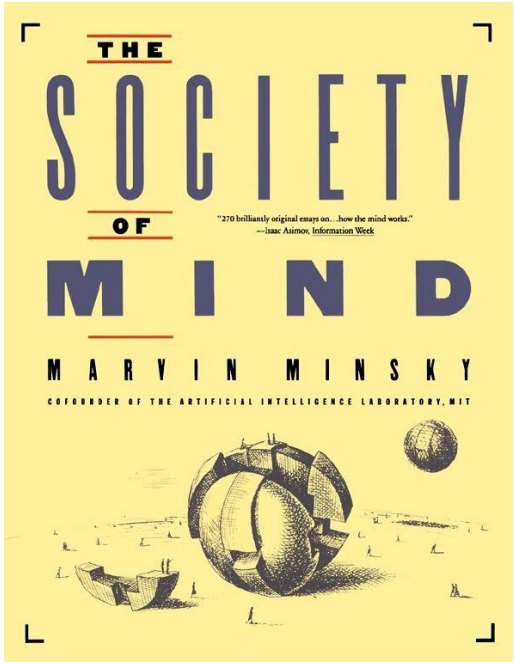
The Society of Mind (1986)

- *Agents* are **mindless** processes
- *Agent* by itself can only do some simple things
- Joining these *agents* in *societies* leads to true *intelligence*

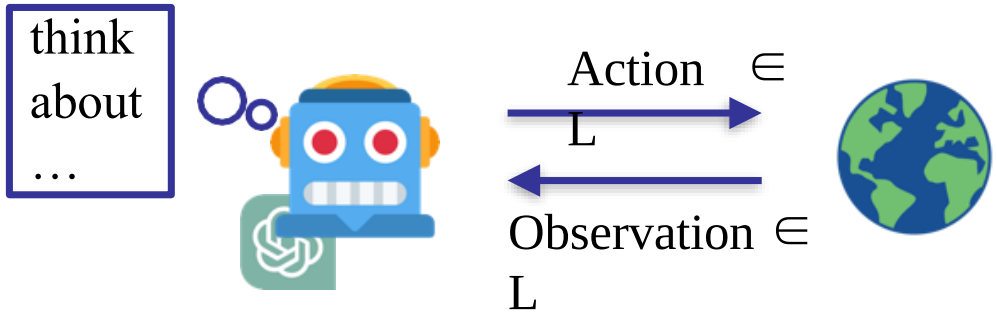


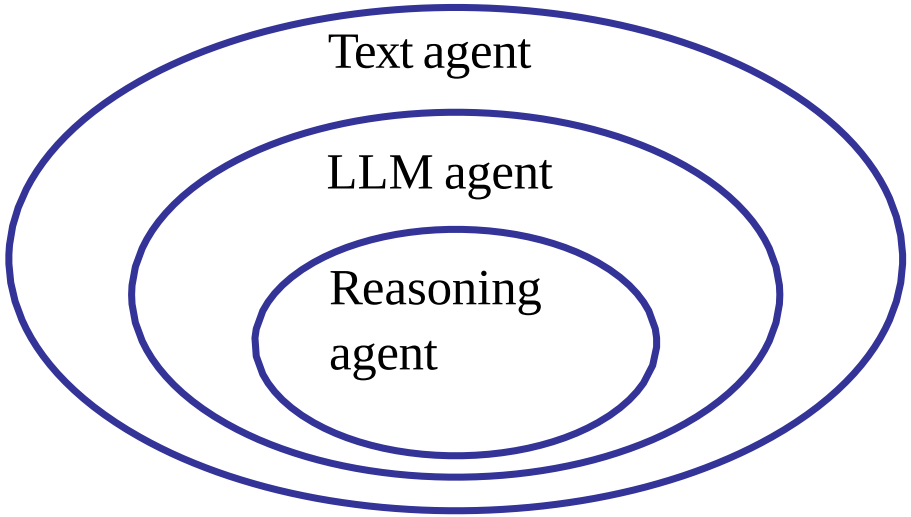
What magical trick makes us intelligent? The trick is that there is no trick. The power of intelligence stems from our vast diversity, not from any single, perfect principle.

—Marvin Minsky, The Society of Mind, p. 308



What Is an "LLM Agent"? Three Levels





- Level 1: Text agent
  - Uses text action and observation
  - Examples: ELIZA, LSTM-DQN
- Level 2: LLM agent
  - Uses LLM to act
  - Examples: SayCan, Language Planner
- Level 3: Reasoning agent
  - Uses LLM to reason to act
  - Examples: ReAct, AutoGPT
  - **The key focus of the field and the talk**

2
3

ELIZA (1966): A Text Agent by Hand-Written Rules

```
  Welcome to
                EEEEEE  LL      IIII   ZZZZZZ   AAAAA
                EE      LL       II       ZZ   AA   AA
                EEEEE   LL       II     ZZZ    AAAAAAA
                EE      LL       II    ZZ      AA   AA
                EEEEEE  LLLLLL  IIII ZZZZZZ    AA   AA

  Eliza is a mock Rogerian psychotherapist.
  The original program was described by Joseph Weizenbaum in 1966.
  This implementation by Norbert Landsteiner 2005.


ELIZA: Is something troubling you ?
YOU:   Men are all alike.
ELIZA: What is the connection, do you suppose ?
YOU:   They're always bugging us about something or other.
ELIZA: Can you think of a specific example ?
YOU:   Well, my boyfriend made me come here.
ELIZA: Is it important to you that your boyfriend made you come here ?
YOU:   He says I'm depressed much of the time.
ELIZA: I am sorry to hear that you are depressed.
YOU:   It's true. I am unhappy.
ELIZA: Can you explain what made you unhappy ?
YOU:
```

Symbolic AI agent

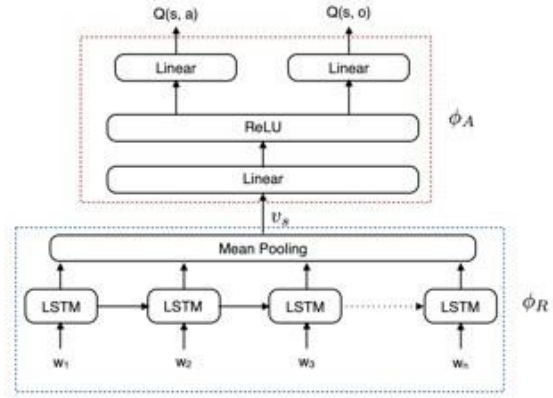
- Domain specific!
- Requires manual design
- Cannot work beyond simple domains

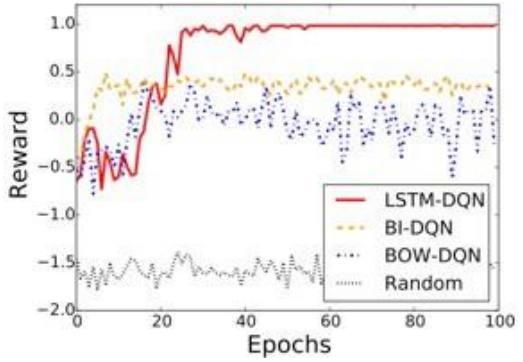
LSTM-DQN (2015): A Text Agent Trained with RL

State 1: The old bridge
You are standing very close to the bridge's eastern foundation. If you go east you will be back on solid ground ... The bridge sways in the wind.

Command: **Go east**

State 2: Ruined gatehouse
The old gatehouse is near collapse. Part of its northern wall has already fallen down ... East of the gatehouse leads out to a small open area surrounded by the remains of the castle. There is also a standing archway offering passage to a path along the old southern inner wall.
Exits: Standing archway, castle corner, Bridge over the abyss





- Domain specific!
- Requires scalar reward signals
- Requires extensive training

Language understanding for text-based games using deep reinforcement learning. EMNLP 2015.

Deep RL agent

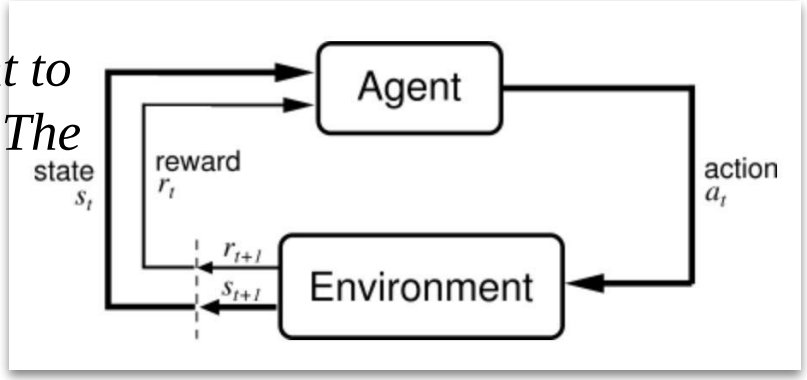
Agents in Reinforcement Learning

- The *learner* and *decision-maker* is called the *Agent*
- *Agent* interacts with an *environment* through actions
- Environment presents new *states* and *rewards*

A learning agent must be able to sense the state of its environment to some extent and must be able to take actions that affect the state. The agent also must have a goal or goals relating to the state of the environment.

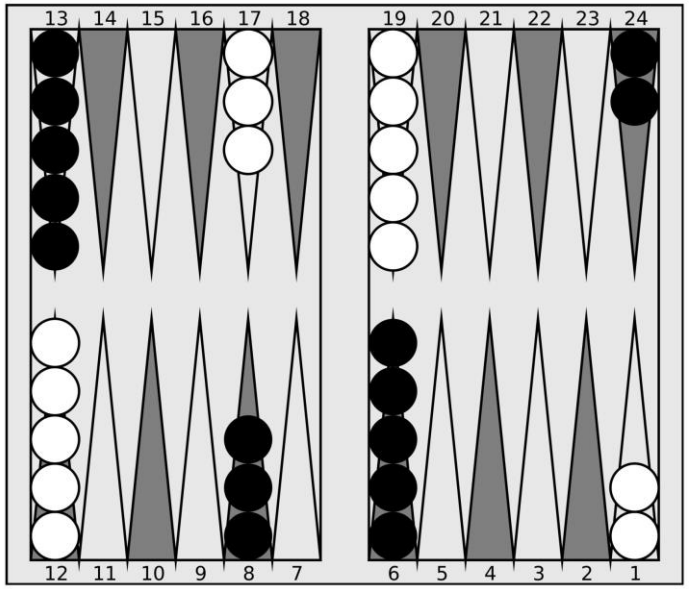
—Richard S. Sutton and Andrew G. Barto,

Reinforcement Learning: An Introduction, p. 2

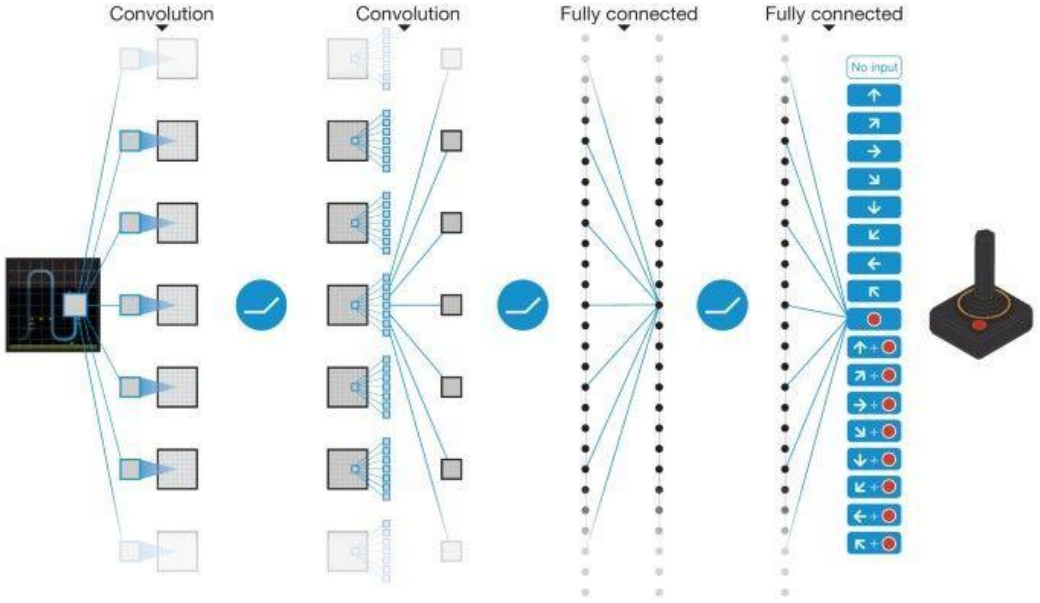


The agent-environment interaction in RL

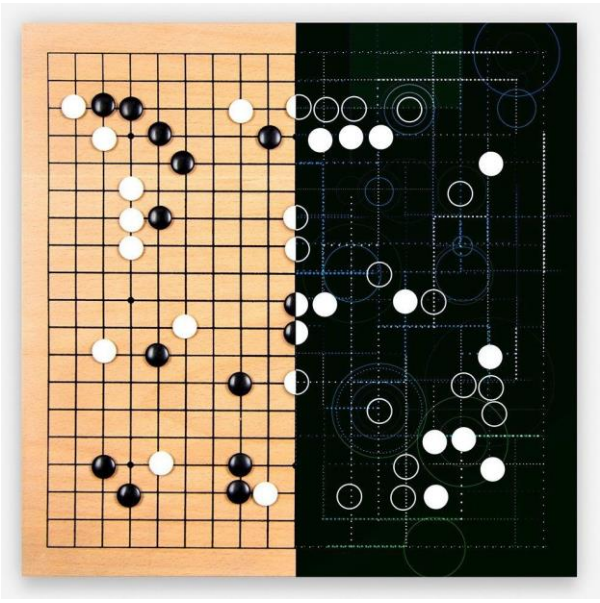
RL Agents: From Backgammon to AlphaGo



Backgammon (Q learning)

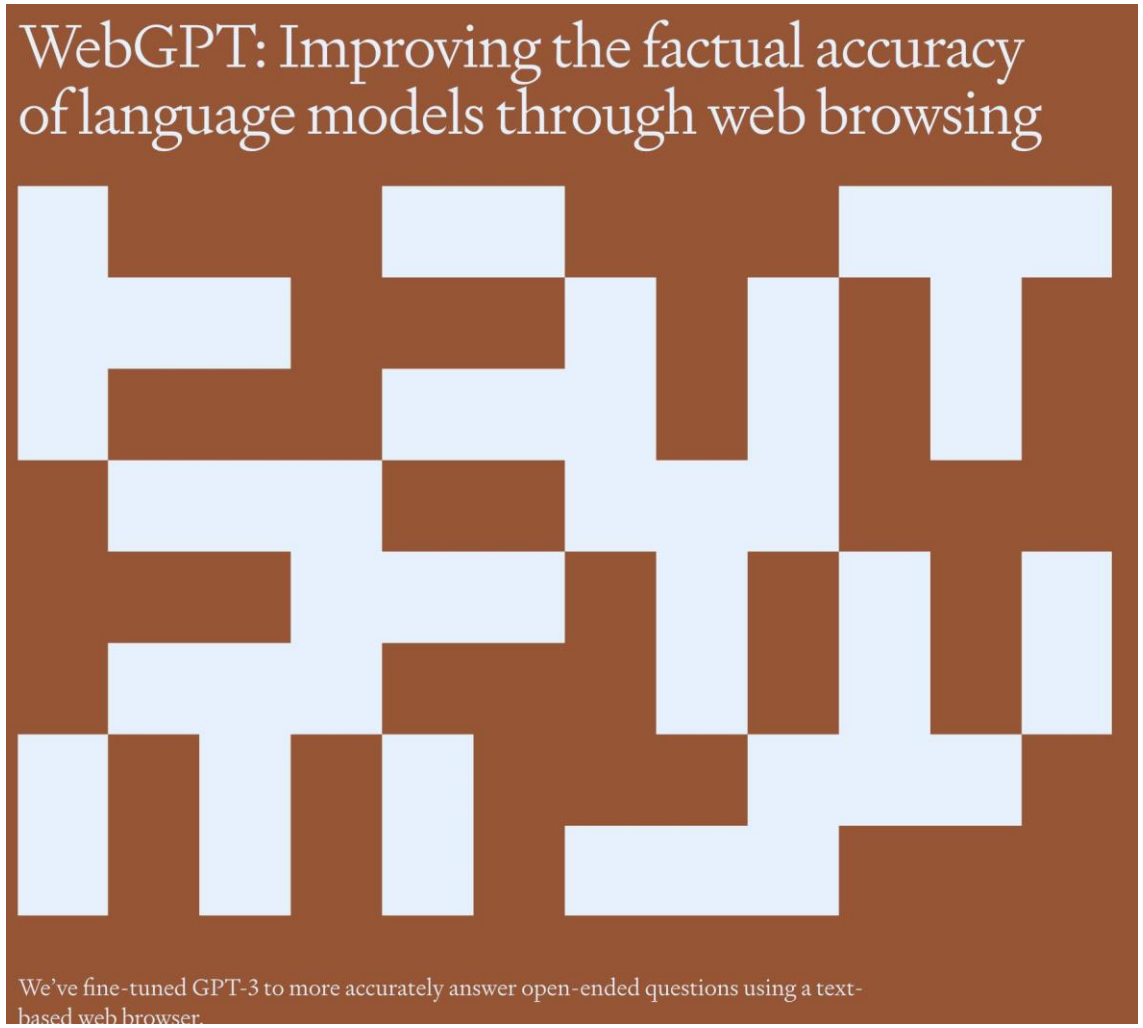


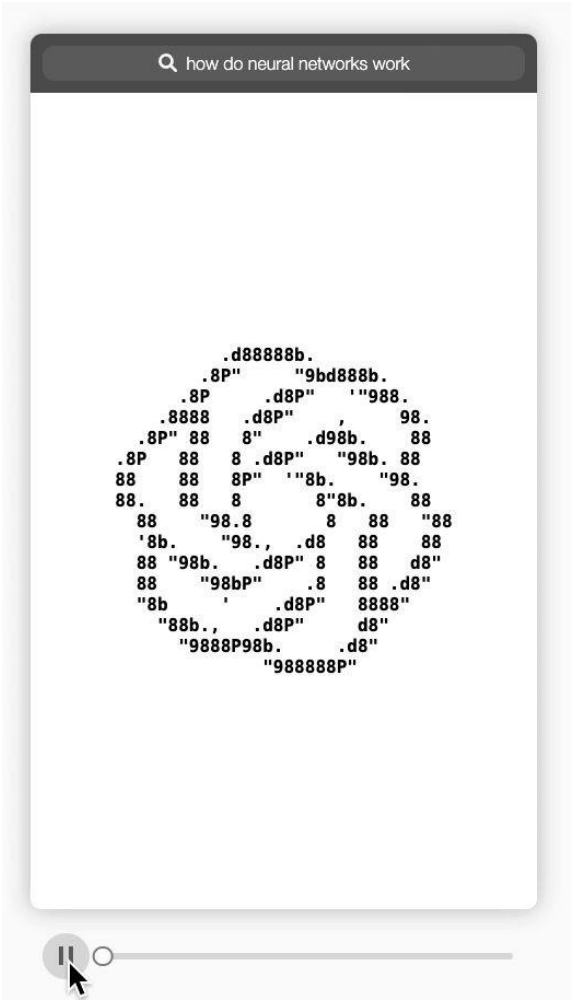
Atari Game (Deep Q Network)



Go Game (AlphaGo)

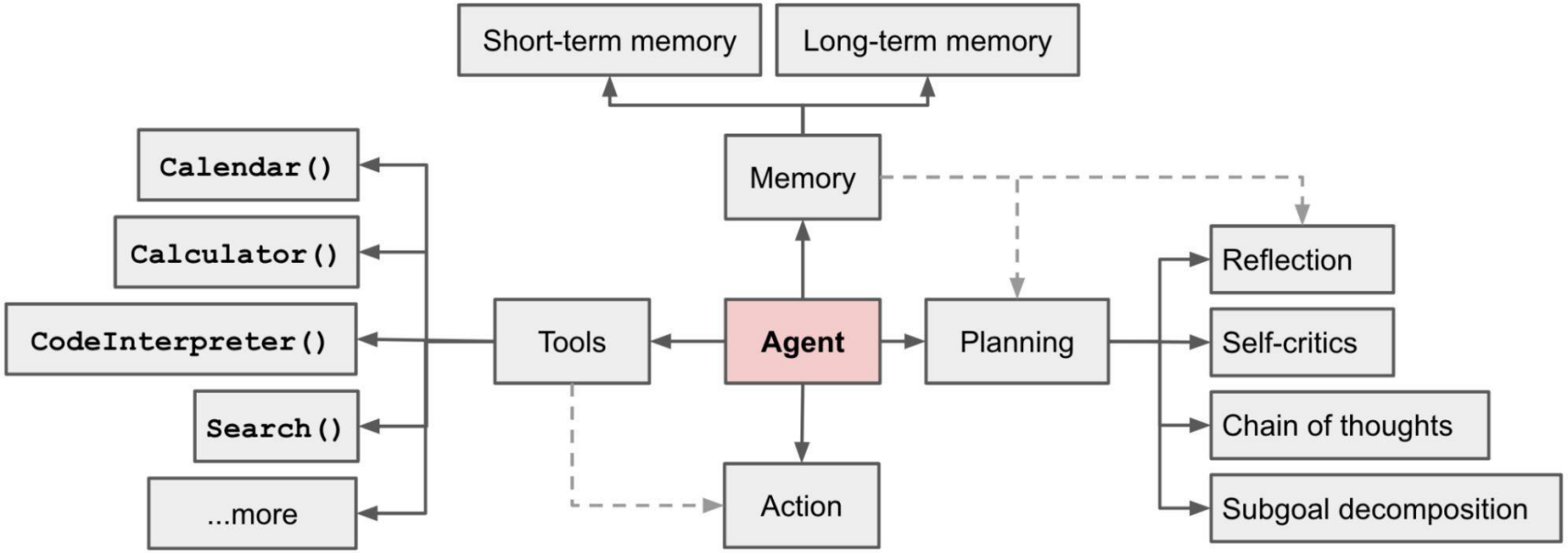
Language Models as Agents: WebGPT





Nakano, Reiichiro, et al. "Webgpt: Browser-assisted question-answering with human feedback." arXiv preprint arXiv:2112.09332 (2021).

Language Models as Agents: The Lilian Weng Picture



Lilian Weng: https://lilianweng.github.io/posts/2023-06-23-agent/

What is missing?

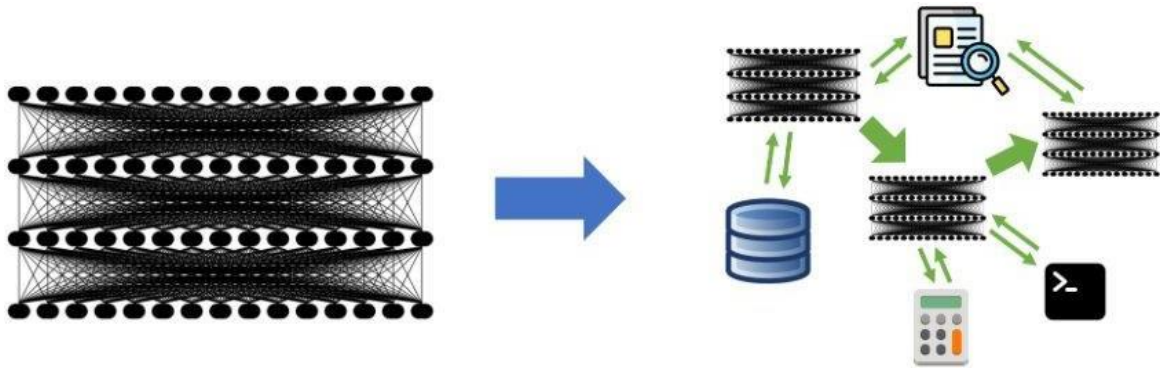
- Communication (*AI society*)
- Learning (*evolution*)
- Embodiment
- ….

Minsky had already told us many of them!

From Generative AI to Agentic AI

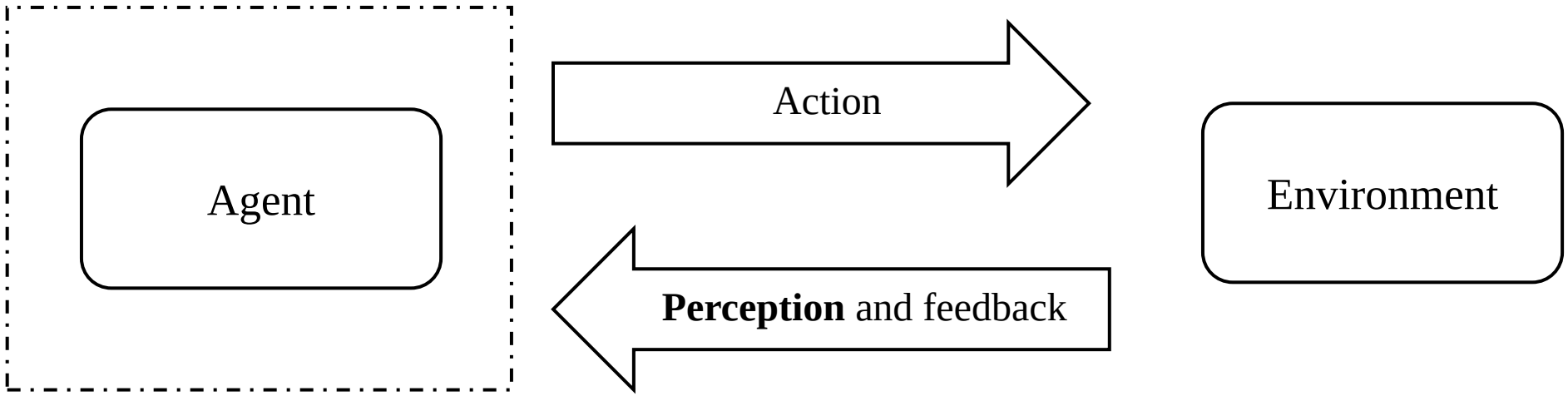
What are future AI applications like?

