CS-461

Foundation Models and Generative AI

Foundation Models and Agentic Systems and Beyond

Charlotte Bunne, Fall Semester 2025/26

Announcements

About the Exam

Friday 16.01.2026 from 15:15 to 18:15 in CE 1 515 and PO 01.

Closed book exam. One A4 crib sheet (both sides).

Details on room assignment will follow shortly!

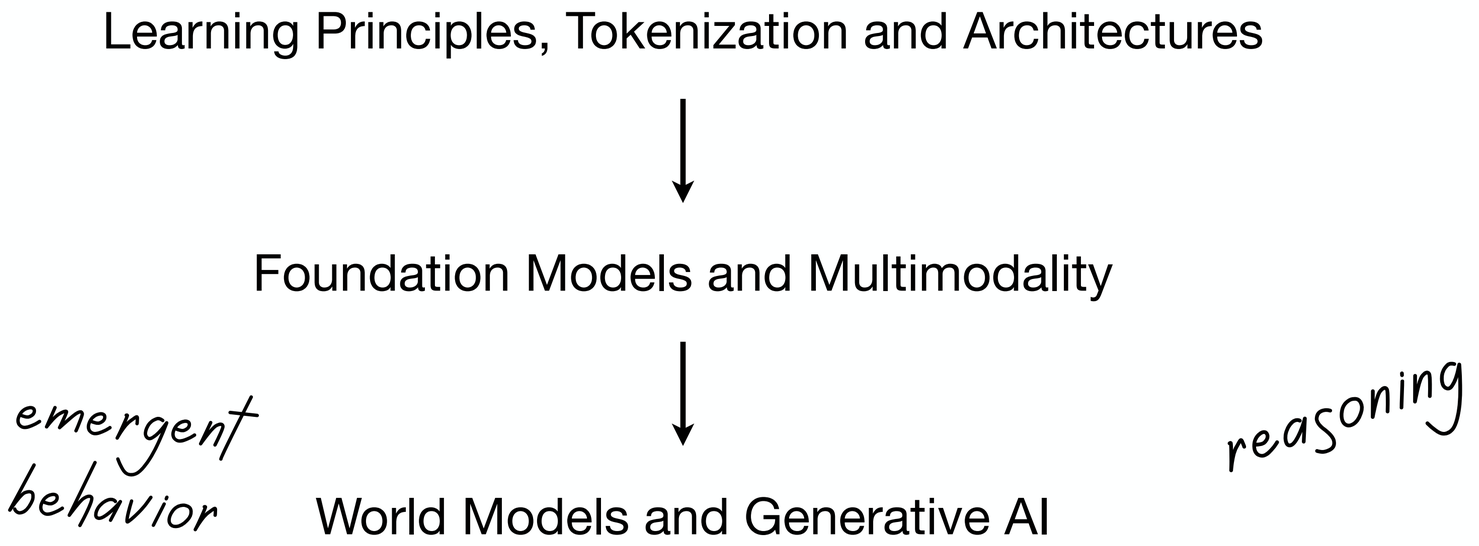
Recap Session with All TAs

… on January 8, 10 am to 12 pm in **AAC 231**.

… all TAs will be present, so come with your questions.

… but before we all go into Christmas break,
let's enjoy this last lecture!

Was Happened So Far?



→ **We have built powerful models; now we ask what happens when they act.**

From Language Models to AI Agents

> Large language models can already analyse prose, write code, and argue a point, yet they live in a closed book: they read tokens, write tokens, and forget the scene once the page turns. An *agent*, by contrast, survives in the wild.

LLMs are powerful but “closed book” systems: read tokens → write tokens

→ *Limited persistence, limited direct interaction with the world.*

Agents are models placed in an environment loop:observe → decide → act → observe → etc.

→ Turns static generation into goal-directed decision-making enabling sustained goal pursuit.

Liu et al., (2025)

What are Agents?

agent

/ˈeɪdʒ(ə)nt/

noun

noun: **agent**; plural noun: **agents**

1. a person who acts on behalf of another person or group.

   "in the event of illness, a durable power of attorney enabled her nephew to act as her agent"

   - a person who works secretly to obtain information for a government or other official body.

     "a trained intelligence agent"

   - a person or company that provides a particular service, typically one that involves organizing transactions between two other parties.

     "speak to your letting agent about refurbishing the property"

From Language Models to AI Agents

Liu et al., (2025)

What *faculties* must any truly autonomous agent possess,
no matter how the modules are wired?

A truly autonomous agent needs **capabilities**, not a particular architecture. It must be able to:

1. **Perceive:** Acquire observations from its environment (text, sensors, APIs, images) and extract task-relevant state.
2. **Maintain state and memory**: Carry forward information across time: working memory (current context), and longer-term memory (facts, past events, commitments).
3. **Represent goals and constraints**: Keep track of what it is trying to achieve (objectives), what it must not do (constraints), and how to trade off competing objectives.
4. **Reason and plan**: Decompose goals, choose actions over a horizon, anticipate consequences (even approximately), and revise plans when reality diverges.
5. **Act and execute interventions**: Have an action interface that can change the environment (tool calls, UI actions, robot actuators), not just produce text.
6. **Learn and adapt**: Improve behavior from experience/feedback: update its policy, its world model, and/or its memory (even if weight updates are offline).
7. **Self-monitor and stay calibrated**: Detect errors, uncertainty, and dead-ends; decide when to verify, ask clarifying questions, abstain, or hand off to humans.

From Language Models to AI Agents

Liu et al., (2025)

What *faculties* must any truly autonomous agent possess, no matter how the modules are wired?

Perception	**Reasoning**	**World Understanding**
Learning to perceive and integrate multimodal information	Learning to perform structured and self-improving reasoning	Learning to build and refine memory, reward, and world models

What are Agents?

AGENTS

An agent is an **autonomous, adaptive intelligent system** designed to actively perceive diverse signals from its *environment*, continuously learn from experience to refine and update structured internal states (such as *memory*, *world models*, *goals*, emotional states, and *reward* signals), and reason about purposeful actions (both external and internal) to autonomously navigate toward complex, long-term objectives.