- Generative
  - Generate content like text & image
- Agentic
  - Execute complex tasks on behalf of human



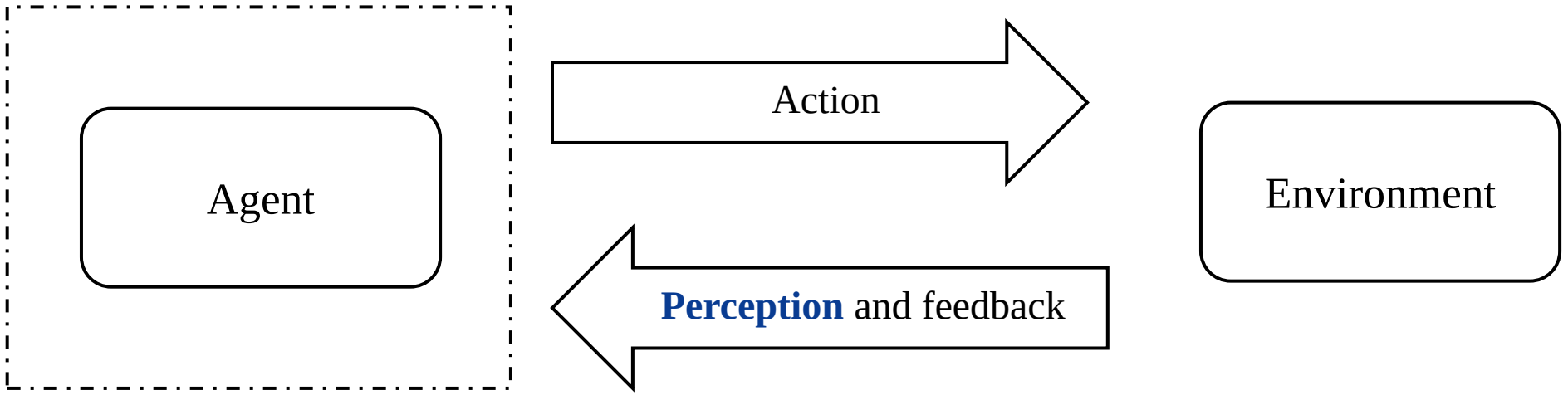
Zaharia et al. 2024. The Shift from Models to Compound AI Systems

A High-Level Picture



LLM for the cognition
e.g. planning, decision making

A High-Level Picture: Perception

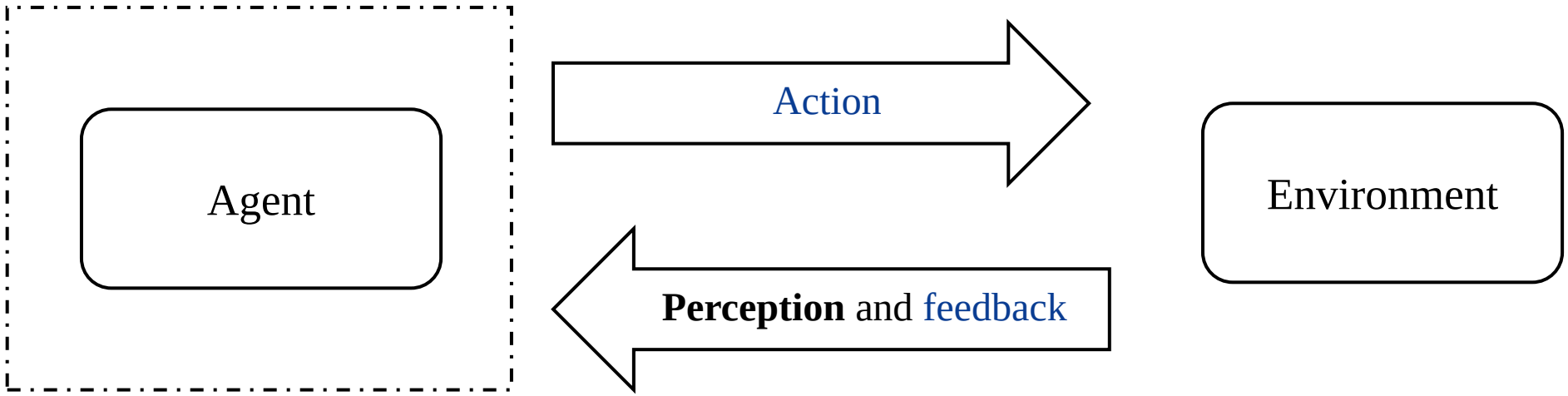


LLM for the cognition
e.g. planning, decision making



Perception

A High-Level Picture: Action and Feedback

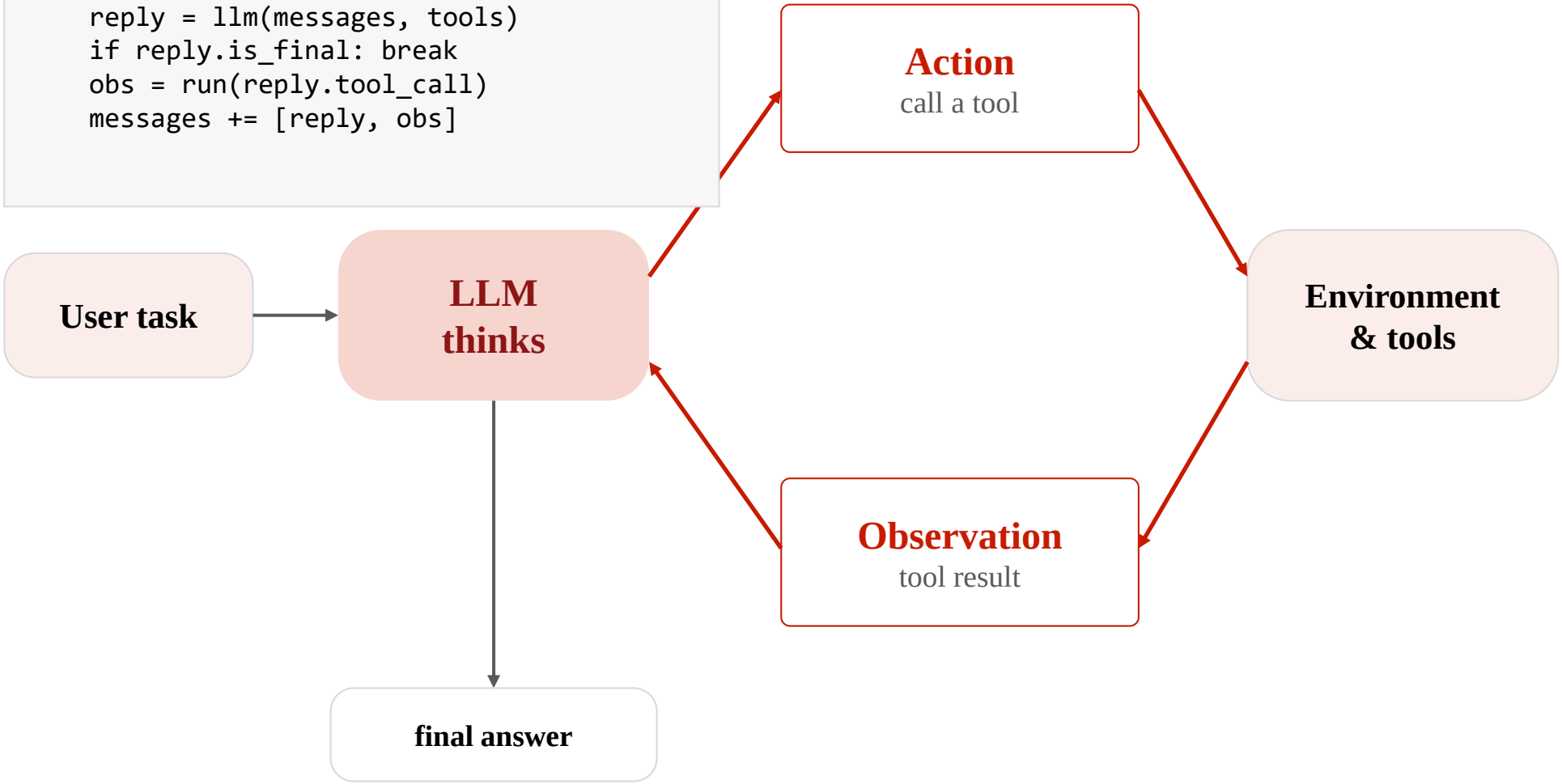


LLM for the cognition
e.g. planning, decision making

Action and feedback helps evolution of LLM agents

The Agent Loop: Think → Act → Observe

```
while True:
    reply = llm(messages, tools)
    if reply.is_final: break
    obs = run(reply.tool_call)
    messages += [reply, obs]
```



These five lines are the core of Claude Code, Codex and Manus. Everything else is the harness.

Yao et al. ReAct (ICLR 2023) · Weng. LLM Powered Autonomous Agents (2023).

3 The Four Elements of an Agent

Planning decides what to do, tools extend what the model can do, memory decides what it remembers, action changes the world.

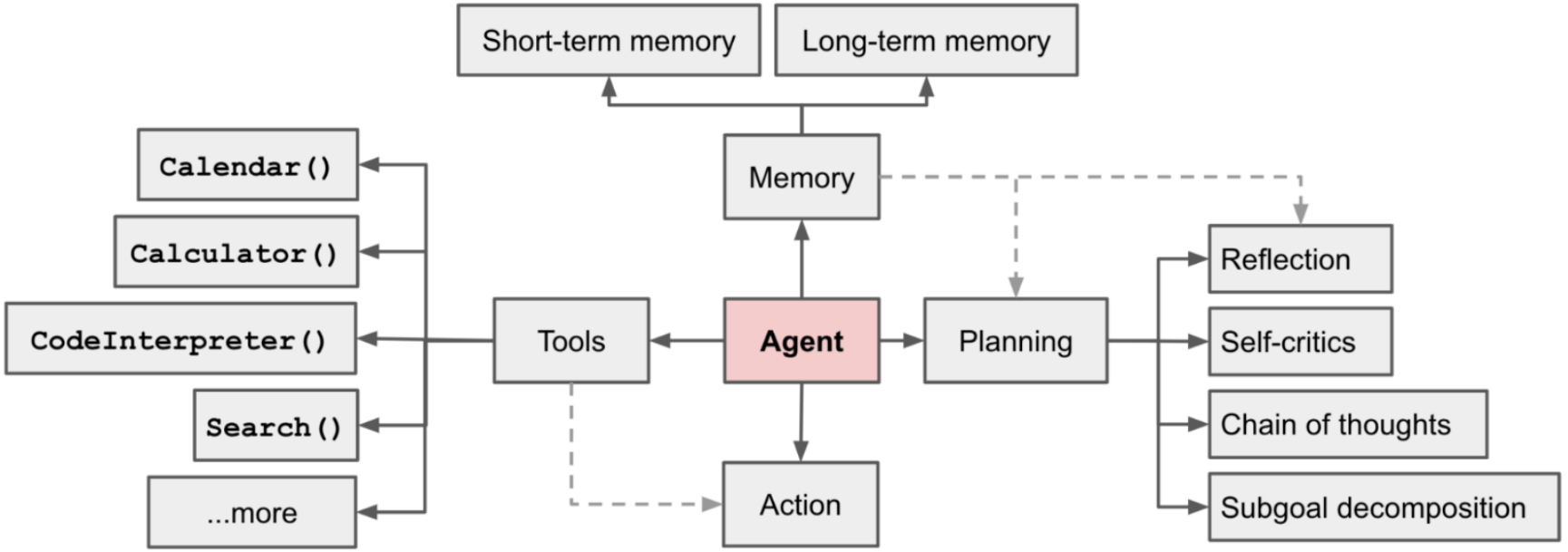
In this part

3.1 Planning
3.2 Tools
3.3 Memory
3.4 Action

The Four Elements of an LLM Agent

https://gptpluginz.com/llm-agents/

AI entities crafted around the **fundamental component** of large language models. Their prowess lies in understanding and creating human-like language, enabling a multitude of applications.



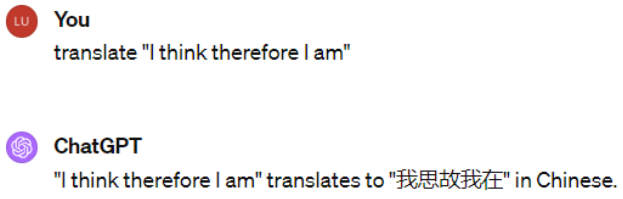
https://lilianweng.github.io/posts/2023-06-23-agent/

Planning: Solving a Complicated Task Step by Step

How to a solve a complicated task **sequentially?**

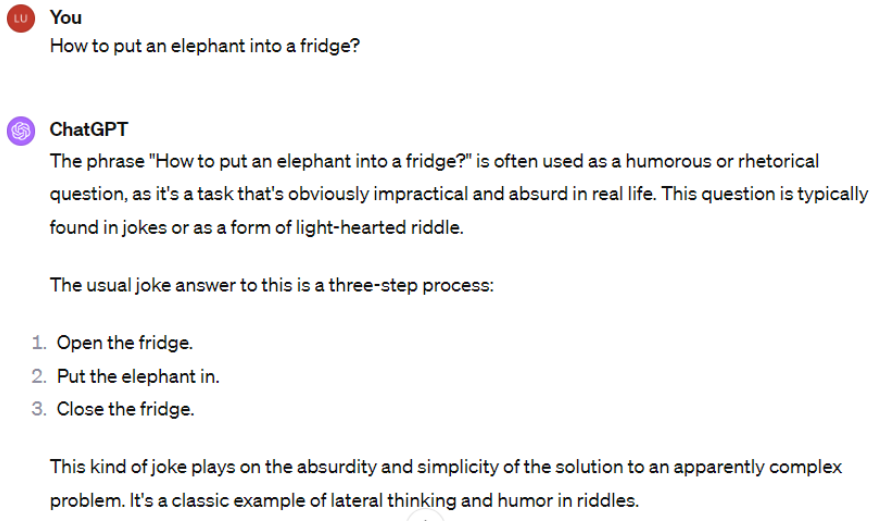
One-step task: translate an paragraph

- simple
- usually without interaction



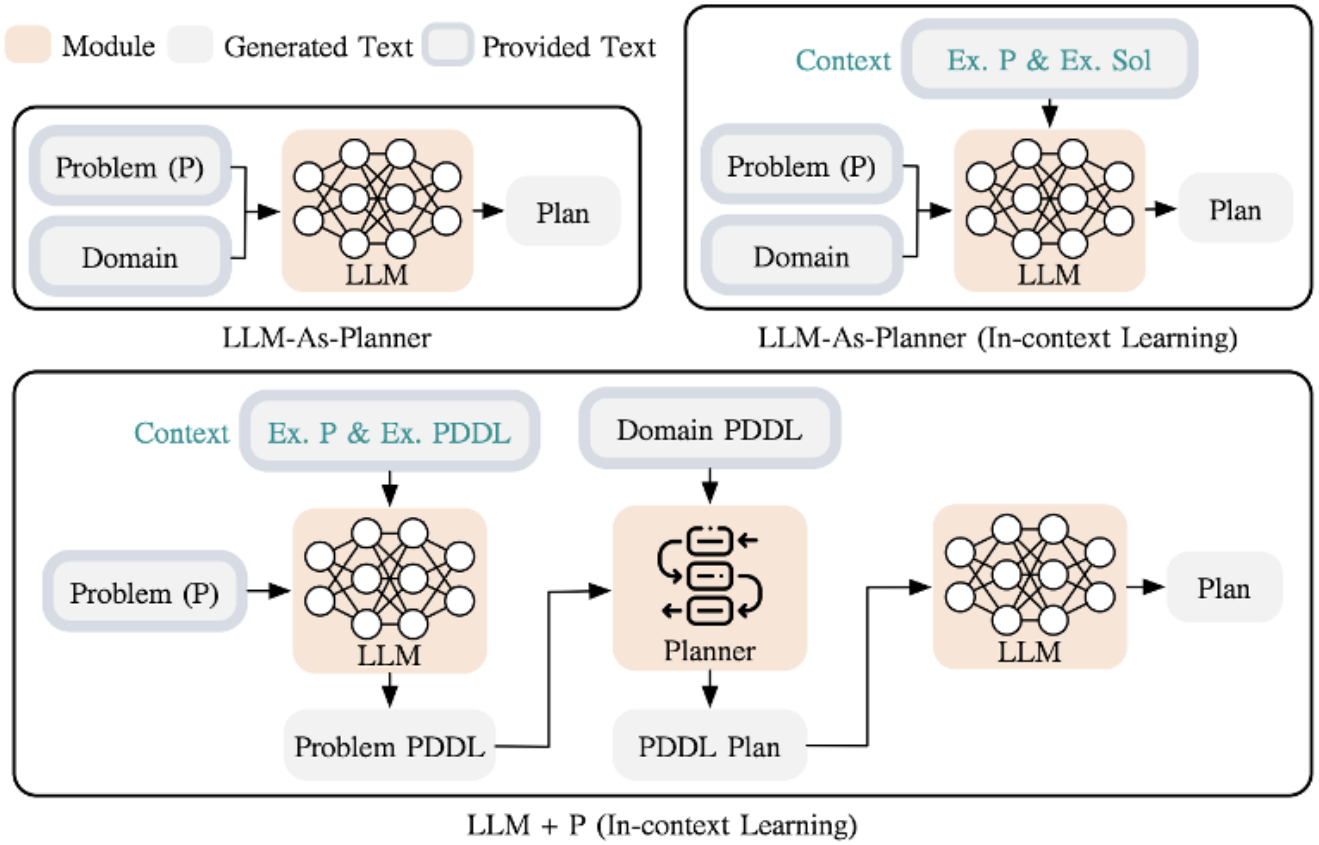
Multi-step task: How to put an elephant into a fridge?

- complicated
- it involves multple steps
- it could use **external tools or do real-world actions (e.g., operate the fridge)**



Task Decomposition with a Classical Planner: LLM+P

LLM+P involves relying on an external classical planner to do long-horizon planning. This approach utilizes the **Planning Domain Definition Language** (PDDL) as an intermediate interface to describe the planning problem.



LLM+P: Empowering Large Language Models with Optimal Planning Proficiency

LLM+P: How the PDDL Pipeline Works

In the PDDL process, LLM

1) **translates** the problem into “Problem PDDL”;
2) requests a **classical planner** to generate a PDDL **plan** based on an existing “Domain PDDL”;
3) **translates** the PDDL plan **back** into natural language.

Essentially, the planning step is outsourced to an external tool, assuming the availability of domain-specific PDDL and a suitable planner.

An Example PDDL Problem File Written by GPT-4 with Context

Context (shaded):

An example planning problem is: You have 5 blocks. b2 is on top of b5. b5 is on top of b1. b1 is on top of b4. b3 is on top of b2. b4 is on the table. b3 is clear. Your arm is empty. Your goal is to move the blocks. b4 should be on top of b3. The problem PDDL is:

```
(:objects b1 b2 b3 b4 b5)
(:init (arm-empty) (on b1 b4) (on
b2 b5) (on b3 b2)
       (on-table b4) (on b5 b1)
(clear b3))
(:goal (and (on b4 b3))))
```

Prompt: Description of Problem (P1) + Context + Provide me with the problem PDDL file that describes the planning problem directly without further explanations.

GPT-4 (the generated problem PDDL):

```
(:objects b1 b2 b3 b4 b5 )
(:init (arm-empty) (on-table b1)
(on b2 b1) (on b3 b4)
       (on b4 b2) (on b5 b3) (clear
b5)))
(:goal (and (on b1 b2) (on b3 b5)
(on b4 b1)))
```

Planner:

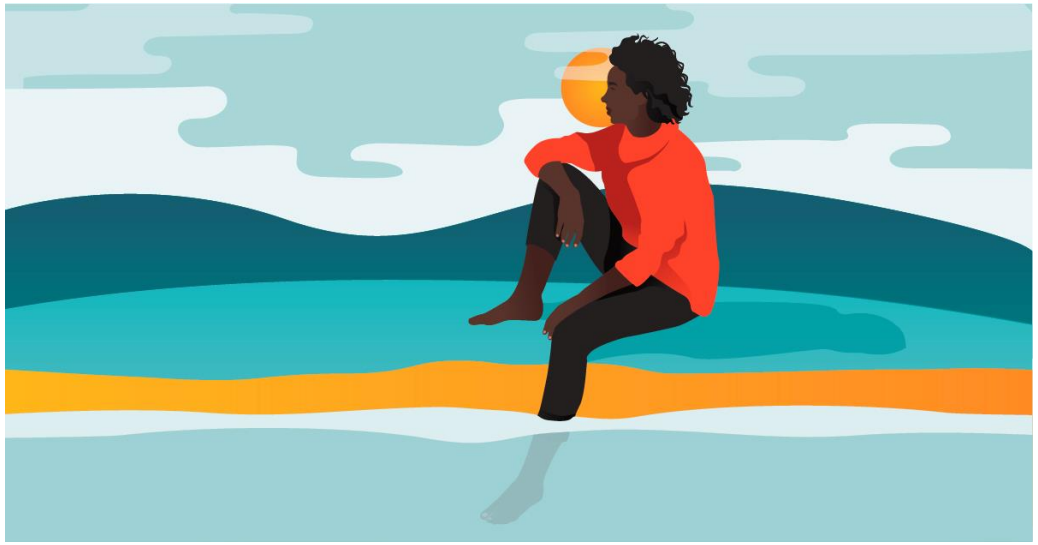
```
(unstack b5 b3) (putdown b5)
(unstack b3 b4) (stack b3 b5)
(unstack b4 b2) (putdown b4)
(unstack b2 b1) (putdown b2)
(pickup b1) (stack b1 b2) (pickup
b4) (stack b4 b1)
```

LLM+P: Empowering Large Language Models with Optimal Planning Proficiency

Self-Reflection

Self-reflection is a vital aspect that allows autonomous agents to **improve iteratively by refining past action decisions and correcting previous mistakes**. It plays a crucial role in real-world tasks where trial and error are inevitable.





ReAct: Reason + Act

ReACT integrates reasoning and acting within LLM by extending the action space to be a combination of task-specific discrete actions and the language space. The former enables LLM to interact with the environment (e.g. use Wikipedia search API), while the latter prompting LLM to generate reasoning traces in natural language.

The ReAct prompt template incorporates explicit steps for LLM to think, roughly formatted as:

```
Thought: ...
Action: ...
Observation: ...
... (Repeated many times)
```

ReAct: Synergizing Reasoning and Acting in Language Models

ReAct Beats Acting Alone

HotpotQA

(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]

AlfWorld Env

(2b) ReAct (Reason + Act)

Act 1: **Think**[First I need to find a pepper shaker...more likely to appear in cabinets (1-6), countertops (1-3), ...]
Act 2: Go to cabinet 1
Obs 2: On the cabinet 1, you see a vase 2.

(Here the agent go to cabinet 1, then cabinet 2, 3, then countertop 1 and 2)

Act 6: Go to countertop 3
Obs 6: On the countertop 3, you see a apple 1, a bread 1, ... a pepper shaker 1, and a vase 3.

Act 7: Take pepper shaker 1 from countertop 3
Obs 7: You pick up the pepper shaker 1 from the countertop 3.

Act 8: **Think**[Now I find a pepper shaker 1. Next, I need to put it in/on drawer 1.]
Act 9: Go to drawer 1
Obs 9: Drawer 1 is closed.

Act 10: Open drawer 1
Obs 10: You open Drawer 1 …

Act 11: Put pepper shaker 1 in/on drawer 1
Obs 11: You put pepper shaker 1 in/on the drawer 1.

In both experiments on knowledge-intensive tasks and decision-making tasks, *ReAct* works better than the *Act*-only baseline where *Thought: …* step is removed.

ReAct: Synergizing Reasoning and Acting in Language Models

3.2 Tools

Model weights are frozen and limited; tools let an agent compute, search and act.

Why Do Humans Use Tools?

As humans, we have limited time and memory, feel tired, and have emotions.

- Human + tool use
  - Enhanced scalability
  - Improved consistency
  - Greater interpretability
  - Higher capacity and productivity



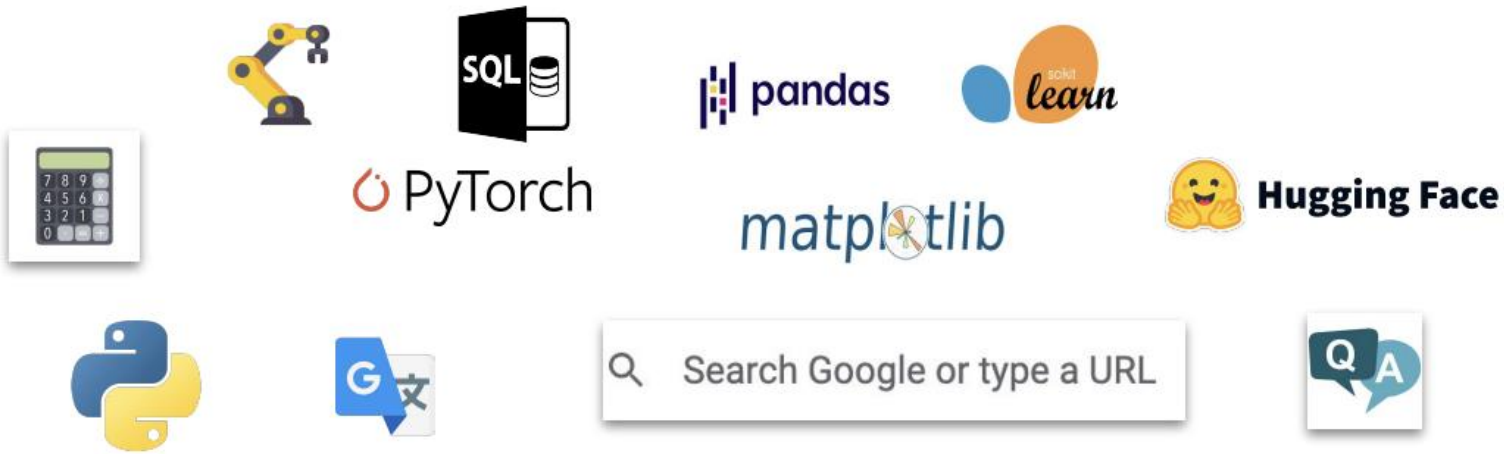




Why Do LLMs Need Tools?

- Just like human, LLMs also benefits from tools
- LLMs + tool use
  - Enhanced scalability
  - Improved consistency
  - Greater interpretability
  - Higher capacity and productivity

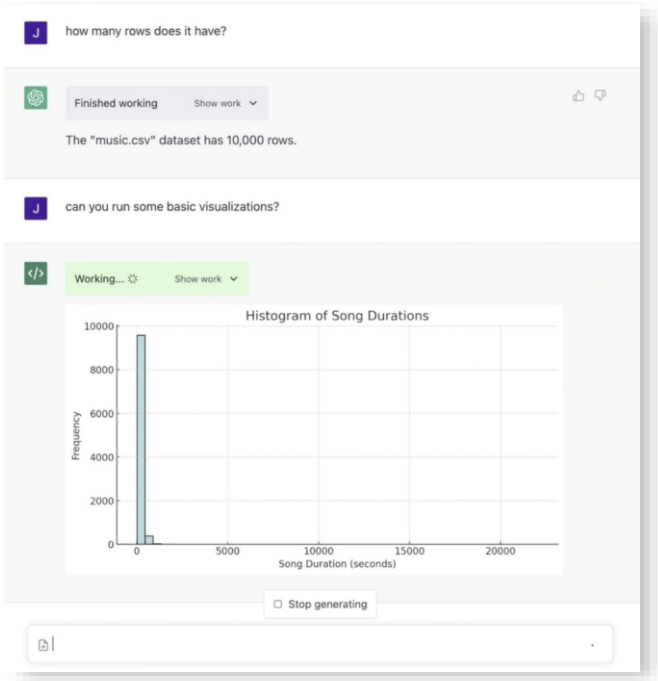




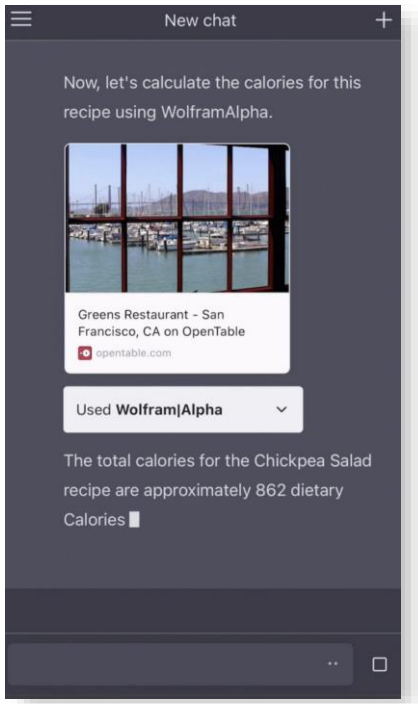
Tools as Executable Language Grounding

Ground language models into executable actions

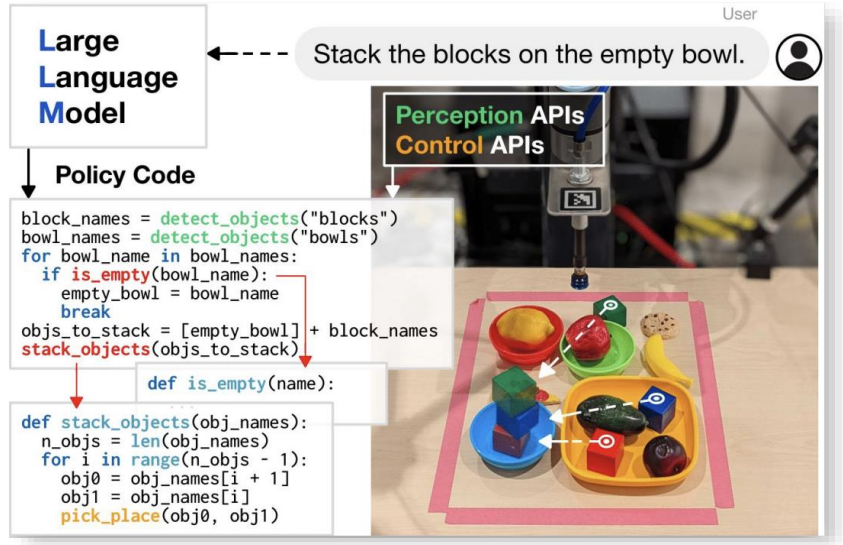
- Mapping natural language instructions into code or actions executable within various environments such as databases, web applications, and robotic physical world.
- LM (planning and reasoning) + actions



Data analysis



Web/Apps



Robotic physical world

https://openai.com/blog/chatgpt-plugins
https://code-as-policies.github.io/

Tool Calling, Step by Step

① The harness declares tools

```
{"name": "get_weather",
 "description": "Current
weather",
 "parameters": {"city":
"string"}}
```

harness

② The model asks for a call

```
{"tool": "get_weather",
 "arguments": {"city":
"Shenzhen"}}
```

model

③ The harness runs it

```
{"temp_c": 28, "sky":
"cloudy"}
```

harness

④ The model answers

```
“It is 28 ° C and cloudy in
Shenzhen.”
```

model

The model never runs anything. It only writes a structured request; the harness decides whether and how to run it.

That is where permissions, sandboxes, timeouts and logging live.

OpenAI function calling (2023) · Anthropic tool use · Schick et al. Toolformer (2023).

Tools for Web and Apps: ChatGPT Plugins

Function calling example

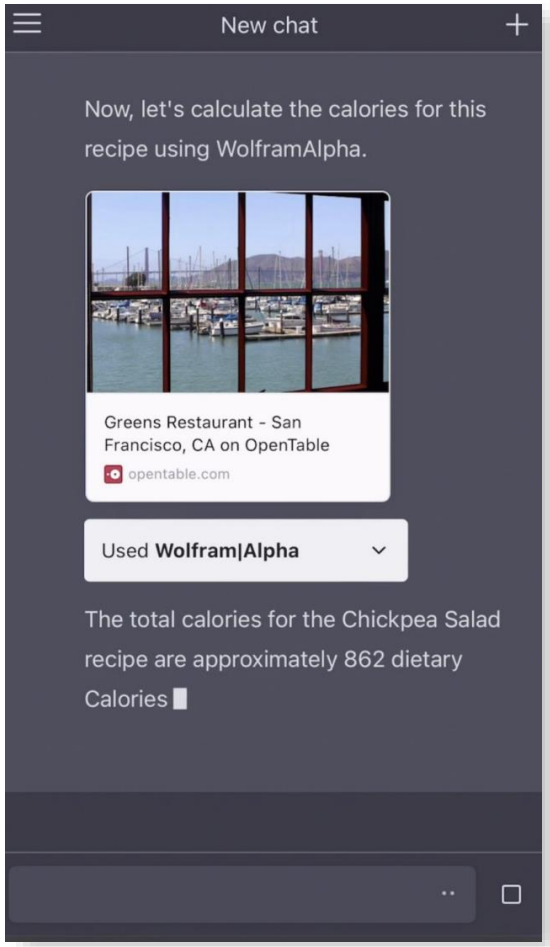
What's the weather like in Boston right now?

Step 1 · OpenAI API

Call the model with functions and the user's input

Request Response

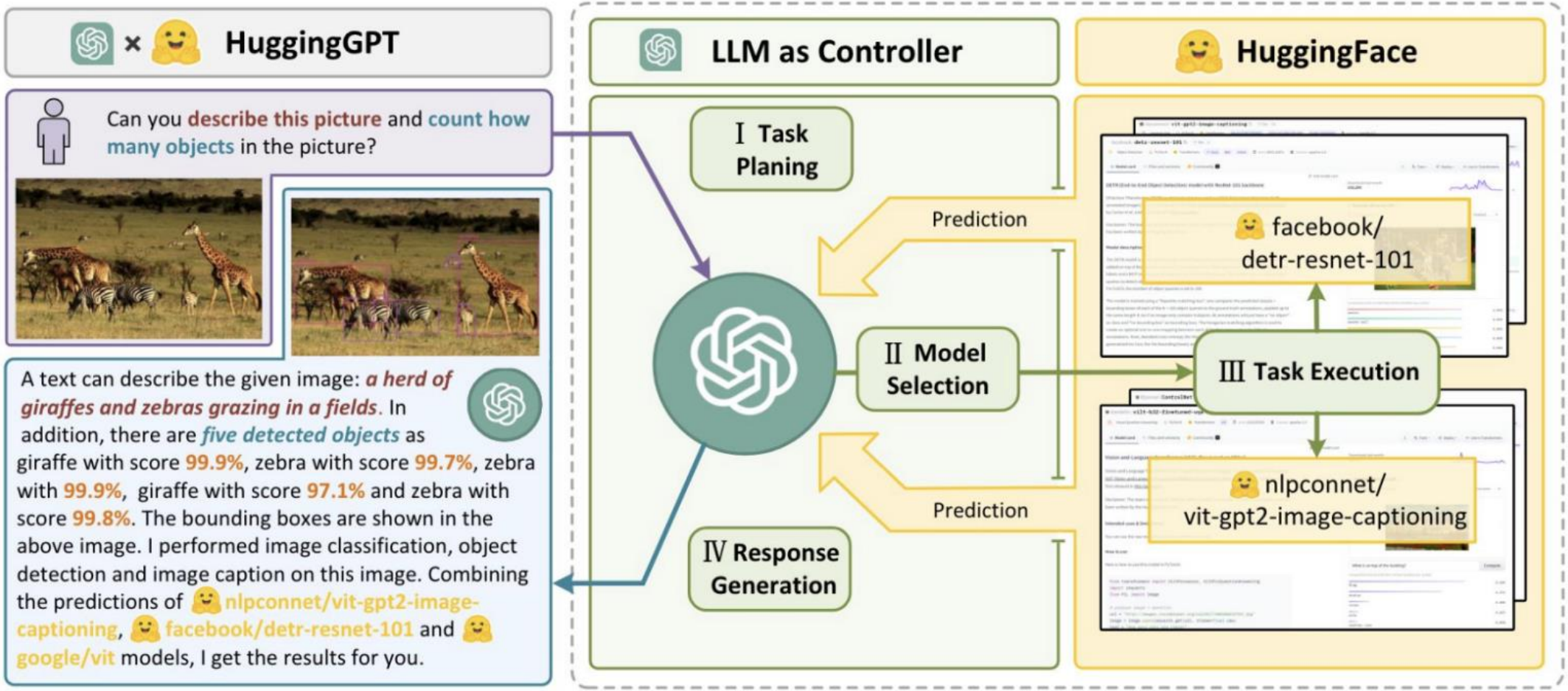
```
curl https://api.openai.com/v1/chat/completions -u :$OPENAI_API_KEY -H 'Content-Type: application/json' -d '{
  "model": "gpt-3.5-turbo-0613",
  "messages": [
    {"role": "user", "content": "What is the weather like in Boston?"}
  ],
  "functions": [
    {
      "name": "get_current_weather",
      "description": "Get the current weather in a given location",
      "parameters": {
        "type": "object",
        "properties": {
          "location": {
            "type": "string",
            "description": "The city and state, e.g. San Francisco, CA"
          },
          "unit": {
            "type": "string",
            "enum": ["celsius", "fahrenheit"]
          }
        },
        "required": ["location"]
      }
    }
  ]
}'
```



https://openai.com/blog/chatgpt-plugins

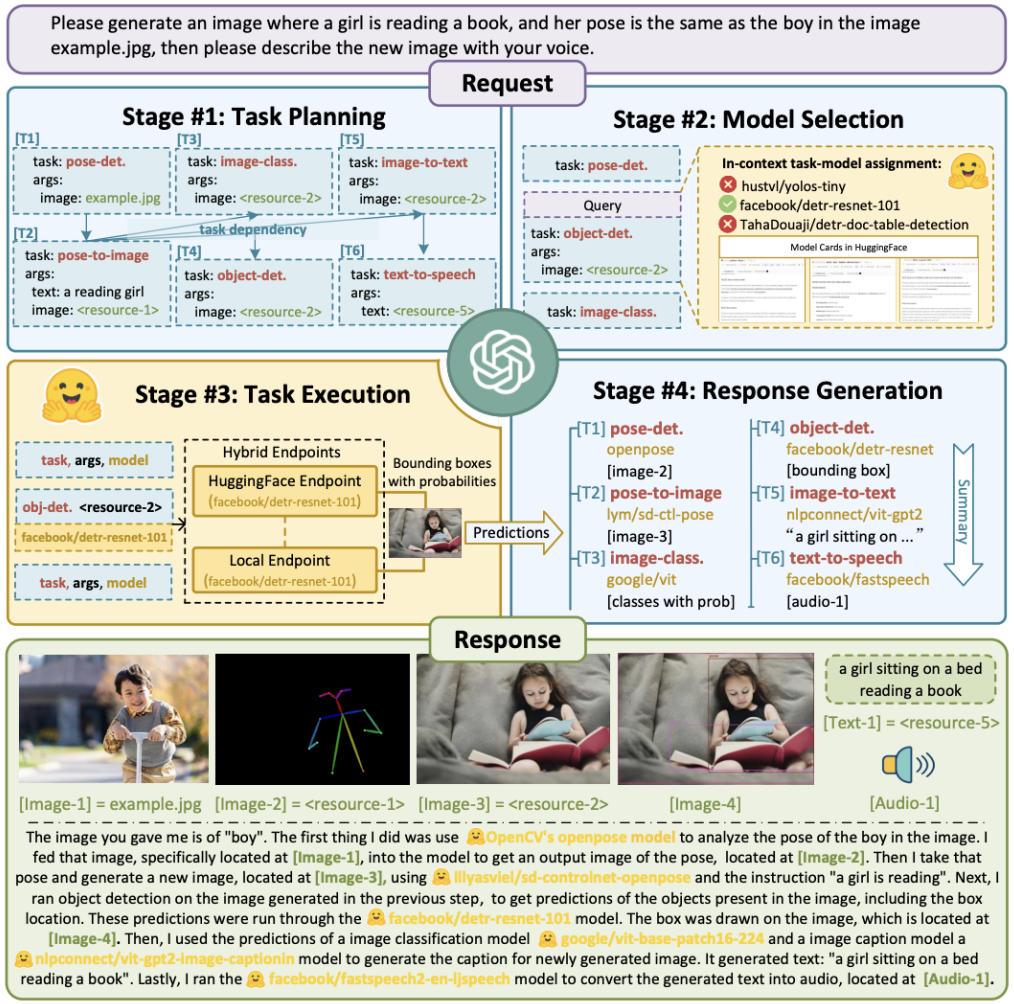
Mind2Web: Towards a Generalist Agent for the Web

Tools as Expert Models: HuggingGPT



HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face

HuggingGPT: Four Stages

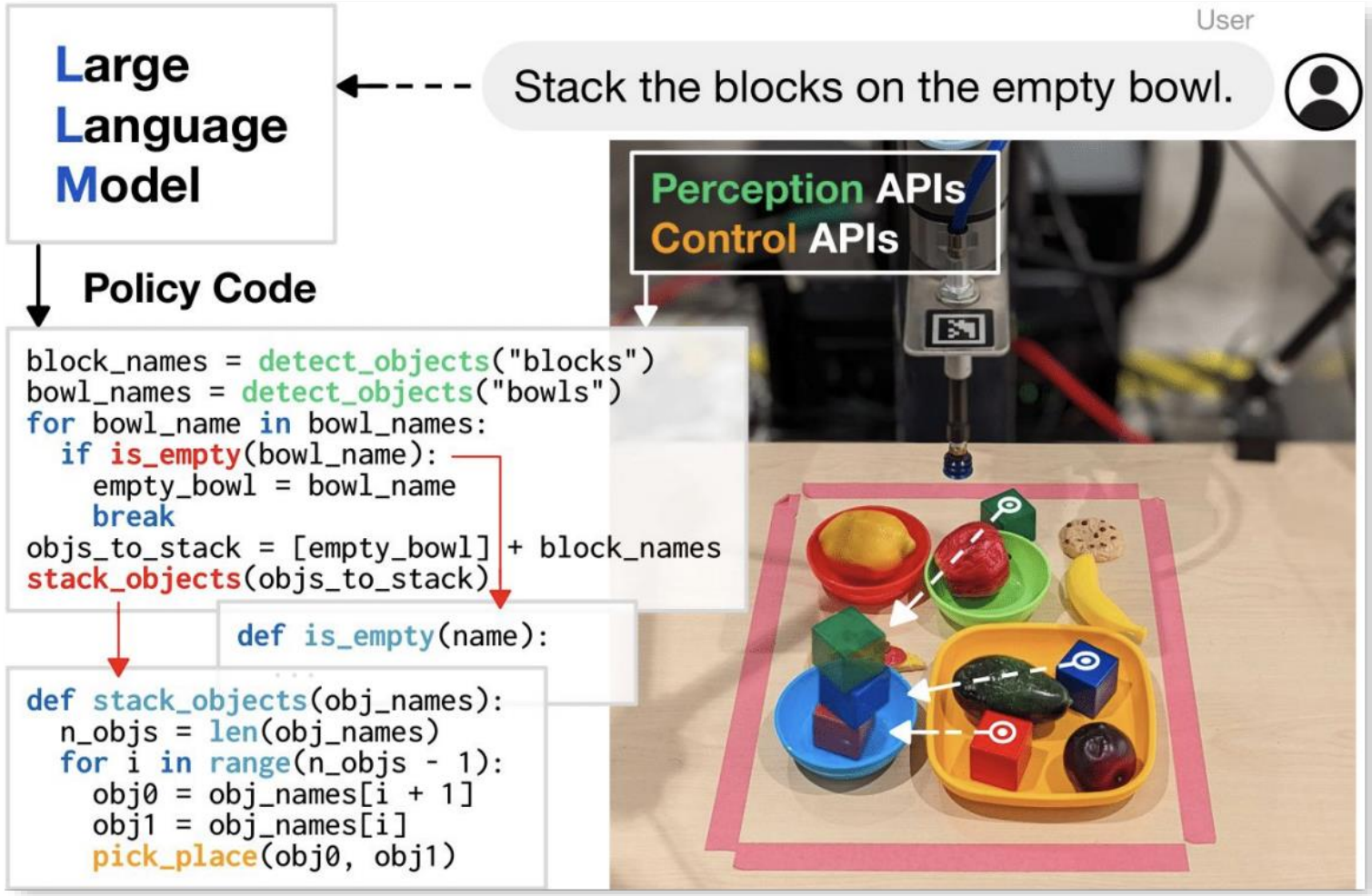


The system comprises of 4 stages:

- **Task Planning:** LLM analyze the user's requests, breaking them down into solvable tasks through prompts.
- **Model Selection:** LLM is presented with a list of models to choose from and distributes the tasks to expert models. LLM.
- **Task Execution:** Expert models execute on the specific tasks and log results.
- **Response Generation:** LLM receives the execution results and provides summarized results to users.

HuggingGPT: Solving AI Tasks with ChatGPT and its Friends in Hugging Face

Code as Policies: Tools for Robots



Code as Policies: Language Model Programs for Embodied Control
Do As I Can, Not As I Say: Grounding Language in Robotic Affordances
ProgPrompt: Generating Situated Robot Task Plans using Large Language Models
Mind's Eye: Grounded Language Model Reasoning through Simulation

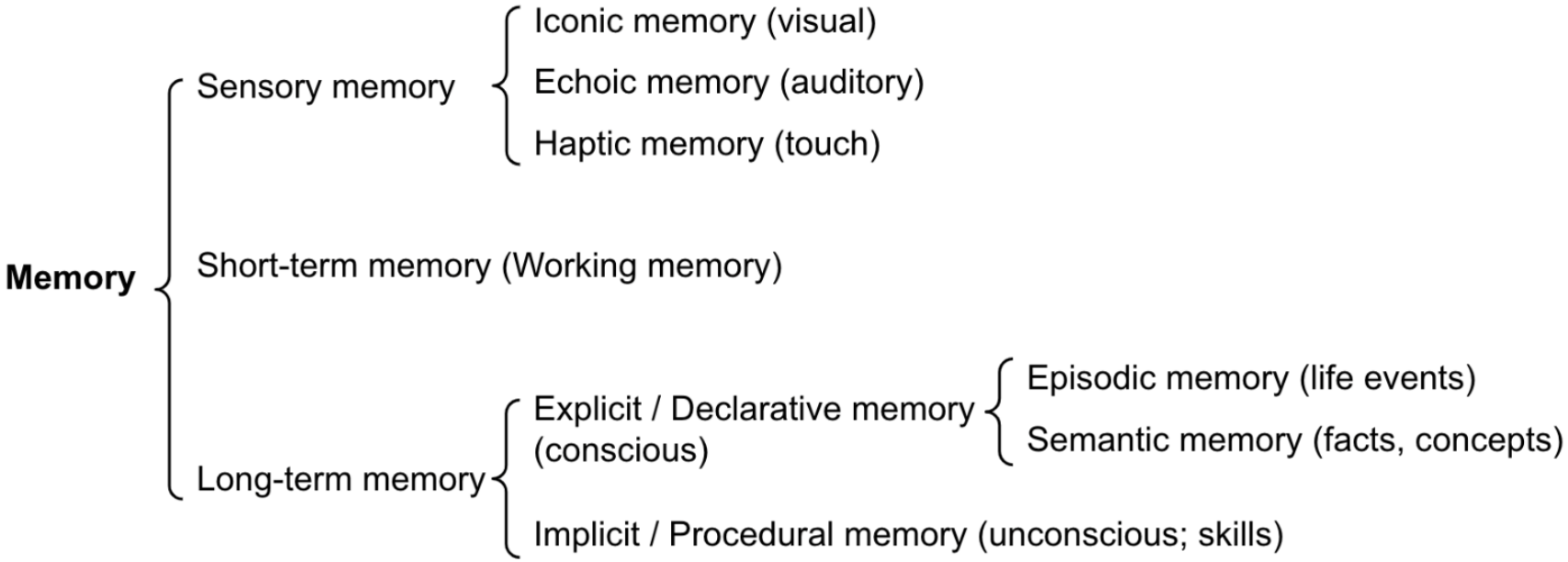
3.3 Memory

The context window is short-term memory. Everything else must be stored outside and brought back at the right time.

Memory in the Human Brain

1. **Sensory Memory:** This is the earliest stage of memory, providing the ability to retain impressions of sensory information (visual, auditory, etc) after the original stimuli have ended. Sensory memory typically only lasts for up to a few seconds. Subcategories include iconic memory (visual), echoic memory (auditory), and haptic memory (touch).
2. **Short-Term Memory (STM) or Working Memory:** It stores information that we are currently aware of and needed to carry out complex cognitive tasks such as learning and reasoning. Short-term memory is believed to have the capacity of about 7 items (Miller 1956) and lasts for 20-30 seconds.
3. **Long-Term Memory (LTM):** Long-term memory can store information for a remarkably long time, ranging from a few days to decades, with an essentially unlimited storage capacity.
   There are two subtypes of LTM:
   a. **Explicit / declarative memory:** This is memory of facts and events, and refers to those memories that can be consciously recalled, including episodic memory (events and experiences) and semantic memory (facts and concepts).
   b. **Implicit / procedural memory:** This type of memory is unconscious and involves skills and routines that are performed automatically, like riding a bike or typing on a keyboard.

Types of Memory in LLM Agents



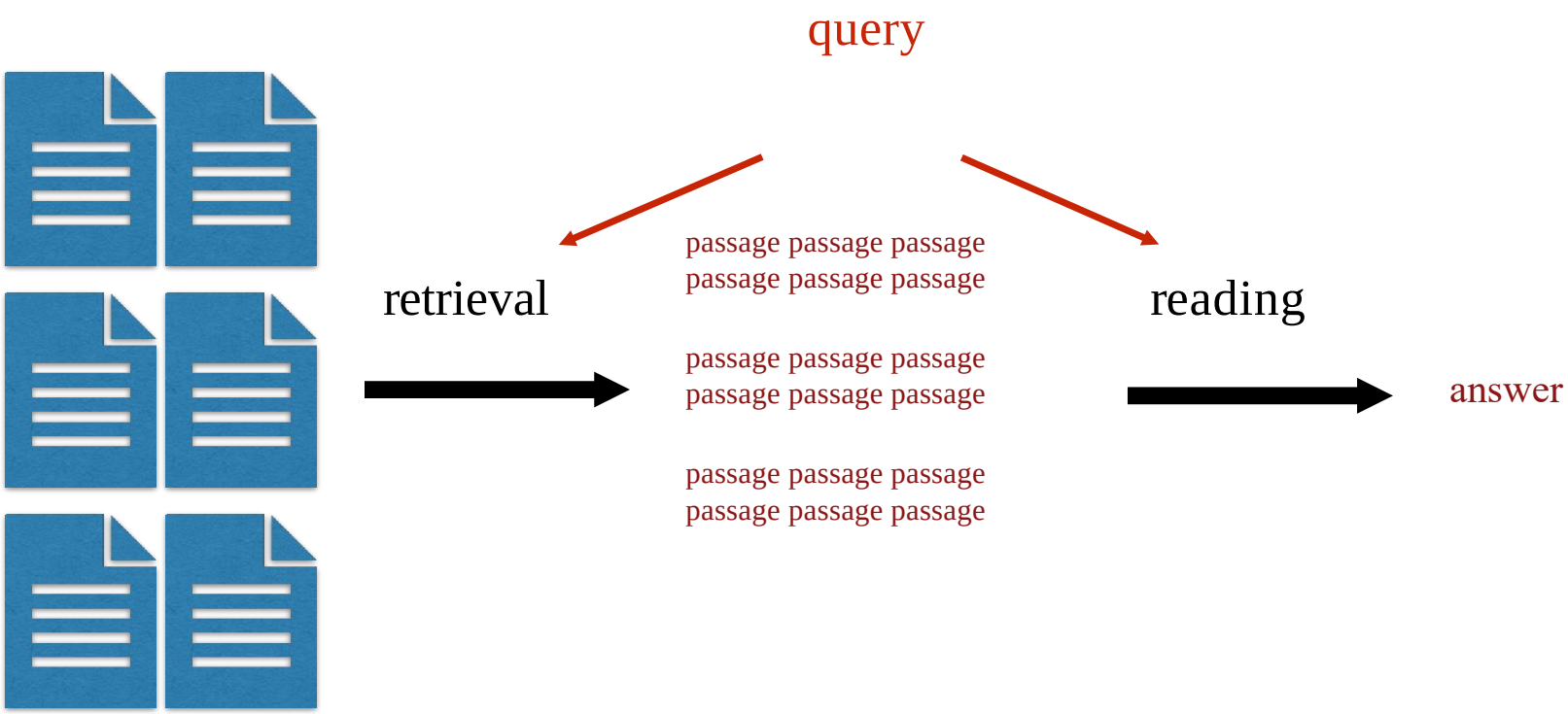
1. **Sensory Memory:** learning embedding representations for raw inputs, including text, image or other modalities;
2. **Short-Term Memory (STM):** in-context learning. It is short and finite, as it is restricted by the finite context window length of Transformer.
3. **Long-Term Memory (LTM):** the external vector store that the agent can attend to at query time, accessible via fast retrieval.

Why External Memory? Problems of Parametric Knowledge

- *Accuracy issues:*
  - **Knowledge cutoffs:** parameters are usually only updated to a particular time
  - **Private data:** data stored in private text or data repositories not suitable for training
  - **Learning failures:** even for data that the model was trained on, it might not be sufficient to get the right answer
- *Verifiability issues:* It is hard to tell if the answer is correct

Retrieval-Augmented Generation (RAG)

- **Retrieve** relevant passages efficiently
- **Read** the passages to answer the query



Memory in Practice: What Goes Back into the Context



In-context

The conversation so far. Fast and exact — but lost at the end of the session and limited by the window.



Instruction files

AGENTS.md / CLAUDE.md: loaded at every start. Project rules, commands, conventions.



Notes & progress files

The agent writes PLANS.md or progress notes, and reads them in the next session.



Retrieval

Search, RAG, vector stores: fetch just-in-time what is relevant now.

For an LLM, memory = whatever the harness decides to put back into the context window.

Weng (2023) · Anthropic. Effective Context Engineering for AI Agents (2025) · OpenAI. Harness Engineering (2026).

3.4 Action

Actions change the environment — from filling a web form to moving a robot arm.

Action: From Text to the World

In the construction of the agent, the action module receives action sequences sent by the planning module and carries out actions to interact with the environment.

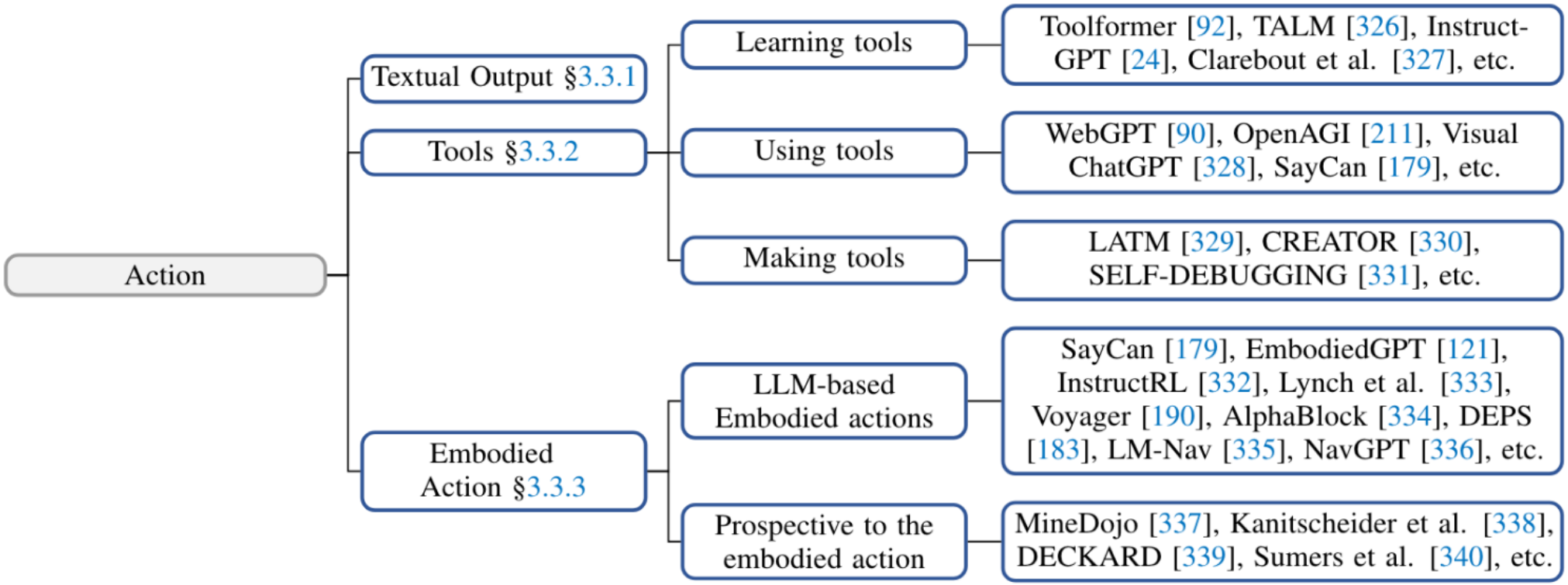
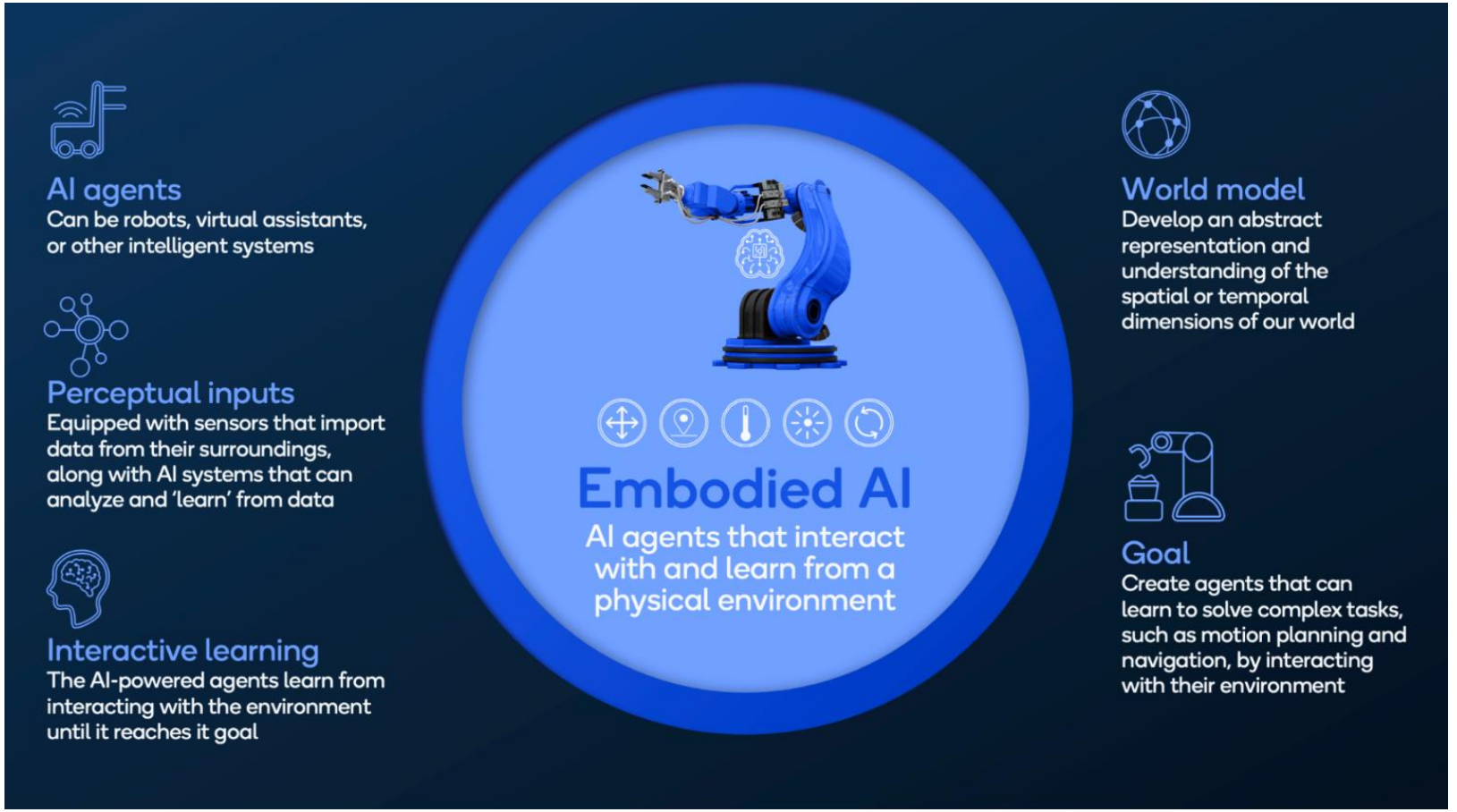


Figure 5: Typology of the action module.

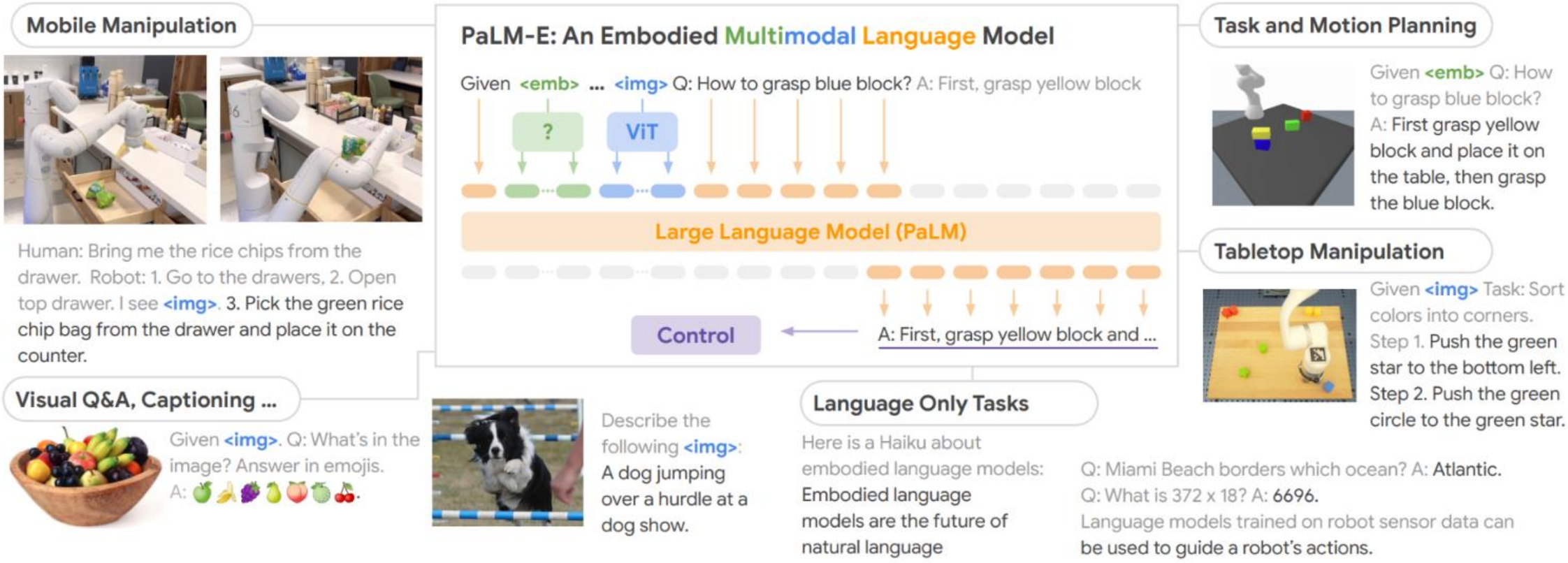
Action: Embodied AI

In the pursuit of Artificial General Intelligence (AGI), the embodied agent is considered a pivotal paradigm while it strives to integrate model intelligence with the physical world.



PaLM-E: An Embodied Multimodal Language Model

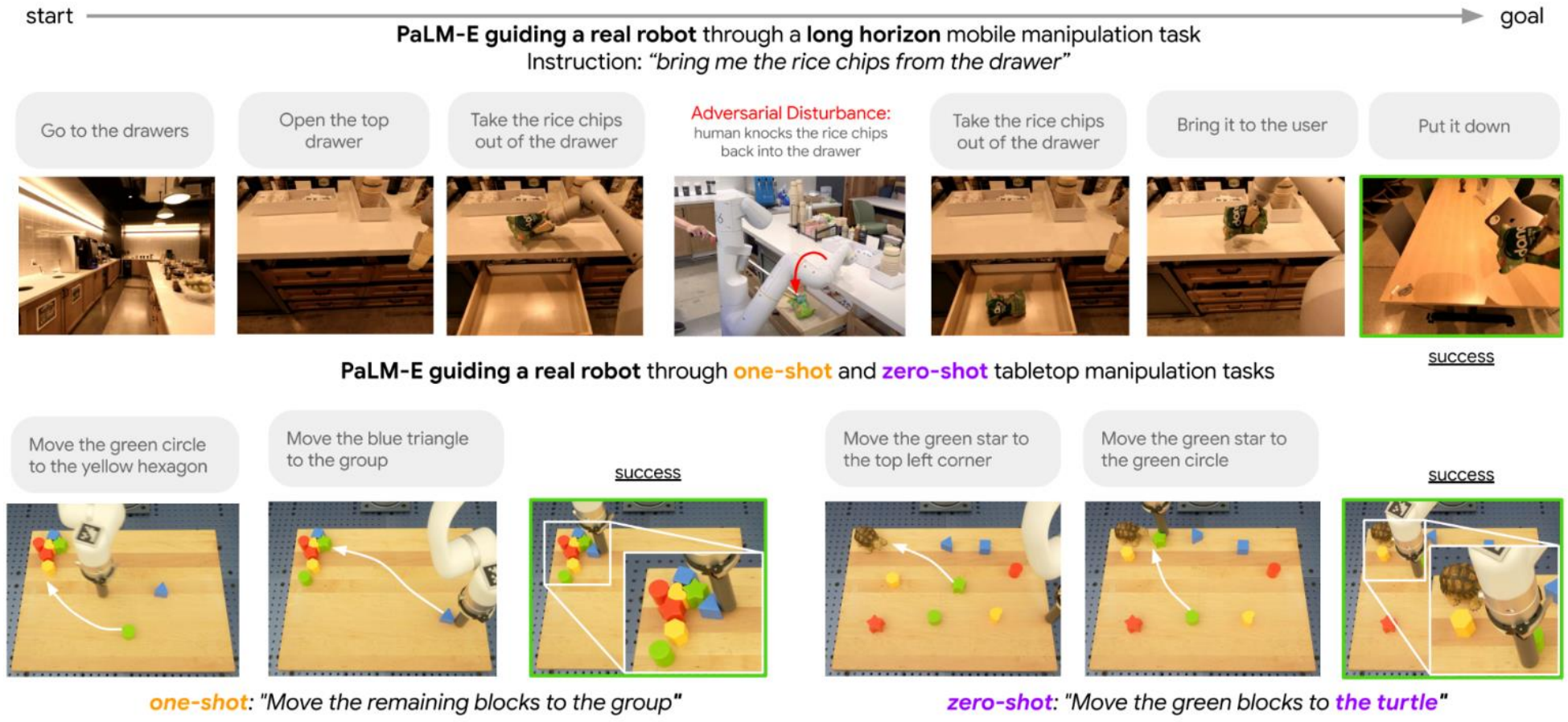
PaLM-E transfers knowledge from visual-language domains into embodied reasoning – from robot planning in environments with complex dynamics and physical constraints, to answering questions about the observable world.



PaLM-E: An Embodied Multimodal Language Model

PaLM-E Controls Real Robots

A single PaLM-E model directs the low-level policies of two real robots. Shown is a long-horizon mobile manipulation task in a kitchen, and one-shot / zero-shot generalization with a tabletop manipulation robot.



PaLM-E: An Embodied Multimodal Language Model

4 Harness Engineering: Principles

The same model can be a mediocre or an excellent agent. The difference is the harness: everything around the model.

In this part

4.1 Agent = Model + Harness

4.2 Context Engineering

4.3 Skills, Tools and MCP

4.4 Guides and Sensors

What Do We Engineer in Each AI Era?

- Feature Engineering
- Architecture Engineering
- Prompt engineering
- Context engineering
- **Environmental** Engineering
  - Building these environments and build the benchmark.

Prompt → Context → Harness Engineering

Prompt engineering 2022–2024	Context engineering 2025	Harness engineering 2026
What should I say?	**What should the model see?**	**What system does the model run in?**
Levers wording, examples, chain-of-thought	**Levers** retrieval, tool results, memory, compaction	**Levers** tools, sandbox, tests, permissions, sub-agents, evals
Unit of work one model call	**Unit of work** one context window	**Unit of work** a whole task, many sessions

Each layer contains the previous one: a good harness still needs good context, and good context still needs good prompts.

Anthropic. Effective Context Engineering for AI Agents (2025) · OpenAI. Harness Engineering (Feb 2026) · Böckeler, martinfowler.com (2026).

Agent = Model + Harness

The harness is everything in an agent except the model.

System prompt
instructions, persona

Tools · skills · MCP
what it can call

Sandbox & filesystem
where it works

Context management
compaction, offloading

Model
(frozen weights)

Memory
AGENTS.md, notes

Orchestration
sub-agents, routing

Hooks & verification
lint, tests, checks

Permissions
approvals, allow-lists

The model is the engine; the harness is the rest of the car — steering, brakes, dashboard and seat belts.

Trivedy. The Anatomy of an Agent Harness (LangChain, Mar 2026) · OpenAI. Harness Engineering (Feb 2026).

Why It Matters: Same Model, Different Harness

LangChain's coding agent on Terminal-Bench 2.0

Original harness

rank ≈ 30

Improved harness (same model)

rank ≈ 5

Only the harness changed: prompts, tools, verification, context handling.

Harness-only changes

move agents by 20+ positions on public leaderboards.

Models adapt to harnesses

a model post-trained with one harness (e.g. a specific file-edit tool) can overfit to it.

Every mistake is a harness bug

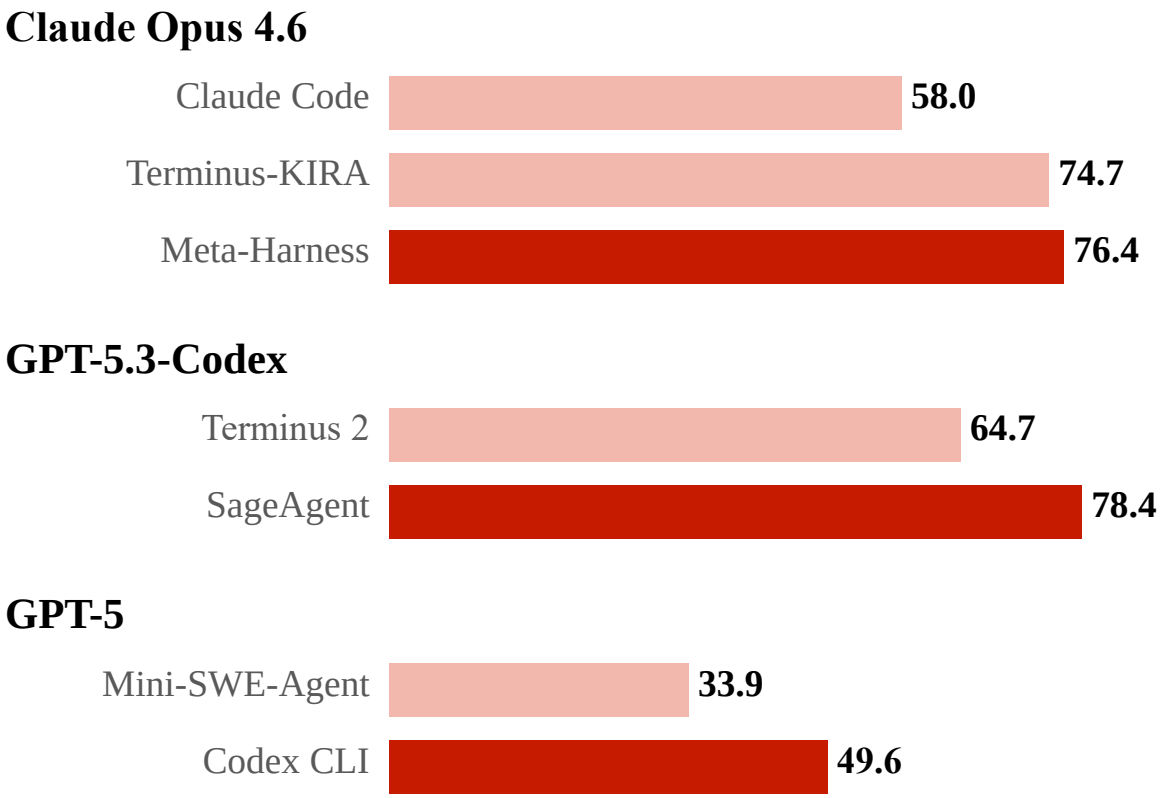
encode the fix in the harness instead of patching the output once.

Benchmarks of "model X" are really benchmarks of "model X in harness Y".

Trivedy, LangChain (Mar 2026) · deepset. Harness Engineering (2026) · OpenAI. Harness Engineering (Feb 2026).

The Harness Tax: Same Model, Different Scores

Terminal-Bench 2.0 — one model, several harnesses (accuracy %)



What this means

- Gaps of 10–20 points — larger than the gaps between many frontier models.
- A leaderboard row is a (model, harness) pair, never a model alone.
- Vendors' own CLIs are not automatically best for their own models.
- Harness rankings move with every release (TB 2.1 reversed several pairs).

Before you compare models, fix the harness. Before you blame the model, read the harness.

Terminal-Bench 2.0 leaderboard (tbench.ai, Snorkel) · Survey on agent system and harness design (arXiv:2606.20683) · Arena.ai HarnessTax (2026).

4.2 Context Engineering

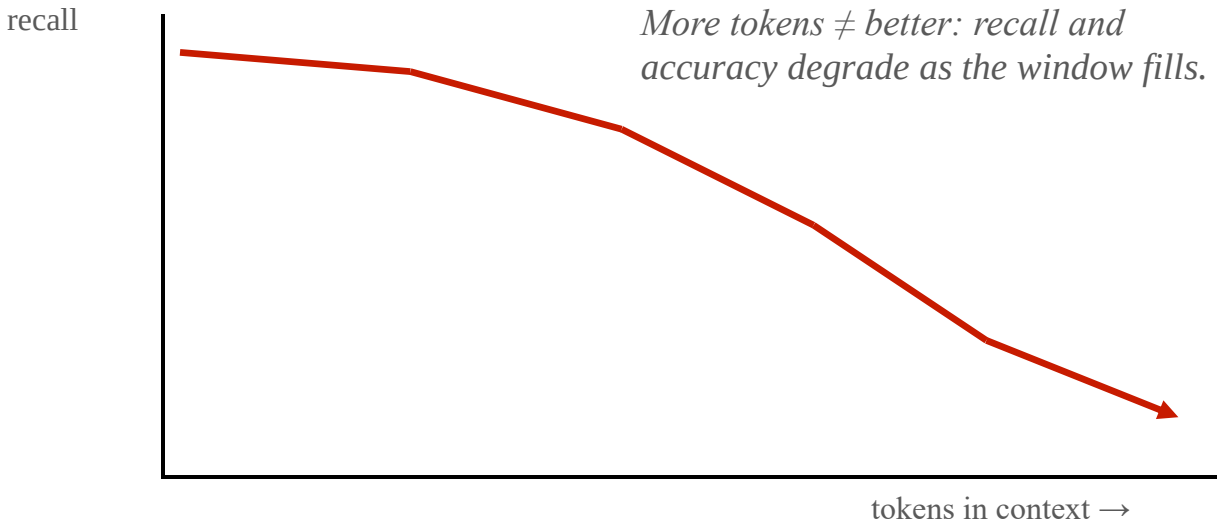
Context is a finite attention budget. What goes into the window — and what stays out — decides what the agent can do.

Context Is a Finite Attention Budget

What fills a coding agent's context window (illustrative)

system prompt	tools	AGENTS .md	conversation	tool outputs: files, logs, search results	free

Context rot



Principle

Find the smallest set of high-signal tokens that maximizes the desired outcome.

Bigger windows do not remove the need — they only move the limit.

Anthropic. Effective Context Engineering for AI Agents (Sep 2025) · Chroma. Context Rot (2025). Bar proportions are illustrative.

Four Moves of Context Engineering

Write

Save information outside the window

scratchpads, notes, PLANS.md, memory files

Select

Pull in only what is relevant now

just-in-time retrieval, grep, file reads, skill loading

Compress

Keep the gist, drop the bulk

compaction, summarizing or clearing old tool outputs

Isolate

Give sub-tasks their own window

sub-agents, sandboxed tool outputs, separate threads

LangChain. Context Engineering for Agents (2025) · Anthropic. Effective Context Engineering for AI Agents (2025).

Compaction: Summarize, then Continue

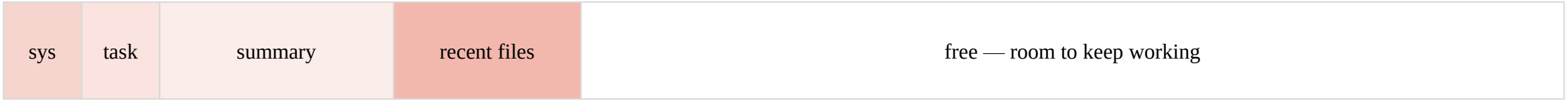
Before: the window is 95% full





Summarize: keep decisions, open bugs, current plan; drop raw tool outputs

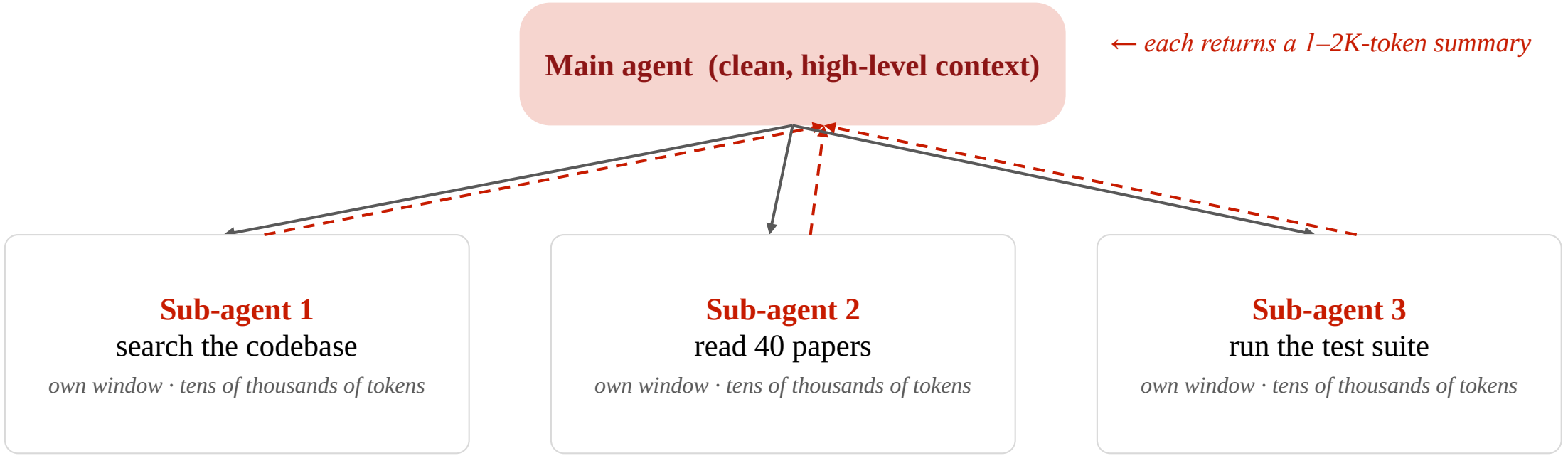
After: a fresh window



The lightest form of compaction: clear old tool results — the agent can always re-read the file.

Anthropic. Effective Context Engineering for AI Agents (2025) · Claude context-management API (2026).

Sub-Agents: Isolate the Exploration



Anthropic's multi-agent research system beat a single agent on broad research questions — at roughly 15× the tokens of a chat.

Use sub-agents when the task is wide and parallel; avoid them when every step depends on the last.

Anthropic. How We Built Our Multi-Agent Research System (2025) · Anthropic. Effective Context Engineering (2025).

4.3 Skills, Tools and MCP

Package know-how once and load it only when needed; connect every tool through one protocol.

From Prompting to Skills

Turning a **prompt** into a **skill** is useful when the same task needs to be done **reliably, repeatedly, and by more than one person**.

- **Consistency**: the same input tends to produce the same kind of output, with fewer forgotten steps.
- **Reuse**: instead of rewriting a long prompt every time, you package it once and call it whenever needed.
- **Scalability**: teams can share a skill, so knowledge stops living in one person's private prompt.
- **Composability**: skills can become building blocks in larger workflows.
- **Maintainability**: improving one skill updates the behavior everywhere it is used.
- **Lower cognitive load**: users do not need to remember a complicated prompt recipe.

Agent Skills: Progressive Disclosure

pdf-forms/SKILL.md

```
---
name: pdf-forms
description: Fill and validate PDF forms.
  Use when the user asks to fill a PDF.
---
# Filling PDF forms
1. Run scripts/extract_fields.py
2. Map user data to the fields
3. Run scripts/validate.py before saving

Edge cases: see reference/forms.md
```

Level 1 · always loaded

name + description of every skill · ≈ 100 tokens each

Level 2 · when the task matches

the SKILL.md body · < 5K tokens

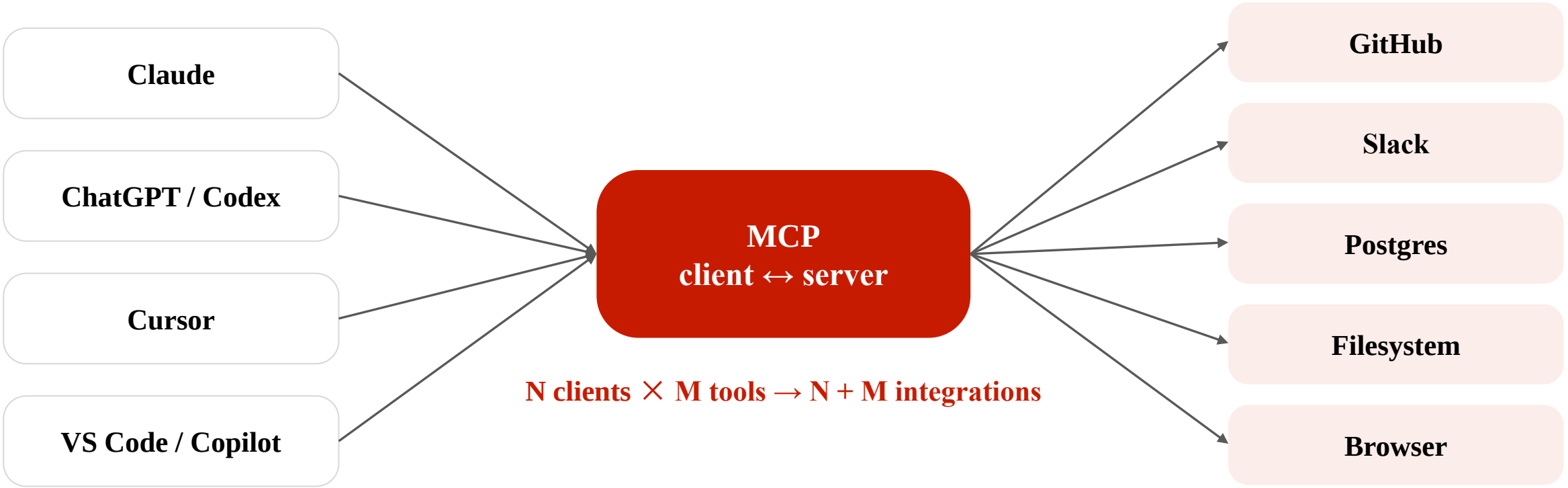
Level 3 · on demand

scripts, references, assets · read or run only if needed

Hundreds of skills cost almost nothing until one is needed. Open standard since Dec 2025; adopted by Codex, Copilot, Cursor and others.

Anthropic. Agent Skills (Oct 2025) · agentskills.io open specification (Dec 2025).

MCP: One Protocol for Tools and Data

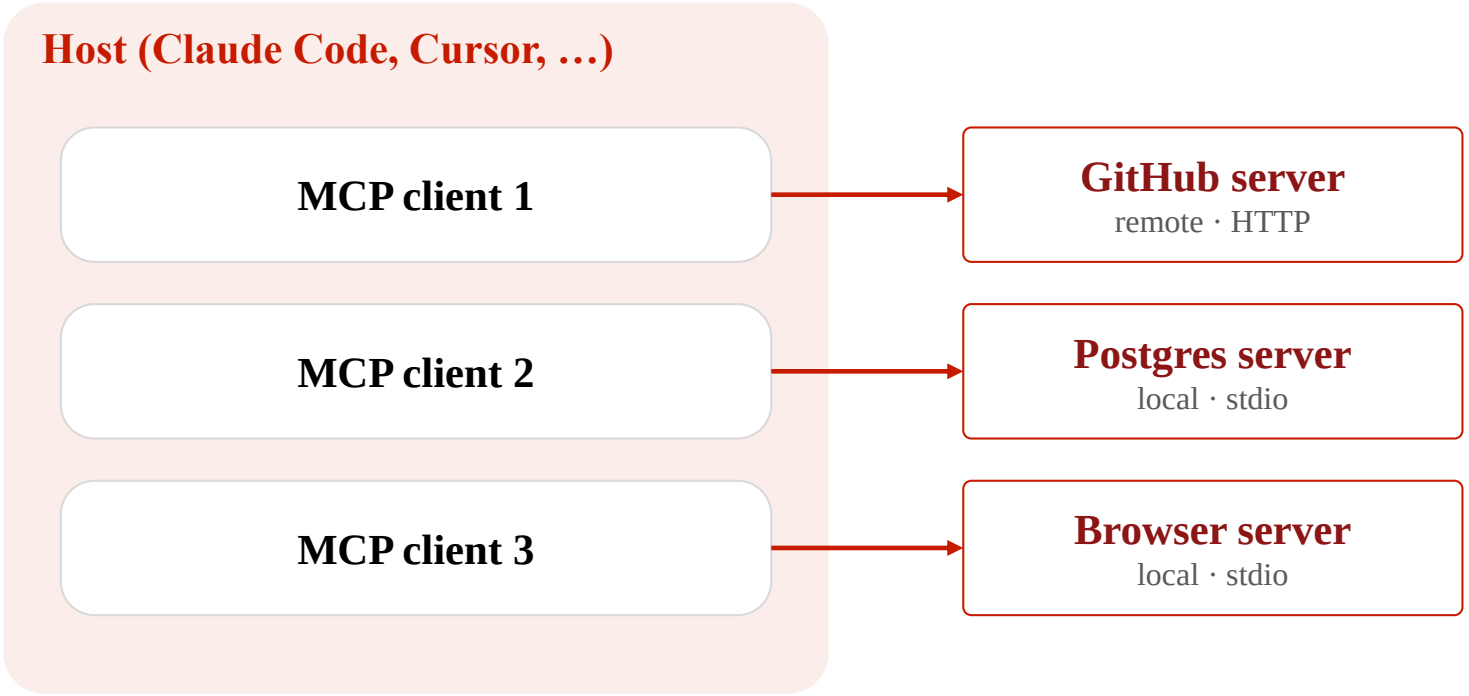


Introduced by Anthropic (Nov 2024) · donated in Dec 2025 to the Agentic AI Foundation (Linux Foundation), together with OpenAI's AGENTS.md and Block's goose.

Caveat: every MCP server is new attack surface — treat tool outputs as untrusted input.

modelcontextprotocol.io · MCP blog: MCP joins the Agentic AI Foundation (Dec 9, 2025).

Inside MCP: Hosts, Clients, Servers and Three Primitives



One client per server · JSON-RPC 2.0 · capability negotiation at start-up

Tools · model-controlled
functions the model decides to call: create_issue, run_query

Resources · application-controlled
data the host chooses to attach: files, rows, docs

Prompts · user-controlled
templates the user invokes: /review-pr

2026 spec: stateless at the protocol layer; "MCP Apps" add UI.

MCP standardizes the agent ↔ tool edge. It says nothing about agent ↔ agent — that is A2A (Part 5).

MCP specification (modelcontextprotocol.io, 2025-06-18 and 2026 releases) · CS329Z Lecture 5: Tool Use & Function Calling (Stanford, 2026).

4.4 Guides and Sensors

Steer the agent before it acts, check it after it acts — and turn every repeated mistake into a fix in the harness.

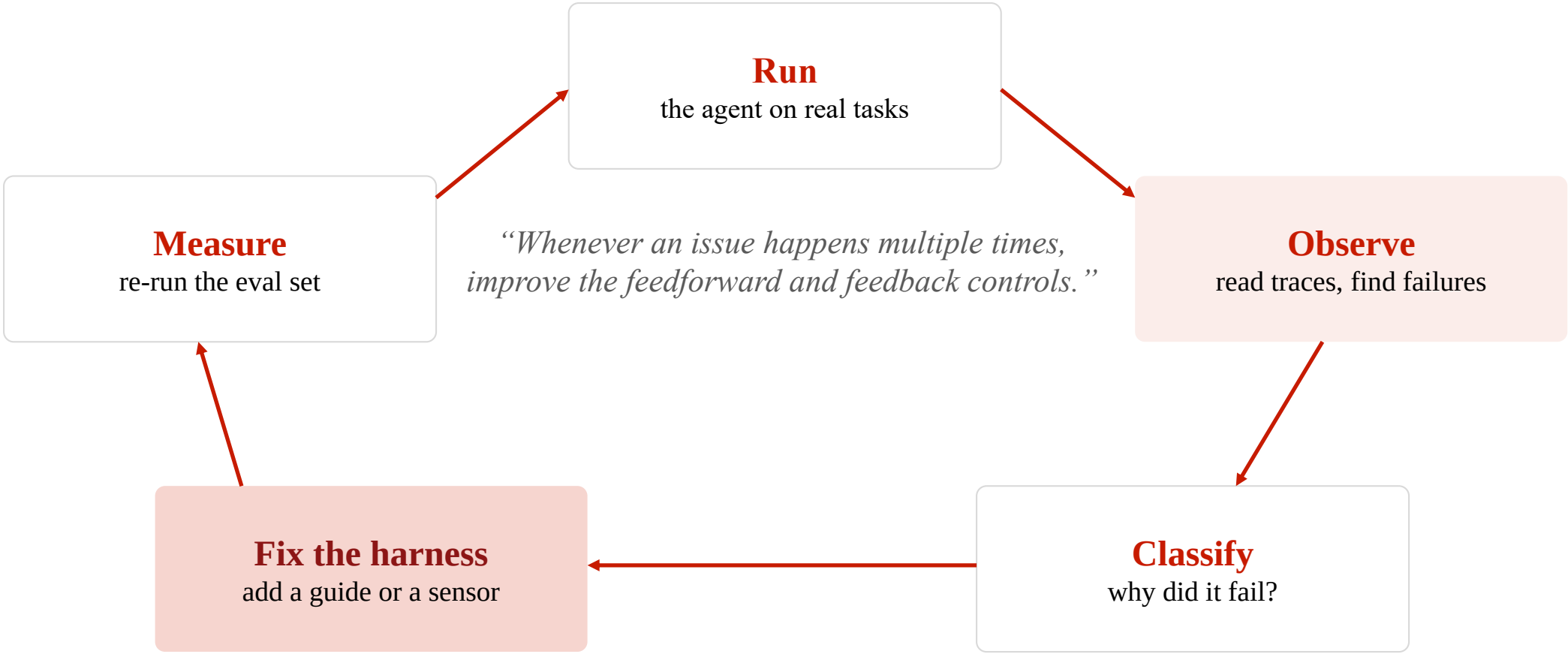
Guides and Sensors: Feedforward + Feedback

	Computational (deterministic)	Inferential (an LLM judges)
Guides feedforward: steer before acting	project templates, type systems, code generators	AGENTS.md, skills, design docs, examples
Sensors feedback: check after acting	compiler, tests, linters, structural tests	LLM code review, LLM-as-judge

Only guides: you never learn whether the rules worked. Only sensors: the agent repeats the same mistake. Use both.

Böckeler. Harness Engineering for Coding Agent Users (martinfowler.com, 2026).

The Steering Loop: Every Mistake Is a Harness Bug



Harness engineering is an ongoing practice, not a one-time configuration.

Böckeler, martinfowler.com (2026) · OpenAI. Harness Engineering (Feb 2026) · deepset (2026).

Case Study: Shipping a Product with Zero Hand-Written Code

~1M	0	5	3 → 7	~3.5
lines of code	lines written by hand	months	engineers	PRs / engineer / day

- **A map, not a manual** — a ~100-line AGENTS.md points to deeper docs
- **The repo is the system of record** — if it is not in the repo, it does not exist for the agent
- **Enforce architecture mechanically** — structural tests and custom linters
- **Errors that teach** — lint messages are written as instructions for the agent
- **Make the app legible to agents** — logs, metrics and traces the agent can query

Lopopolo (OpenAI). Harness Engineering: Leveraging Codex in an Agent-First World (Feb 2026). Team-size and PR numbers as reported in summaries of the post.

AGENTS.md: Give the Agent a Map

✘ One giant manual

- 1,000+ lines of rules in one file
- Crowds the context window
- Goes stale quickly; nobody can verify it
- Everything looks equally important

✔ A short map (~100 lines)

```
# AGENTS.md

## Build & test
- `make test` must pass before every PR

## Where things are
- Architecture: docs/ARCHITECTURE.md
- Current plan:  docs/PLANS.md
- Security:      docs/SECURITY.md

## Rules
- Never edit generated/ by hand
```

AGENTS.md is the table of contents; the details live in versioned docs the agent opens only when needed.

OpenAI. Harness Engineering (Feb 2026) · agents.md (open format, Agentic AI Foundation).

Anatomy of a Harnessed Repository

```
my-project/
├── AGENTS.md              map
├── docs/
│   ├── ARCHITECTURE.md  guide
│   └── PLANS.md         memory
├── .agents/skills/
│   └── release/SKILL.md skill
├── .mcp.json              tools
├── hooks/pre_tool.sh   permission
├── tests/ + lint config sensors
└── .github/workflows/  CI sensor
```

Guides — AGENTS.md, docs, skills

Memory — PLANS.md, progress notes, git history

Tools — MCP servers, scripts, CLI

Sensors — tests, linters, CI, LLM review

Permissions — hooks, sandbox, approvals

Illustrative layout combining OpenAI (2026), Anthropic Claude Code and Agent Skills conventions.

5 Harness Engineering in Practice

Real harnesses must coordinate agents, survive failures and be measured — and they should get simpler as models get stronger.

In this part

5.1 Anatomy of Real Harnesses

5.2 Agents Talking to Agents

5.3 Asynchrony and Fault Tolerance

5.4 Evaluating Agents and Harnesses

5.5 The Bitter Lesson of Harnesses

Three Harnesses, One Loop

The shared core

```
while True:
    reply = model(context, tools)
    if not reply.tool_calls:
        return reply.text
    results =
harness.run(reply.tool_calls)
    context += [reply, results]
```

Claude Code, Codex and OpenClaw all run this loop. They differ in everything around it: how context is built, which tools exist, who may approve what, and where the loop lives.

Claude Code

terminal coding agent

- one flat message history
- TodoWrite plan in context
- sub-agents for exploration

Codex

one core, many surfaces

- Rust “Codex core”
- App Server (JSON-RPC)
- CLI · IDE · desktop · cloud

OpenClaw

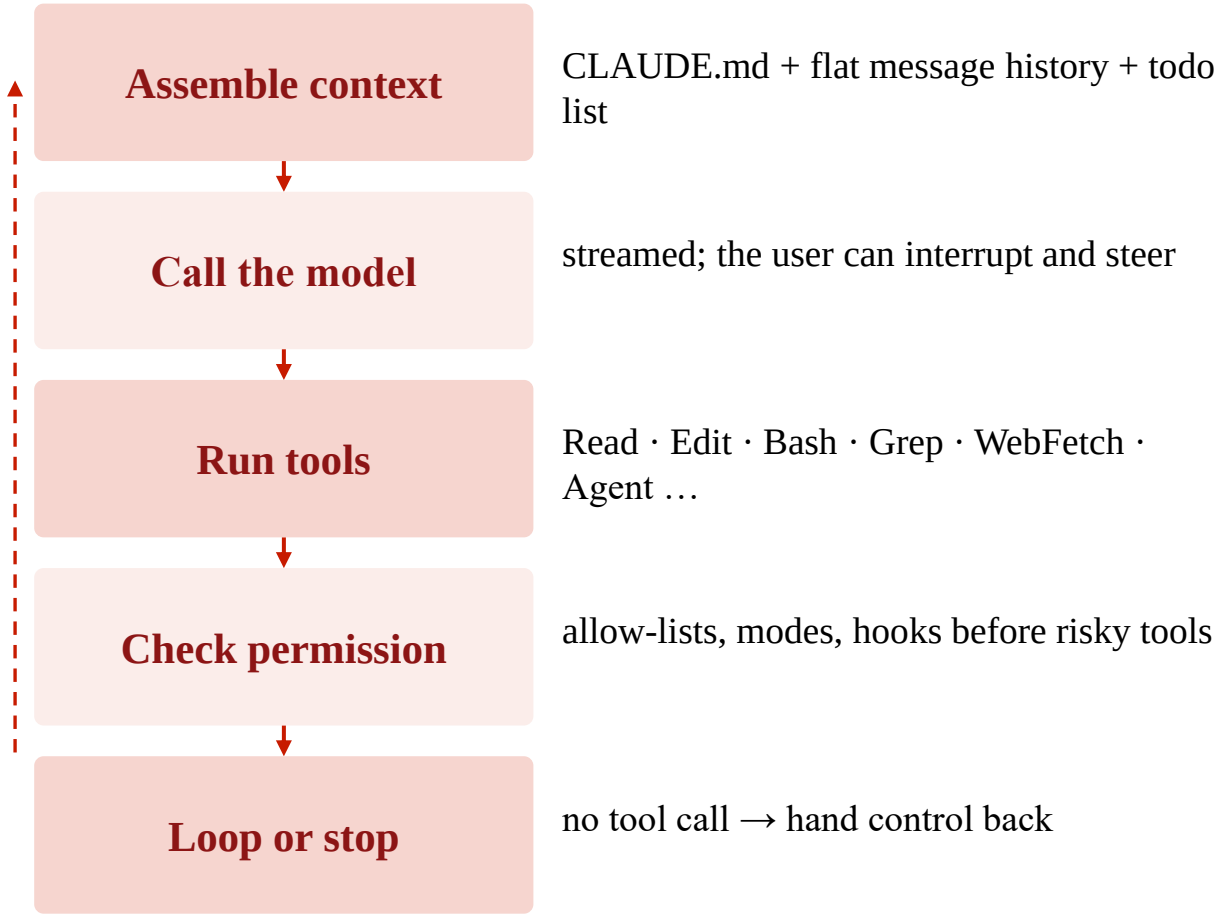
always-on personal agent

- Gateway daemon
- 30+ chat channels
- heartbeat wakes the agent

The loop is commodity. The harness — context, tools, permissions, deployment — is the product.

OpenAI. Unrolling the Codex Agent Loop (Jan 2026) · PromptLayer / ZenML analyses of Claude Code · OpenClaw docs (2026).

Claude Code: A Deliberately Simple Single Loop



Design choices

- **One thread**, no agent swarm: easier to steer, debug and trust
- **TodoWrite**: the plan is re-injected after every tool call
- **Sub-agents** = the same loop with a fresh context; return a 1–2K-token summary

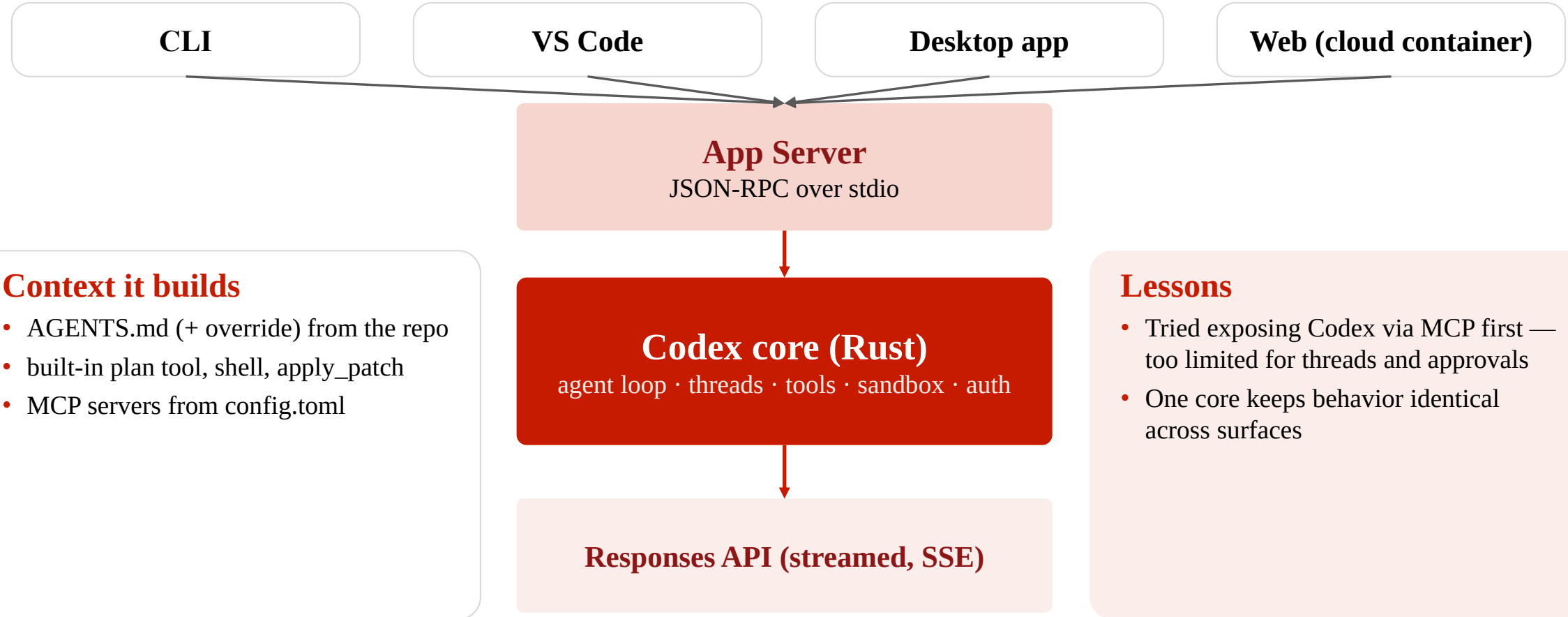
Philosophy

"Give the model tools and a goal, then get out of the way." The team treats the Bitter Lesson as required reading and keeps deleting scaffolding as models improve.

Controllable autonomy = a simple loop + disciplined tools + explicit plan + permissions.

PromptLayer. Behind the Scenes of the Master Agent Loop (2025) · Anthropic. Claude Code Best Practices (2025) · B. Cherny interviews (2025–26).

OpenAI Codex: One Harness, Many Surfaces



Context it builds

- AGENTS.md (+ override) from the repo
- built-in plan tool, shell, apply_patch
- MCP servers from config.toml

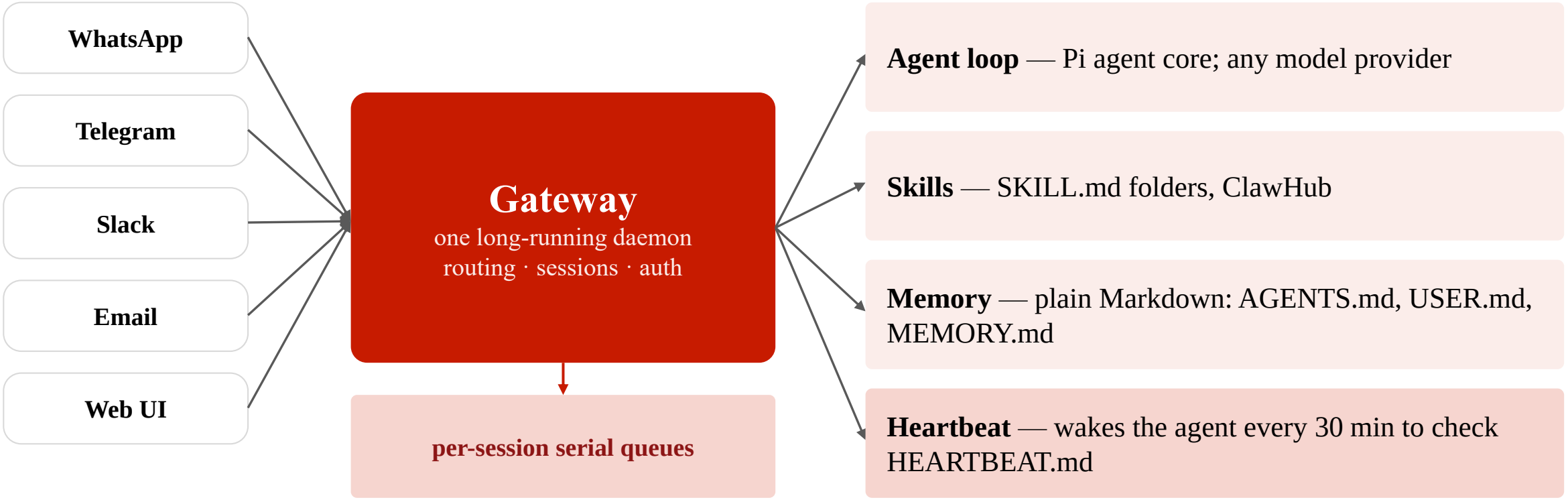
Lessons

- Tried exposing Codex via MCP first — too limited for threads and approvals
- One core keeps behavior identical across surfaces

Separate the agent core from the UI: the same harness then serves every product surface — and third parties (OpenClaw embeds it).

OpenAI. Unrolling the Codex Agent Loop (Jan 2026) · OpenAI. Unlocking the Codex Harness: How We Built the App Server (Feb 2026).

OpenClaw: A Personal Agent Is a Gateway Problem



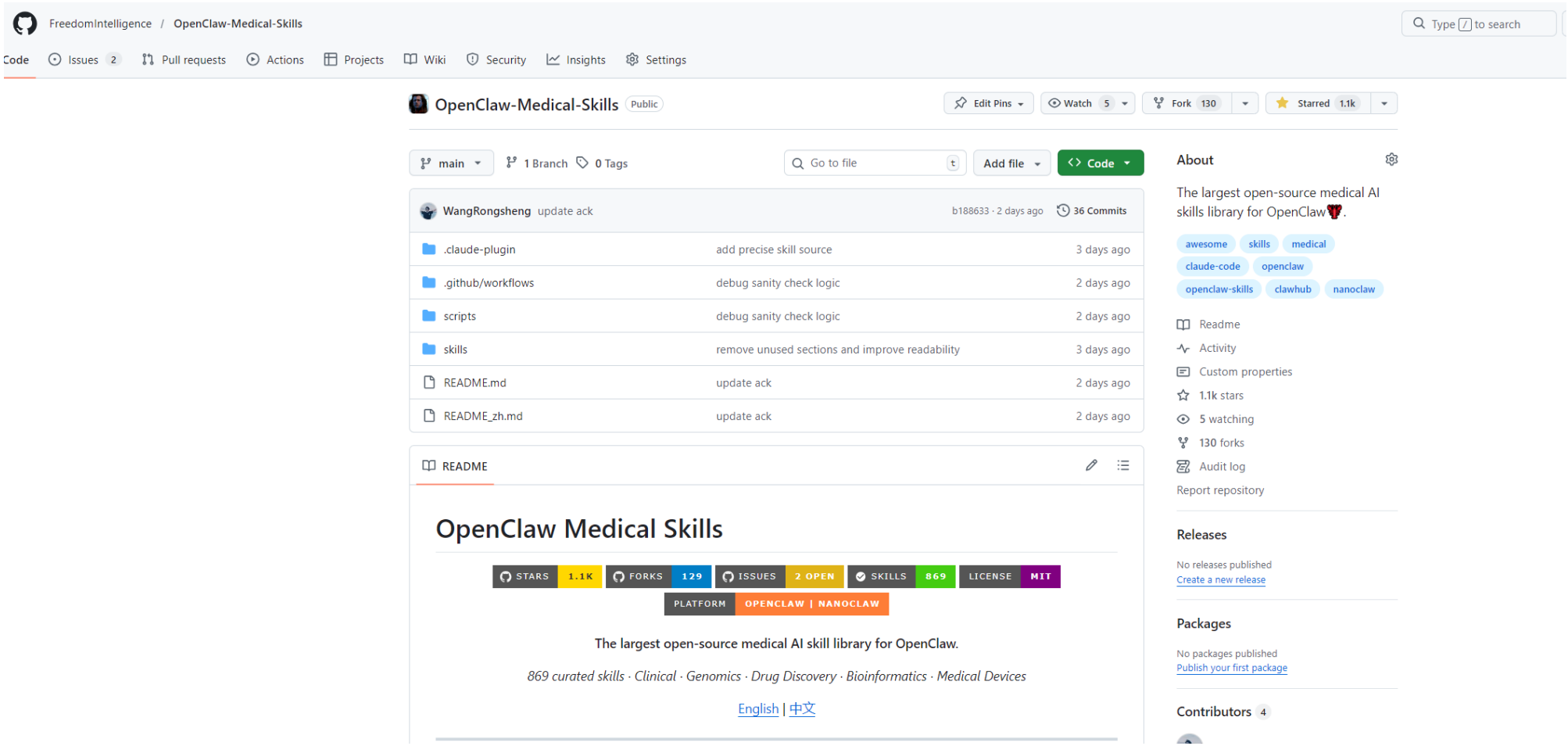
Heartbeat turns a reactive chatbot into a proactive agent — but tools run with the user's own permissions: the safety boundary is your machine.

OpenClaw docs (docs.openclaw.ai) · Agentailor. Lessons from OpenClaw's Architecture (2026) · CS329Z Lecture 18: Proactive Agents (Stanford, 2026).

OpenClaw: A Local Personal Assistant



OpenClaw Medical Skills



OpenClaw at CUHK-Shenzhen





Comparing Harness Designs

	Claude Code	Codex	OpenClaw
Where it runs	your terminal / IDE	local core; also cloud containers	always-on daemon on your machine
Who starts a turn	the user	the user (or CI)	users on any channel + heartbeat
Instructions	CLAUDE.md, skills	AGENTS.md, skills	AGENTS.md, USER.md, skills
Tools	built-in + MCP	shell, apply_patch, plan + MCP	skills + MCP; Codex as plug-in
Safety boundary	permission modes + hooks	OS sandbox + approvals	user's own permissions
Multi-agent	sub-agents (same loop)	threads; parallel cloud tasks	routing to several agents

Same loop, different bets: Claude Code bets on a simple loop, Codex on one core for every surface, OpenClaw on being always on.

Anthropic, OpenAI and OpenClaw documentation (2025–2026). Simplified for teaching.

5.2 Agents Talking to Agents

Once one agent is not enough, we need protocols for discovery, delegation and long-running tasks — and a sober view of when many agents help.

The Agent Protocol Stack

A2A agent ↔ agent	discover, delegate, track long-running tasks across vendors
MCP agent ↔ tools & data	call functions, read resources, use prompt templates
AGENTS.md · Skills agent ↔ project know-how	instructions and procedures loaded into context

Shared governance

- Agentic AI Foundation (Linux Foundation), founded Dec 2025 by Anthropic, Block and OpenAI
- Hosts MCP, AGENTS.md, goose, A2A and more
- Agent Skills: open standard since Dec 2025, adopted across vendors

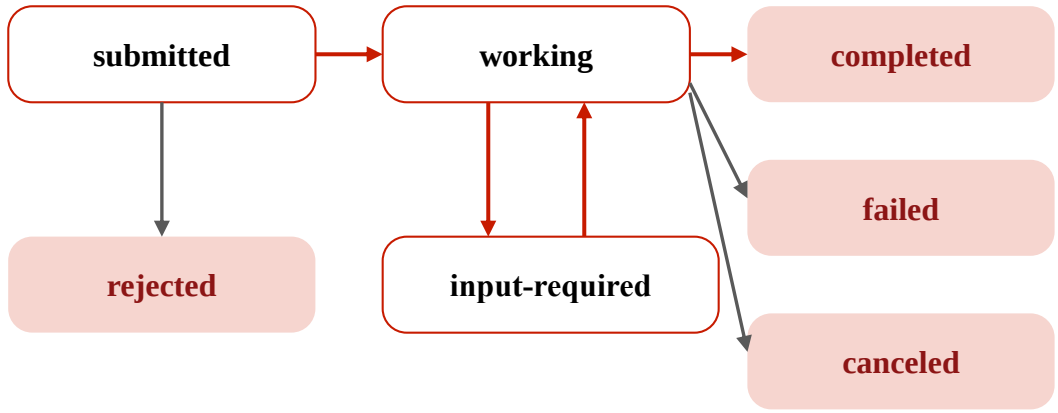
Like TCP/IP for networks: open protocols let agents from different labs compose — and let you swap any layer.

MCP blog: MCP joins the Agentic AI Foundation (Dec 2025) · a2a-protocol.org · agentskills.io · CS329Z Lecture 9: Multi-Agent Systems (Stanford, 2026).

A2A: How One Agent Hires Another

1 Discover	read the Agent Card at /.well-known/agent-card.json: name, skills, endpoints, auth
2 Delegate	send a message → the remote agent creates a Task with an id
3 Collaborate	stream updates; answer "input-required" on the same task id
4 Receive	results come back as Artifacts (files, data, text)

Task lifecycle



MCP call vs. A2A task

- MCP: a function call — returns once, usually fast
- A2A: a job — may run for hours, ask questions, stream progress, return artifacts

A2A treats the remote agent as an opaque colleague: you see its card and its outputs, never its prompts or tools.

A2A protocol specification v1 (a2a-protocol.org) · Google. Announcing A2A (Apr 2025); donated to the Linux Foundation (Jun 2025).

Asynchronous Communication: Four Patterns

Synchronous call	wait for the answer *simple, but the caller is blocked; breaks for long jobs*
Polling	ask “done yet?” every N seconds *robust, but wasteful and laggy*
Streaming (SSE)	server pushes events over an open connection *live progress; the connection must stay up*
Push / webhook	server calls you back when the state changes *best for hours-long, server-to-server work*

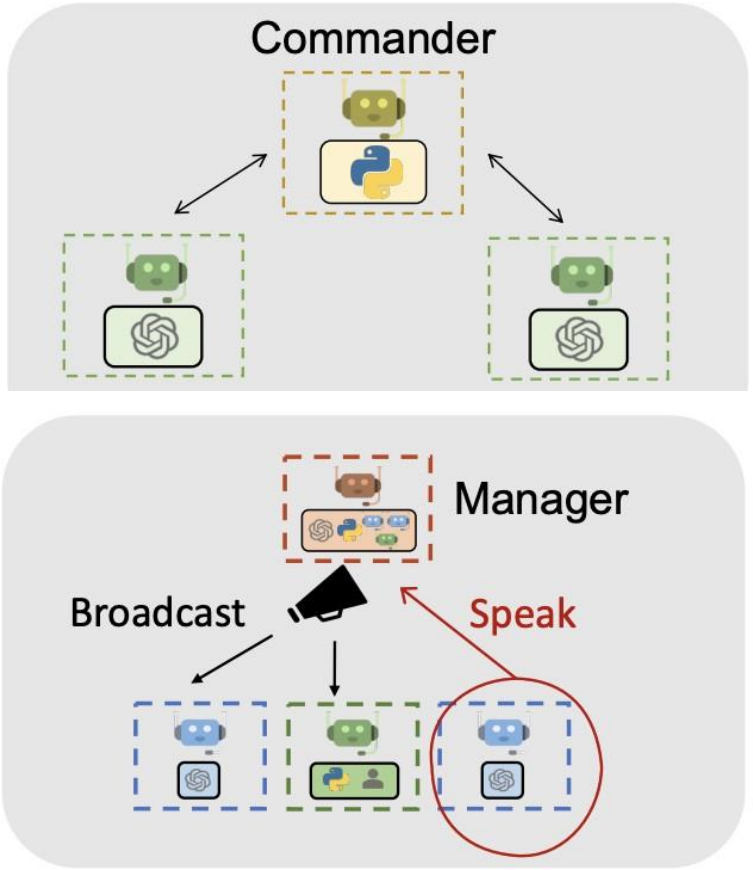


A2A supports polling, SSE streaming and push notifications; long agent jobs should never block a caller.

A2A specification v1: transports and update delivery (a2a-protocol.org) · OpenAI Responses API streaming.

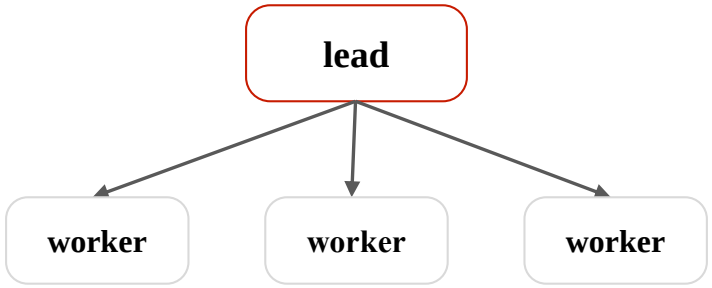
Multi-Agent Orchestration

- Static/dynamic
- NL/PL
- Context sharing/isolation
- Cooperation/competition
- Centralized/decentralized
- Intervention/automation



Multi-Agent Patterns

Orchestrator–workers



a lead agent splits the task and merges results

good for: broad research, large refactors

Pipeline



planner → coder → reviewer

good for: well-understood workflows

Debate / critic



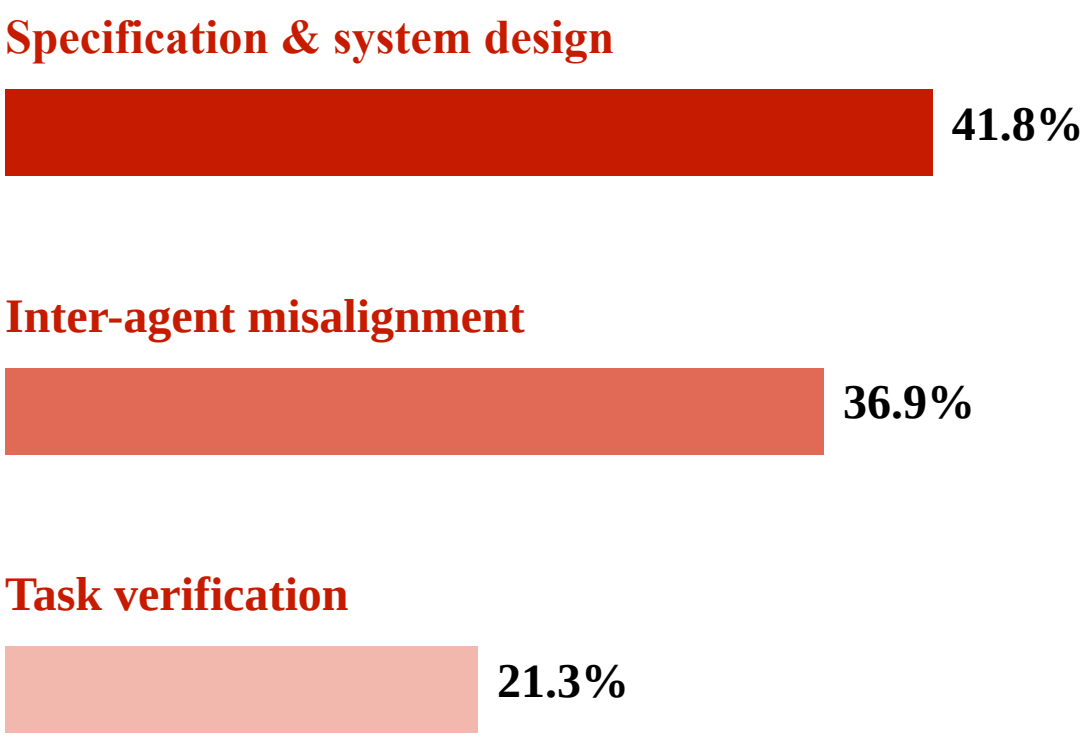
agents argue or review each other

good for: hard judgments, safety review

More agents ≠ better. Each agent adds tokens, latency and coordination failures — add one only when the task is truly parallel.

Anthropic. Building Effective Agents (2024) · Anthropic. Multi-Agent Research System (2025) · Wu et al. AutoGen (2023).

Why Do Multi-Agent Systems Fail?

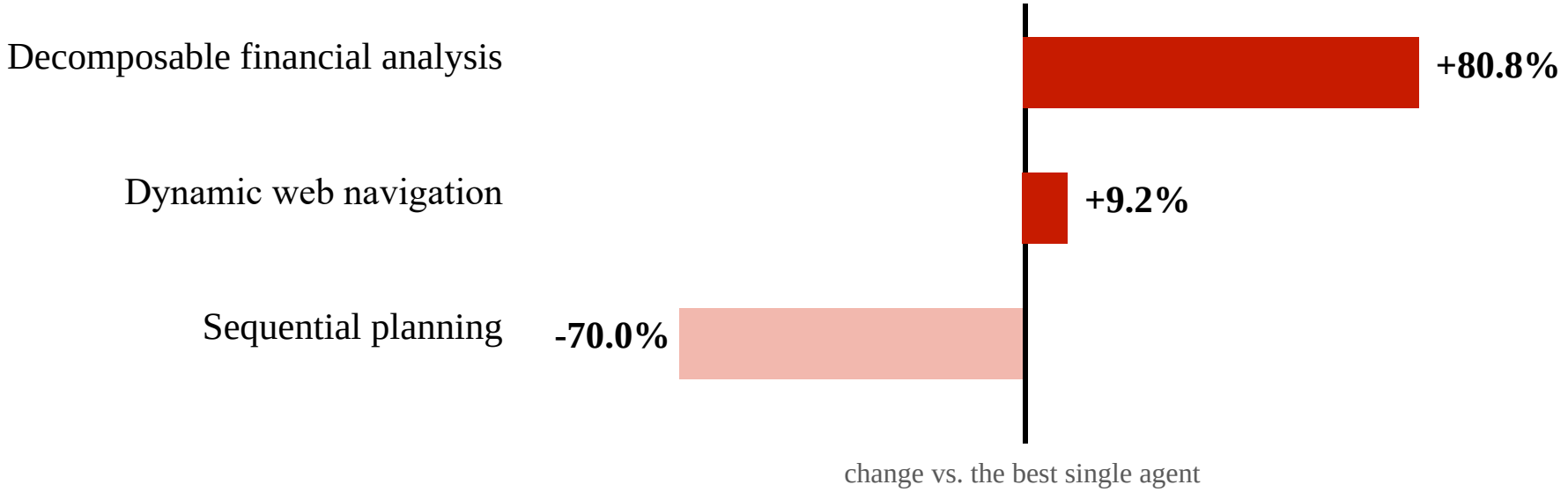


"Many failures arise from organizational design and agent coordination rather than from the limits of individual agents."

Cemri et al. Why Do Multi-Agent LLM Systems Fail? (MAST: 14 failure modes, 7 frameworks; NeurIPS 2025 D&B, arXiv:2503.13657).

When Do More Agents Help?

260 controlled configurations · 6 benchmarks · 5 architectures · 3 model families



- **Coordination tax**: global context must be squeezed into messages between agents.
- **Saturation**: once a single agent already scores ~45%, adding agents tends to hurt.
- **Rule of thumb**: parallel & decomposable → multi-agent with a central verifier; sequential → one agent with tools.

Kim et al. Towards a Science of Scaling Agent Systems (arXiv:2512.08296, 2025; current version) · CS329Z Lecture 9 (Stanford, 2026).

5.3 Asynchrony and Fault Tolerance

Agents run for hours, call flaky tools and crash. A production harness must pause, resume, retry and recover without losing state.

Fault Tolerance: What Breaks, and What the Harness Does

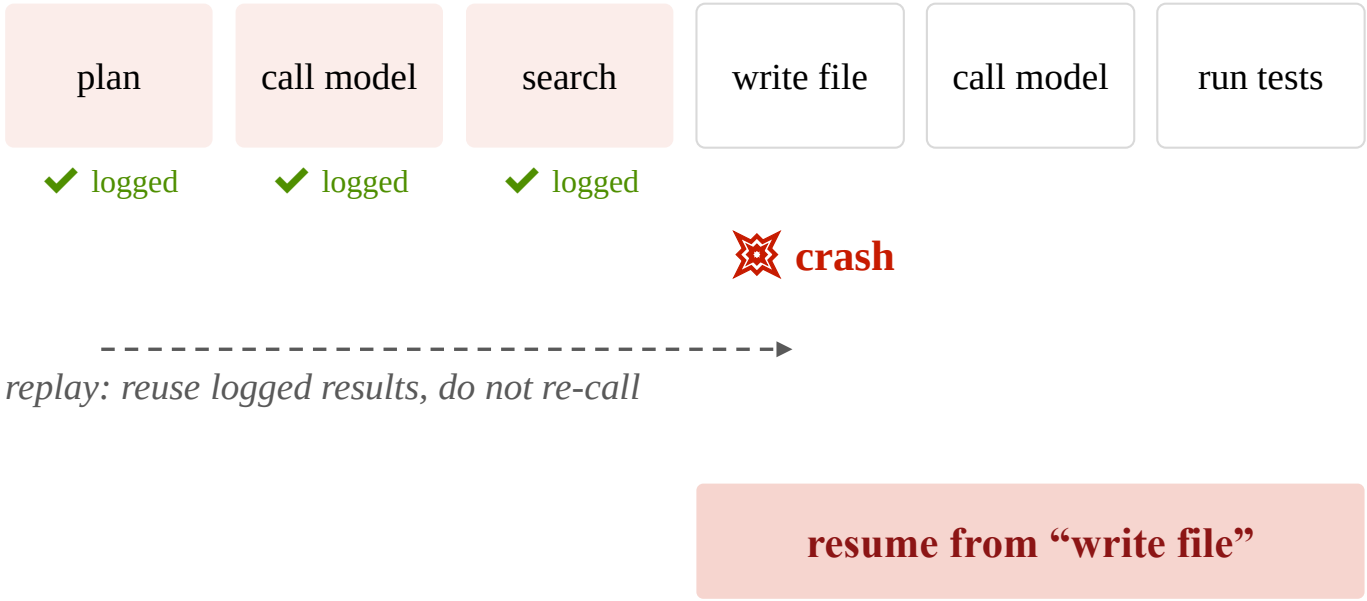
Failure	Harness response
Model API timeout / rate limit	retry with exponential backoff; fall back to another model
Tool error or bad arguments	return the error text to the model — let it self-correct
Agent loops or stalls	step and token budgets; loop detection; escalate to a human
Context overflow	compaction; offload outputs to files; start a fresh session
Process crash mid-task	checkpoints: durable state, git commits, progress files
Duplicate side effects on retry	idempotency keys; read-before-write; dry runs
Wrong "done"	verification gate: tests / end-to-end checks before marking complete

Agents fail in normal, expected ways. A harness turns those failures into retries, checkpoints and escalations instead of disasters.

Anthropic. Effective Harnesses for Long-Running Agents (Nov 2025) · Temporal / Inngest durable-execution docs · CS329Z Lecture 6 (Stanford, 2026).

Durable Execution: Crash, Replay, Resume

An agent workflow as a durable log of steps



The model (e.g. Temporal)

- **Workflow**: deterministic orchestration; its state is persisted at every step
- **Activities**: model calls, tools, APIs — retried automatically
- **At-least-once**: activities may run twice → make writes idempotent
- **Human approval** = pause for hours or days without losing state

Caveat: LLM calls are not deterministic — a retried call may answer differently, so log results instead of recomputing them.

Temporal. Durable AI agents with Pydantic AI and Temporal (2025) · Inngest. Durable Execution for AI Agents · CS329Z Lecture 6: durable orchestration.

Long Tasks: Outliving the Context Window



Persistent artifacts shared by all sessions

PLANS.md	progress notes	git commits	tests
the goal and the step list	what is done, what is next	checkpoints you can roll back to	proof that a step really works

Start every session by reading the plan; end it by committing and updating notes; never claim "done" without passing tests.

Anthropic. Effective Harnesses for Long-Running Agents (2025) · OpenAI. Harness Engineering (Feb 2026).

Case Study: A Harness for Multi-Session Coding

Initializer agent

first session only

- writes init.sh to start the app
- expands the prompt into a feature list (200+ items, all “passes”: false)
- creates progress log + first git commit

Coding agent

every later session

- read progress log and git log
- run init.sh + a smoke test
- implement ONE feature, test it end-to-end, commit, update the log

feature_list.json

```
{"category": "chat",
 "description": "user can rename
   a conversation",
 "steps": ["open sidebar", …],
 "passes": false}
```

Observed failure	Harness fix
Tries to one-shot the whole app	**one feature per session**
Declares the project done too early	**feature list with explicit pass flags**
Marks features done without real tests	**browser automation as a user would test**

Young (Anthropic). Effective Harnesses for Long-Running Agents (Nov 26, 2025). JSON is used because models overwrite it less often than Markdown.

Safety: Agents Touch Real Systems

What can go wrong

- **Prompt injection** — instructions hidden in web pages, files or tool outputs
- **Destructive actions** — deleting files, force-pushing, dropping tables
- **Data exfiltration** — secrets sent to an outside URL
- **Untrusted tools** — a malicious MCP server or skill

Harness controls

- **Permission modes** and human approval for risky actions
- **Sandbox**: file-system and network allow-lists
- **Hooks**: deterministic checks before and after each tool call
- **Least privilege**: scoped tokens, read-only by default
- **Traces**: log every action for audit

Treat everything the model reads as untrusted, and everything it does as needing a permission.

OWASP Top 10 for LLM Applications (2025) · Anthropic Claude Code security docs · OpenAI Codex sandboxing docs.

5.4 Evaluating Agents and Harnesses

You cannot improve a harness you cannot measure. Agent evals grade outcomes in an environment, many times over.

An Agent Eval Is a 4-Tuple

Request	Environment	Stopping criteria	Scorer
what the agent is asked to do	the world it acts in, reset for each trial	when the episode ends	how the final outcome is graded
"Fix issue #482 in this repo"	*a container with the repo at commit abc123*	*agent says done · 50 steps · 30 minutes*	*hidden unit tests pass*

Vocabulary: **task** (one test) · **trial** (one attempt) · **transcript** (everything that happened) · **outcome** (final state of the environment)

Grade the outcome — is the booking in the database? — not what the agent claims it did.

You always evaluate model + harness together: change the harness and you have a new system under test.

Graders, and Why pass^k Matters

Code-based

tests, string match, state checks

fast, cheap, objective — but brittle

Model-based

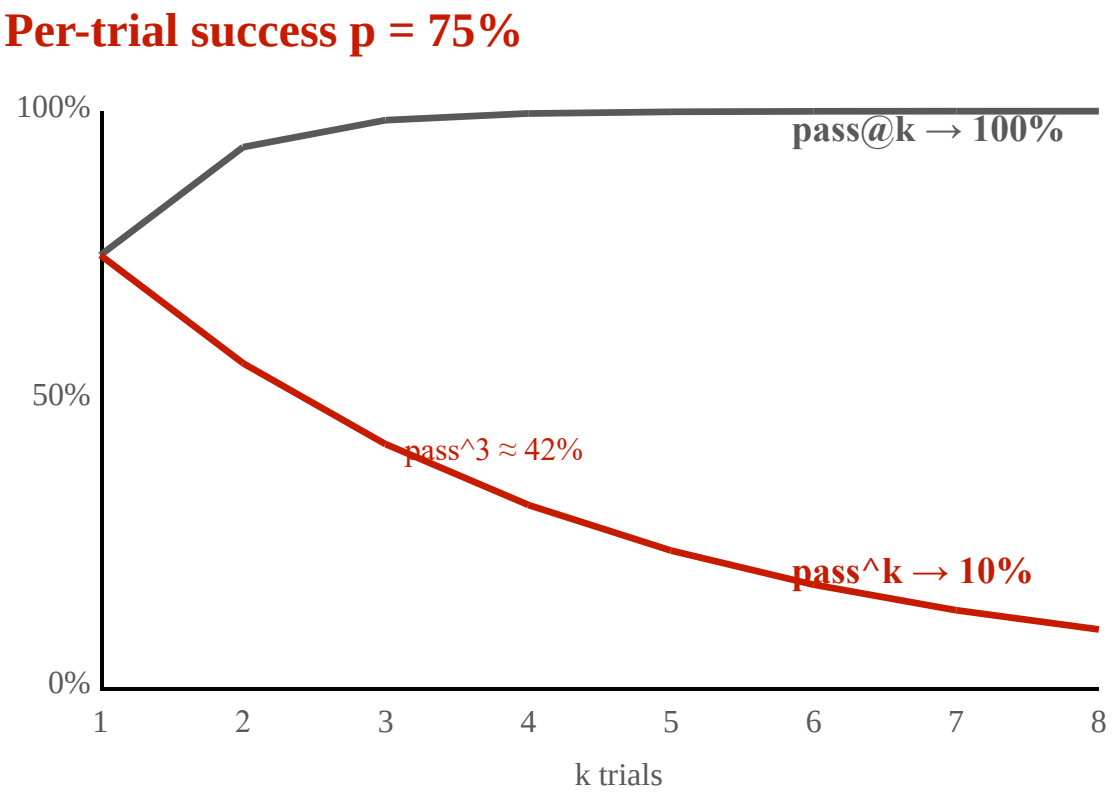
rubrics, LLM-as-judge, pairwise

flexible — but needs calibration (judge bias!)

Human

expert review, A/B tests

gold standard — but slow and costly



pass@k: "can it ever do it?" (research). pass^k: "does it do it every time?" (products). Users experience pass^k.

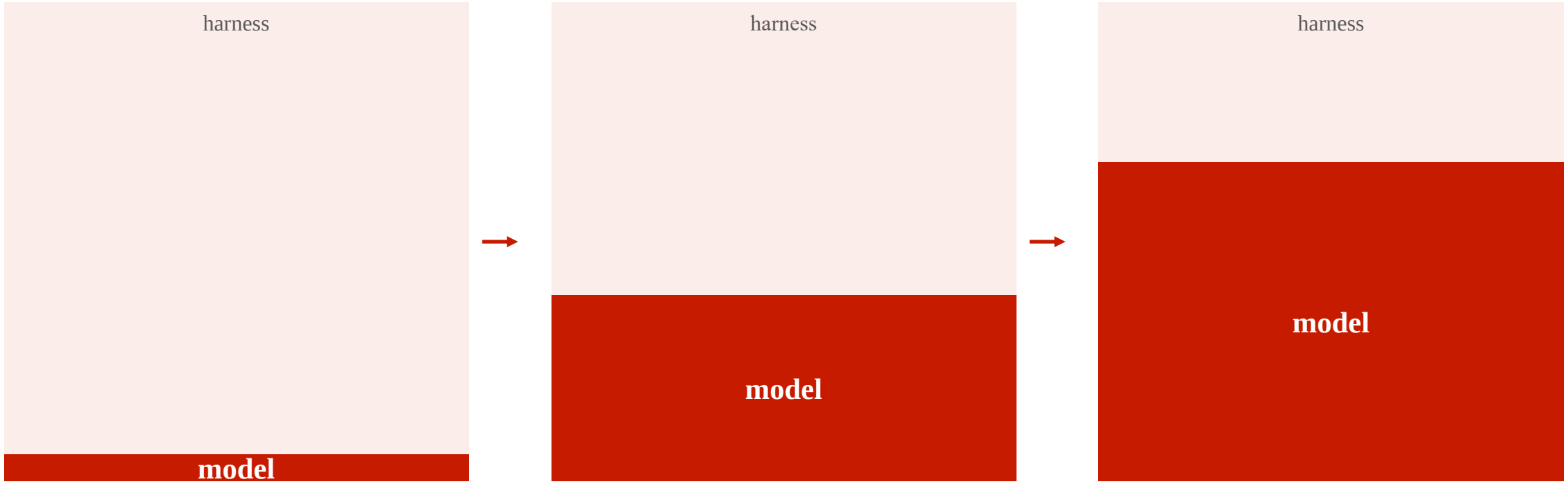
Anthropic. Demystifying Evals for AI Agents (Jan 2026) · Yao et al. τ-bench (2024) · CS329Z Lecture 15: LLM-as-Judge & Evaluation Infrastructure.

5.5 The Bitter Lesson of Harnesses

As models get stronger, the best harness gets thinner. What should we build — and what should we delete?

The Bitter Lesson, Applied to Harnesses

"General methods that leverage computation are ultimately the most effective, and by a large margin." — Rich Sutton (2019)



thick scaffolding: rigid workflows, long prompts, many special tools

fewer rules: the model plans and recovers on its own

thin harness: goal + tools + permissions + verification

Sutton. The Bitter Lesson (2019) · B. Cherny on Claude Code: scaffolding gains of ~10–20% are often erased by the next model (interviews, 2025–26).

What Moves into the Model, What Stays in the Harness

Moves into the model (post-training, RL)	Stays in the harness
• Planning and task decomposition	• Permissions, sandboxes, approvals
• Tool-call formats and error recovery	• State, checkpoints, durability
• Self-verification (“run the tests first”)	• Connectors: MCP servers, credentials
• When to stop, when to ask	• Evals and observability
• Prompt tricks and role-play scaffolds	• The environment itself — repos, browsers, hospitals

Evidence: models are now post-trained inside their own harnesses (Codex models with the Codex tools), and teams report deleting large parts of system prompts when a new model arrives.

Build for the model six months out: keep what enforces safety and state; expect the rest to be absorbed.

Trivedy, LangChain (2026): models can overfit to the harness they were trained with · Cherny interviews (2025–26) · OpenAI Codex docs. Reported prompt-deletion figures are secondhand.

Ten Principles of Harness Engineering

1. **Start with one loop**
   add agents only when the task is parallel
2. **Context is a budget**
   smallest set of high-signal tokens
3. **Map, not manual**
   short AGENTS.md pointing to versioned docs
4. **Load just in time**
   skills and retrieval instead of preloading
5. **Guides + sensors**
   steer before, verify after
6. **Errors that teach**
   tool and lint errors written for the model
7. **Durable by default**
   checkpoints, idempotent writes, resumable runs
8. **Least privilege**
   sandbox, allow-lists, human approval
9. **Measure pass^k**
   outcome-graded evals in clean environments
10. **Delete regularly**
    re-test without scaffolds on every new model

Synthesized from Anthropic (2024–26), OpenAI (2026), Böckeler (2026), LangChain (2026) and Stanford CS329Z (2026).

Outlook (Late 2026): Where Harness Engineering Is Going

Harnesses that evolve themselves
agents read their own traces and rewrite their harness (e.g. observability-driven automatic harness evolution, 2026)

Open protocol stack
MCP, A2A, AGENTS.md and goose under one foundation — 280+ member organizations by Oct 2026

Proactive agents
heartbeats and user models: agents that act before being asked (OpenClaw, CS329Z “Proactive Agents”)

Harness-aware leaderboards
Terminal-Bench reports every (model, harness) pair; harness choice moves scores by 10–20 points

The harness becomes the RL environment
the same tools, sandboxes and graders are used to train the next model (Lecture 6)

Models will keep absorbing the scaffolding. The durable skills are designing environments, protocols, safety boundaries and evals.

arXiv:2604.25850 (Agentic Harness Engineering, 2026) · Agentic AI Foundation statistics (Oct 2026) · tbench.ai · CS329Z (Stanford, Fall 2026).

6 Agents in the Wild

Where agents already work — and how we measure whether they really do.

In this part

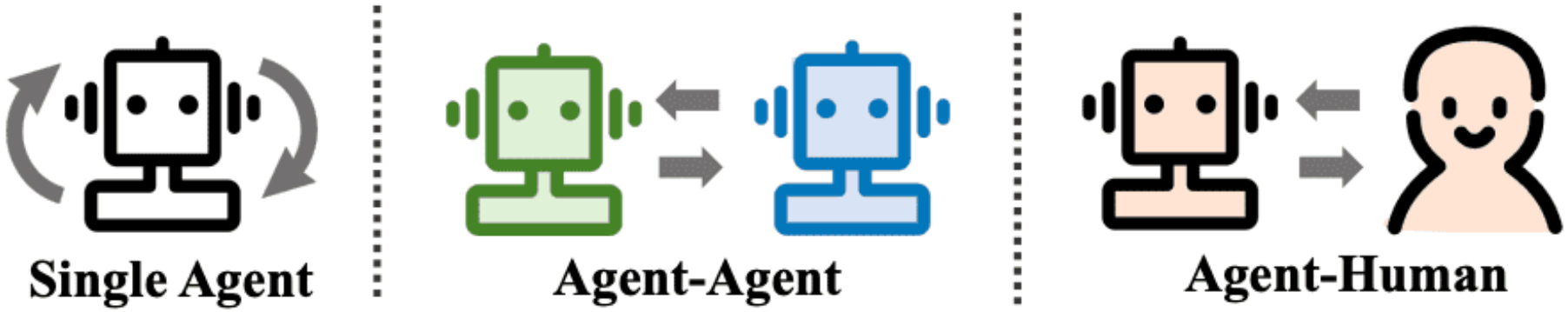
6.1 Applications

6.2 Benchmarks

Three Kinds of Agent Applications

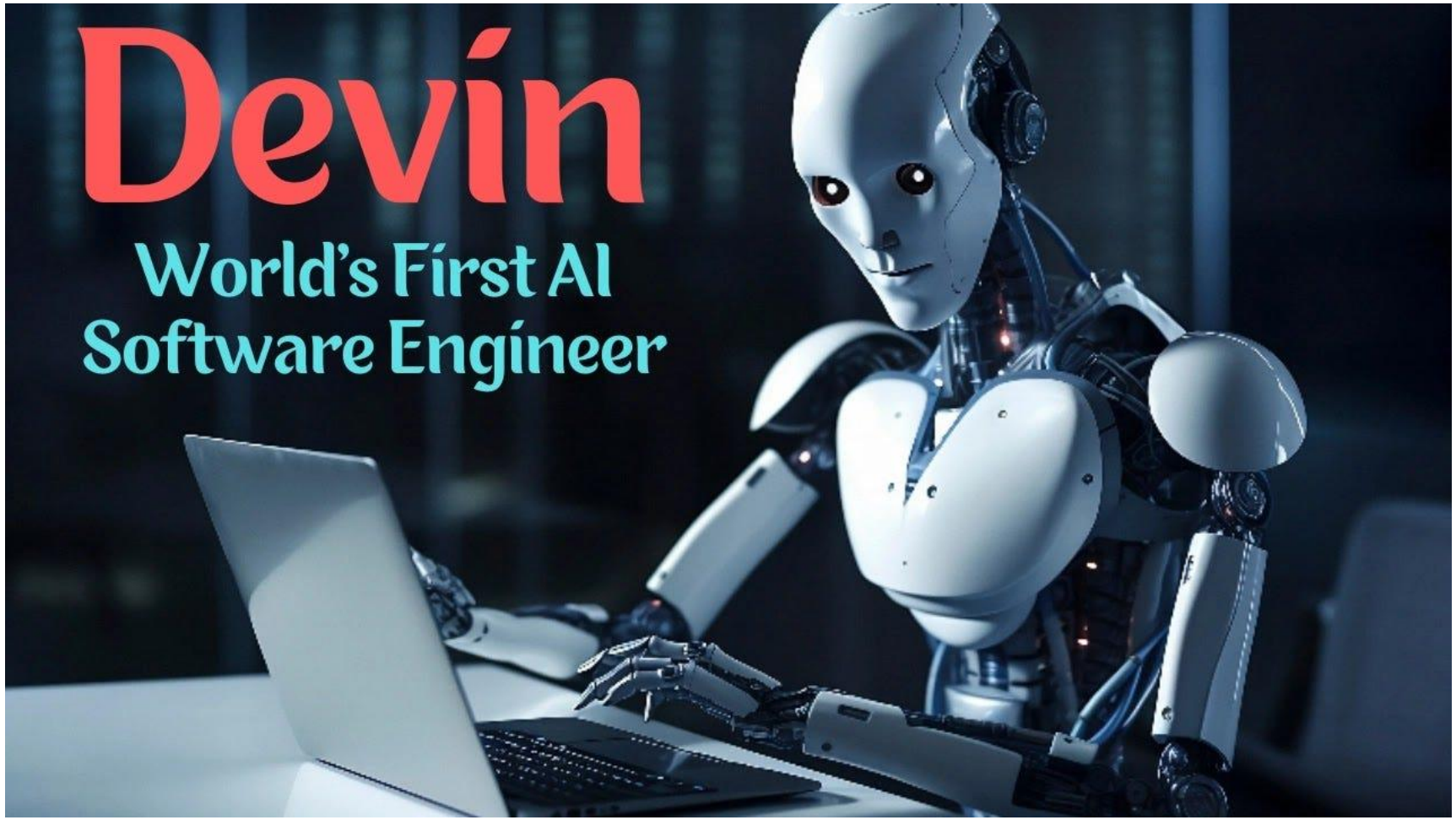
The use cases for LLM agents, or Language Model-based agents, are vast and diverse. These agents, powered by large language models (LLMs), can be used in various scenarios, including:

1. Single-agent applications
2. Multi-agent systems
3. Human-Agent cooperation



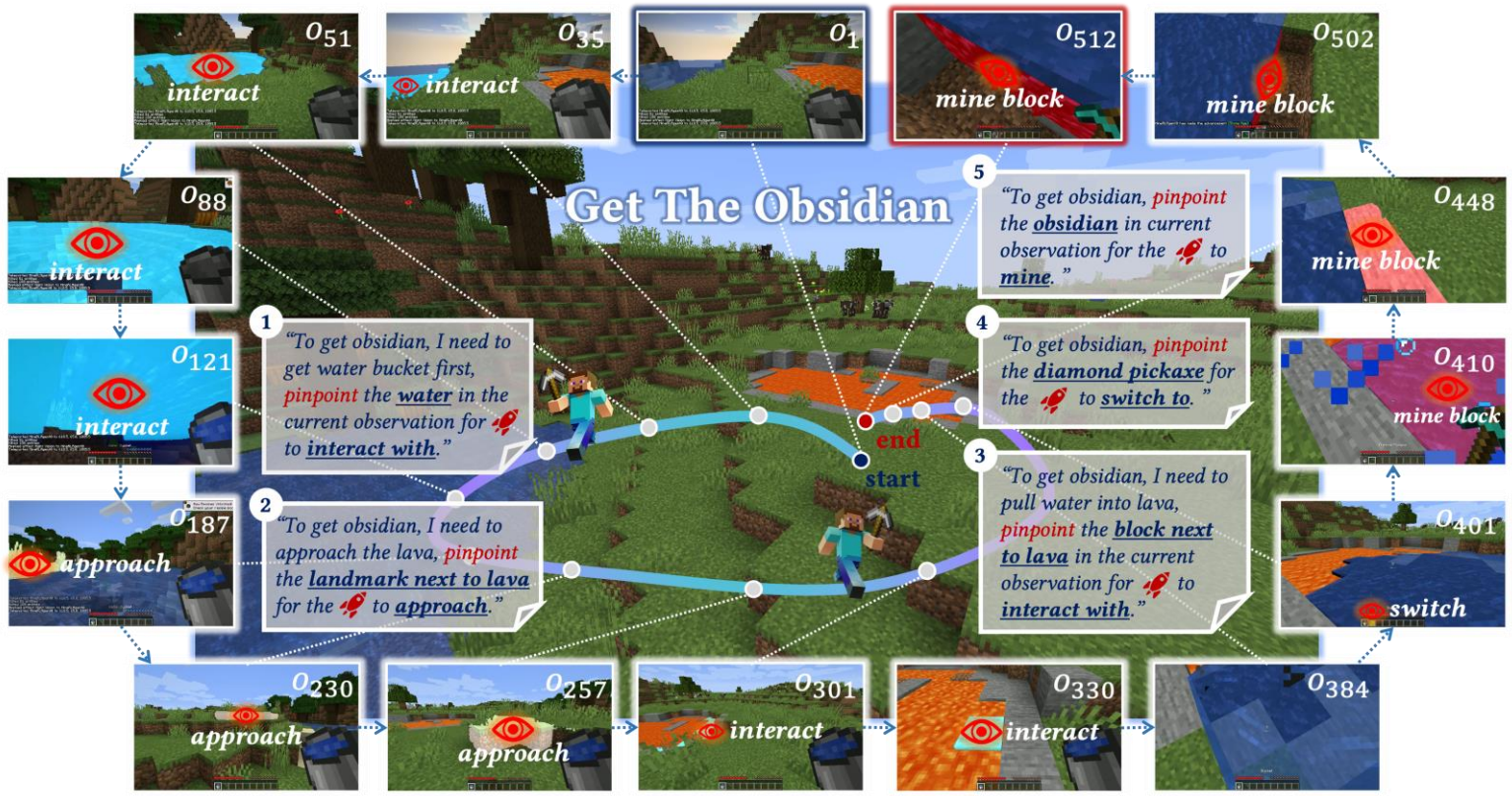
https://gptpluginz.com/llm-agents/

Devin: An AI Software Engineer (2024)



https://www.cognition-labs.com/introducing-devin

Open-World Interaction: Minecraft (ROCKET-1)



https://craftjarvis.github.io/ROCKET-1/

Shaofei Cai, Zihao Wang, Kewei Lian, Zhancun Mu, Xiaojian Ma, Anji Liu, Yitao Liang. ROCKET-1: Master Open-World Interaction with Visual-Temporal Context Prompting. https://arxiv.org/abs/2410.17856

Multi-Agent Games: Werewolf

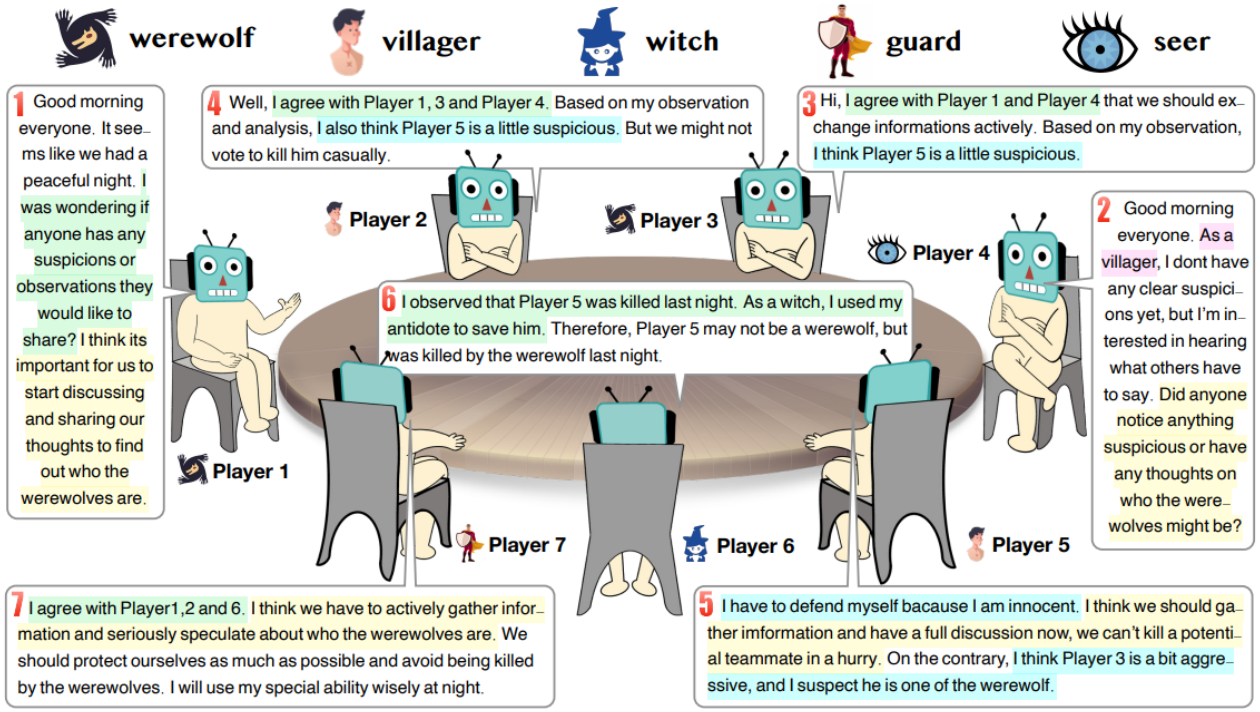
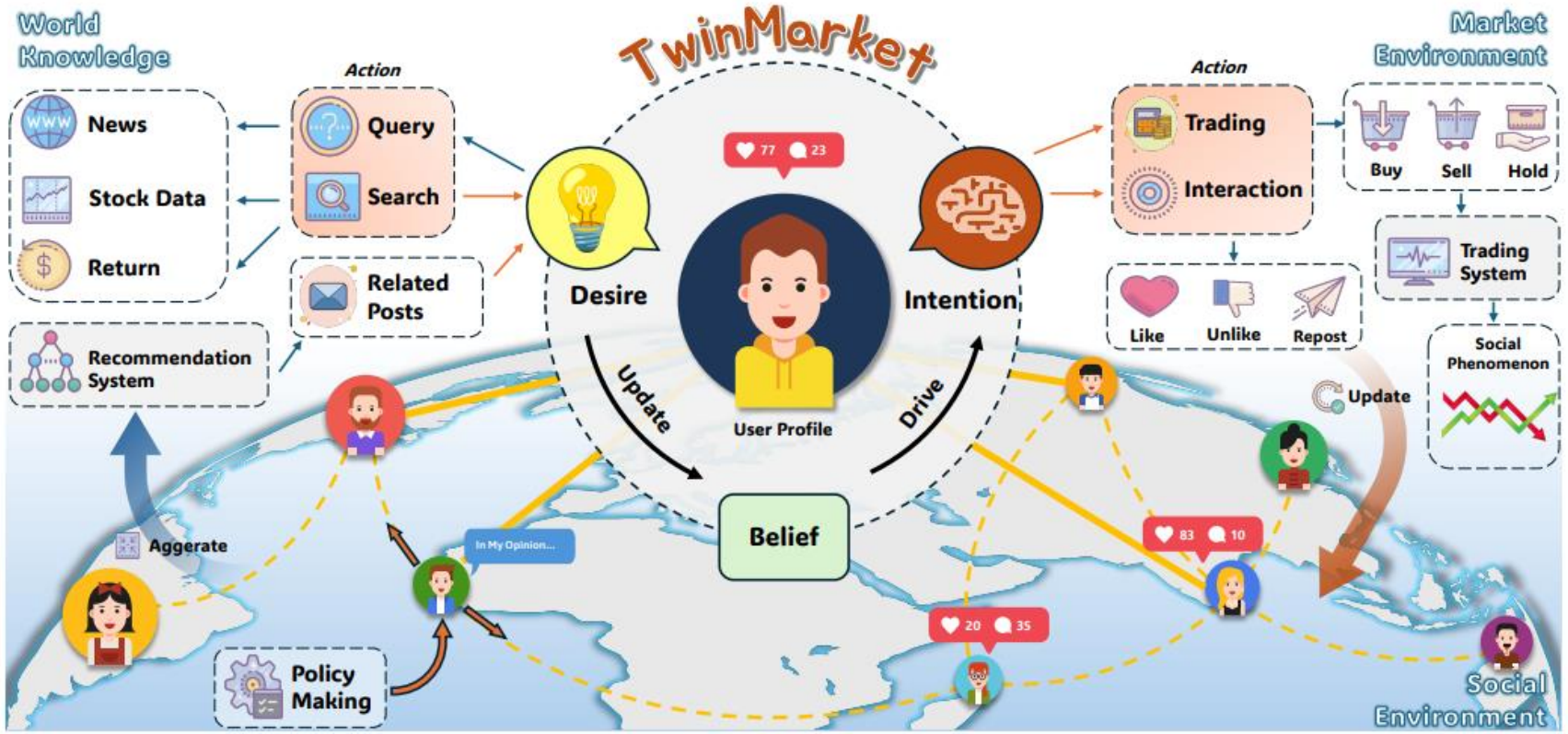


Figure 1: A snapshot of our implemented Werewolf game. There are 5 roles and 7 players, and each of them is acted by an LLM autonomously. The number before each talking denotes the speaking order. Some social behaviors can be primarily observed in this figure, including trust , confrontation , camouflage , and leadership .

Yuzhuang Xu , Shuo Wang, Peng Li,, Fuwen Luo, Xiaolong Wang , Weidong Liu, Yang Liu. Exploring Large Language Models for Communication Games: An Empirical Study on Werewolf. https://arxiv.org/pdf/2309.04658.pdf

TwinMarket: Simulating a Financial Market

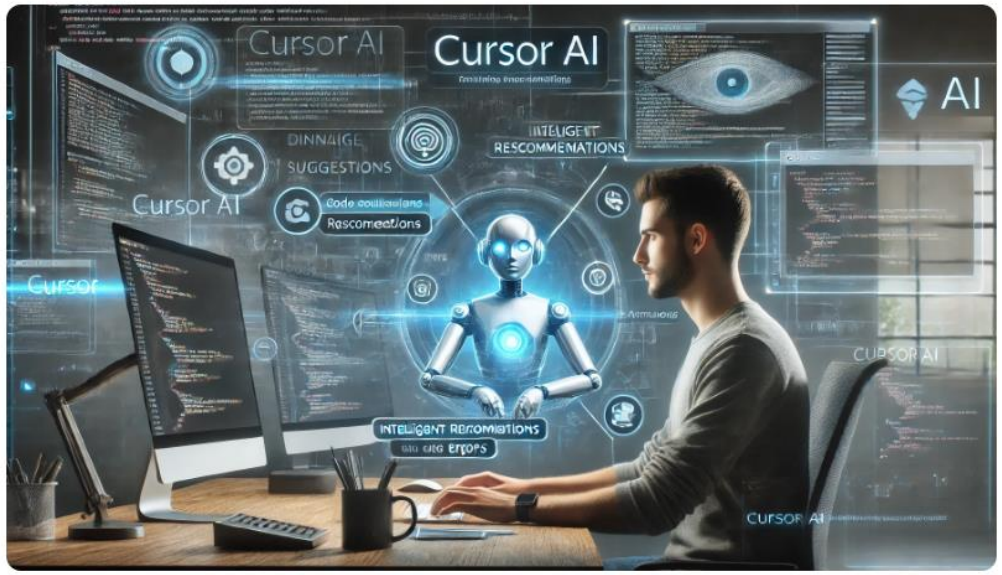


Yuzhe Yang, Yifei Zhang, Minghao Wu, Kaidi Zhang, Yunmiao Zhang, Honghai Yu, Yan Hu, Benyou Wang. TwinMarket: A Scalable Behavioral and Social Simulation for Financial Markets. https://arxiv.org/abs/2502.01506

Coding Alone vs. Coding with Agents



Coding with yourself



Coding with LLM agents

6.2 Benchmarks

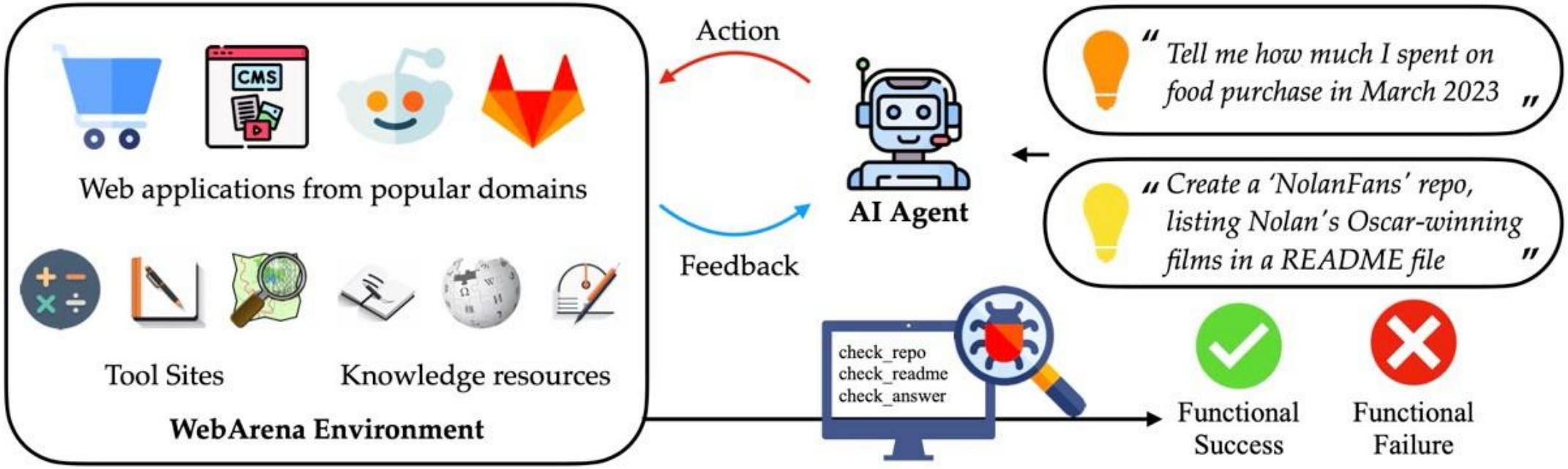
How do we know an agent works? Realistic environments with automatic checks.

WebShop (2022)



- Large-scale complex environment based on 1.16M Amazon products
- Automatic reward based on instruction and product attribute matching
- Challenges language and visual understanding, and decision making

WebArena (2023)



SWE-bench (2023)

Metadata

Repo	scikit-learn/scikit-learn	Issue #s	[14858]
Instance ID	scikit-learn__scikit-learn-14869	Pull Number	14869
Created At	Aug 31, 2019	Base Commit	1018f9f...

Problem Statement

HGBC with categorical_crossentropy fails silently on binary classification

```
import numpy as np
from sklearn.experimental import enable_hist_gradient_boosting
from sklearn.ensemble import HistGradientBoostingClassifier

X = [[1, 0], [1, 0], [1, 0], [0, 1], [1, 1]]
y = [1, 1, 1, 0, 1]
gb = HistGradientBoostingClassifier(loss='categorical_crossentropy',
                                    min_samples_leaf=1)
gb.fit(X, y)
print(gb.predict([[1, 0]]))
print(gb.predict([[0, 1]]))
```

gives:

```
[0]
[0]
```

And `binary_crossentropy` works fine. `categorical_crossentropy` should either generalize or raise an error on binary classification.

Ping @NicolasHug @ogrisel

Input: a GitHub repo and an issue

Output: a file diff to resolve the issue

Evaluation: unit tests from pull request

Test Patch

sklearn/ensemble/_hist_gradient_boosting/tests/test_gradient_boosting.py

```
418      assert stump_clf.fit(X, y_isnan).score(X, y_isnan) == 1
419
420 + def test_crossentropy_binary_problem():
421 +     # categorical_crossentropy should only be used if there
422 +     # are more than two classes present. PR #14869
423 +     X = [[1], [0]]
424 +     y = [0, 1]
425 +     gbrt = HistGradientBoostingClassifier(loss='categorical_crossentropy')
426 +     with pytest.raises(ValueError, match="'crossentropy' not suitable"):
427 +         gbrt.fit(X, y)
428
429   @pytest.mark.parametrize("scoring", [None, 'loss'])
```

ChemCrow: An Agent That Discovered a Novel Chromophore

a

```
from chemcrow.agents import ChemTools, ChemCrow

chemtools = ChemTools()

# Initialize ChemCrow object with toolset and LLM
crow = ChemCrow(
    chemtools.all_tools,
    model="gpt-4",
    temp=0.1,
)

# Task definition
task = (
  "Find and synthesize a thiourea organocatalyst "
  "which accelerates a Diels-Alder reaction."
)

# Execute ChemCrow
crow.run(task)
```

b **Task:** Find and synthesize a thiourea organocatalyst which accelerates a Diels-Alder reaction.

First, I need to find a thiourea organocatalyst that can accelerate the Diels-Alder reaction. I will perform a web search to find a suitable catalyst.
Web Search tool: Schreiner's thiourea catalyst

Now, I will obtain the SMILES. **Name2Smiles tool:**
FC(F)(F)c1cc(NC(=S)Nc2cc(C(F)(F)F)cc(C(F)(F)F)c2)cc(C(F)(F)F)c1

I will plan a synthesis for Schreiner's thiourea catalyst.
SynthesisPlanner tool: Detailed synthesis plan

I will execute the synthesis.
SynthesisExecuter tool: Successful synthesis.

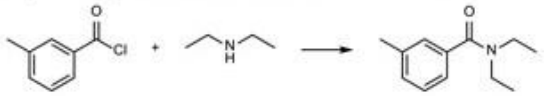
Connection with physical world →

c **RoboRXN synthesis platform**



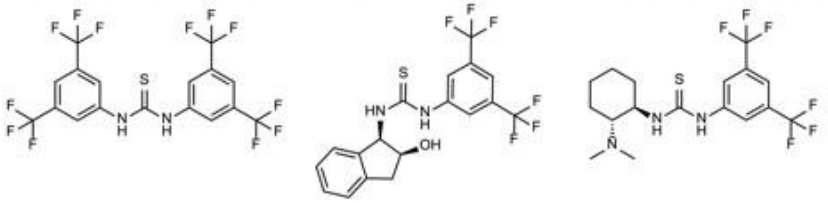
d **Chemcrow workflows with experimental validation**

Insect repellent (plan and execute) — *DEET*

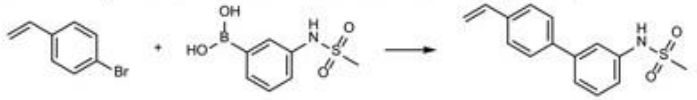


Thiourea organocatalysts (plan and execute)

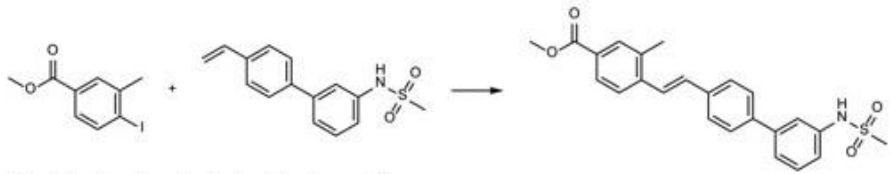
Schreiner's catalyst *Ricci's catalyst* *Takemoto's catalyst*



Novel chromophore (clean data, train model, and predict)



Synthesis step 1: Bromo Suzuki coupling



Synthesis step 2: Iodo Heck reaction

[Bran et al., 2023]

7 Agent Societies and Economic World Models

When many agents interact, we can simulate markets, hospitals and cities — a new kind of scaling.

Background: Toward a Human–AI Society

- From a **human-centered society** to a new form of **intelligent society characterized by human-AI collaboration.**
- The **actions** of intelligent agents extend beyond physical operations (as in embodied AI) to encompass **social and economic activities**.
- **Interaction** between agents and their **social environment** is a necessary condition for the **self-evolution of intelligence**.
- The environment is neither static nor solely human-centered, but rather a human–machine–integrated social ecosystem.

A New Scaling Law toward AGI?

- Old scaling law saturated:like pre-training & test-time scaling
- Scale number of agents?



SocioVerse: A World Model for Social Simulation Powered by LLM Agents and A Pool of 10 Million Real-World Users

Xinnong Zhang[1‡], Jiayu Lin[1‡], Xinyi Mou[1‡], Shiyue Yang[1], Xiawei Liu[1], Libo Sun[1], Hanjia Lyu[3], Yihang Yang[1], Weihong Qi[4], Yue Chen[1], Guanying Li[1], Ling Yan[5], Yao Hu[5], Siming Chen[1], Yu Wang[1], Xuanjing Huang[1], Jiebo Luo[3], Shiping Tang[1], Libo Wu[1,2], Baohua Zhou[1], Zhongyu Wei[1,2]
[1]Fudan University, [2]Shanghai Innovation Insititute,
[3]University of Rochester, [4]Indiana University, [5]Xiaohongshu Inc.
zywei@fudan.edu.cn
SocioVerse: https://github.com/FudanDISC/SocioVerse





Shanghai Innovation Institute: 10M population pool with high-precision, systematic, and generalizable large-scale social simulations.

Zhongguancun Academy: 1B scale simulation

- New Scaling law: scaling interactions with social environments

Economic World Models

- Individual simulation
  - micro: simulated students/patients/citizens/retail investors
  - Macro: with a group of people
- Simulation for social interaction
  - **Social** networks, **Information** sharing, **Money** flow, Import and export of **goods**
- **Physical Simulation**
  - **e.g. Minecraft.**
- **Simulation to operating** mechanism
  - **Stock Systems**
  - **Exchange** rate system

Example: Individual Simulation



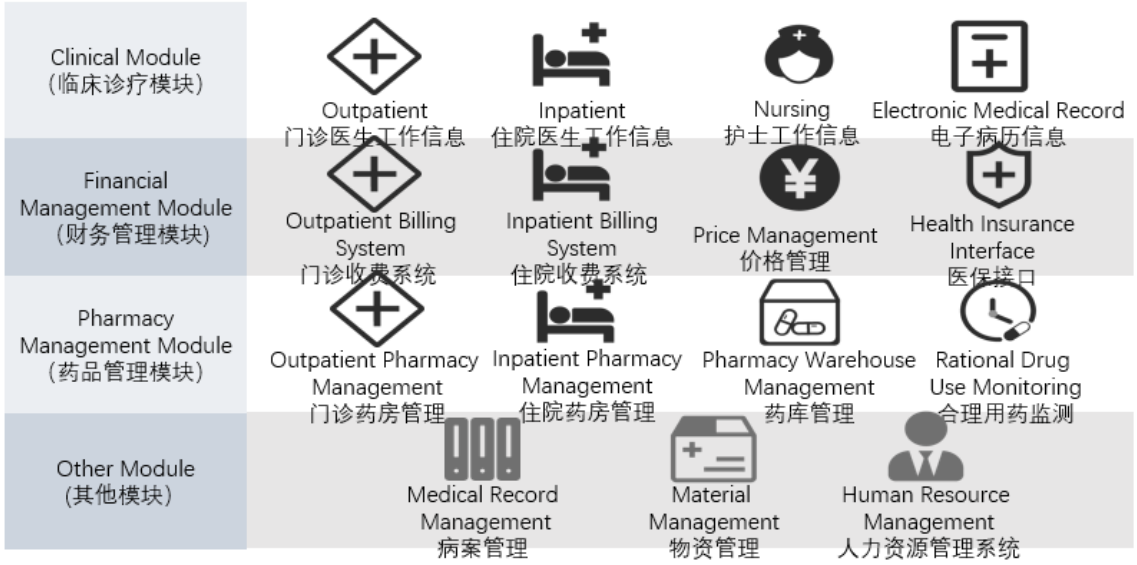


Example: Social Interaction

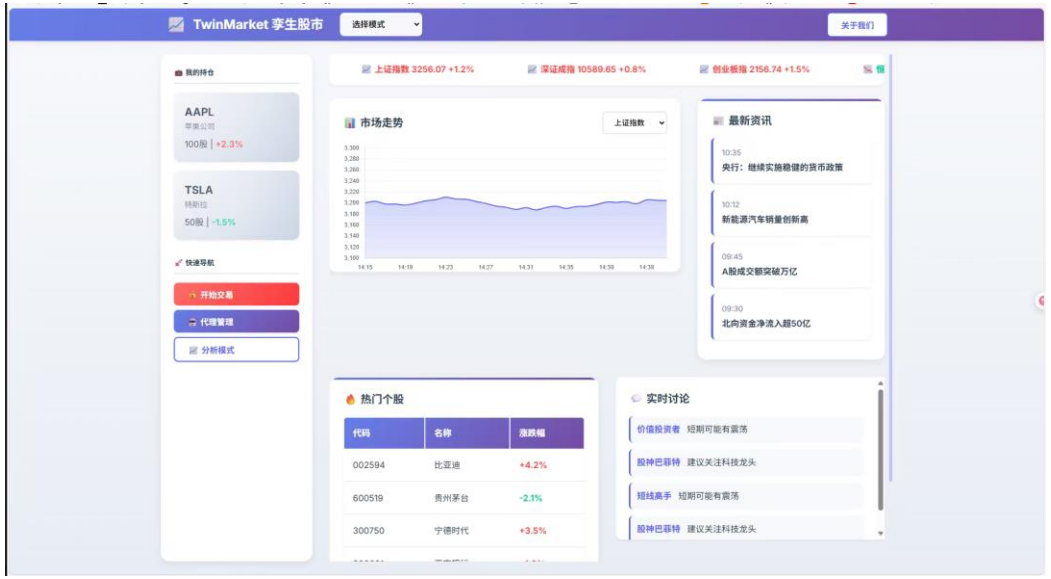


Providing new possibilities for **smart cities**, **spatial intelligence**, **embodied intelligence**, **5G communications**, and the **low-altitude economy**.

Example: Operating Mechanisms



Twin Hospital



Twin Market

A Virtual University

The future essence of education lies in **teaching people how to interact—with both humans and intelligent agents.**

- **Simulation and Interaction of Individuals:**
  - AI tutors provide real-time and assistive education.
  - AI students train teachers (just as AI patients train doctors).
  - Real-time, full-duplex voice dialogue and VR/AR technologies enable fully immersive learning experiences.
- **Simulation of Environments :**
  - Simulation of chemical reactions.
  - Simulation of circuit diagrams.
  - Interactive theorem proving (via Lean verifiers).
  - Simulation of drug–protein interactions.
  - Virtual computers.
  - Microscopic world simulators.

A Sandbox for Public Policy

- Example to concretizing policies regarding Shenzhen–Hong Kong cooperation



The simulation aims to demonstrate **the potential and risks** of Shenzhen–Hong Kong cooperation, provide decision-makers with more intuitive and quantitative insights, promote institutional innovation, and accelerate the development of the Northern Metropolis in Hong Kong.

- Personnel mobility and border management
- Capital flow and financial regulation
- Information exchange
- Goods circulation
- Policy and institutional innovation

Join Us: Building the Twin Society

Create the Twin Society — where agents learn, evolve, and thrive

Takeaways

Prompts	Prompting tricks became agent components: planning, tools, verification, reflection.
Agents	An agent is an LLM in a think → act → observe loop, with planning, tools, memory and action.
Harness	Agent = Model + Harness. Harness choice alone moves benchmark scores by 10–20 points.
Practice	Small high-signal context, skills and MCP for tools, A2A between agents, durable and fault-tolerant execution, pass^k evals.
Philosophy	Build the thinnest harness that keeps the agent safe, stateful and measurable — and delete the rest as models improve.

Acknowledgements and Further Reading

Agents and courses

- Stanford CS329Z: Engineering AI Agents (Fall 2026), cs329z.stanford.edu
- Princeton COS 597G: Understanding Large Language Models
- Lilian Weng. LLM Powered Autonomous Agents (2023)
- Berkeley LLM Agents course: llmagents-learning.org
- LLM-Agent-Survey / LLM-Agent-Paper-List (GitHub)
- Cemri et al. Why Do Multi-Agent LLM Systems Fail? (2025)
- Kim et al. Towards a Science of Scaling Agent Systems (2025)
- R. Sutton. The Bitter Lesson (2019)

Harness engineering

- OpenAI. Harness Engineering; Unrolling the Codex Agent Loop; Codex App Server (2026)
- Anthropic. Building Effective Agents (2024); Effective Context Engineering (2025)
- Anthropic. Effective Harnesses for Long-Running Agents (2025); Demystifying Evals (2026)
- B. Böckeler. Harness Engineering (martinfowler.com, 2026)
- V. Trivedy. The Anatomy of an Agent Harness (LangChain, 2026)
- Model Context Protocol; A2A protocol; agentskills.io; agents.md
- OpenClaw documentation (docs.openclaw.ai)
- Temporal: durable execution for AI agents

Thank You!

Questions?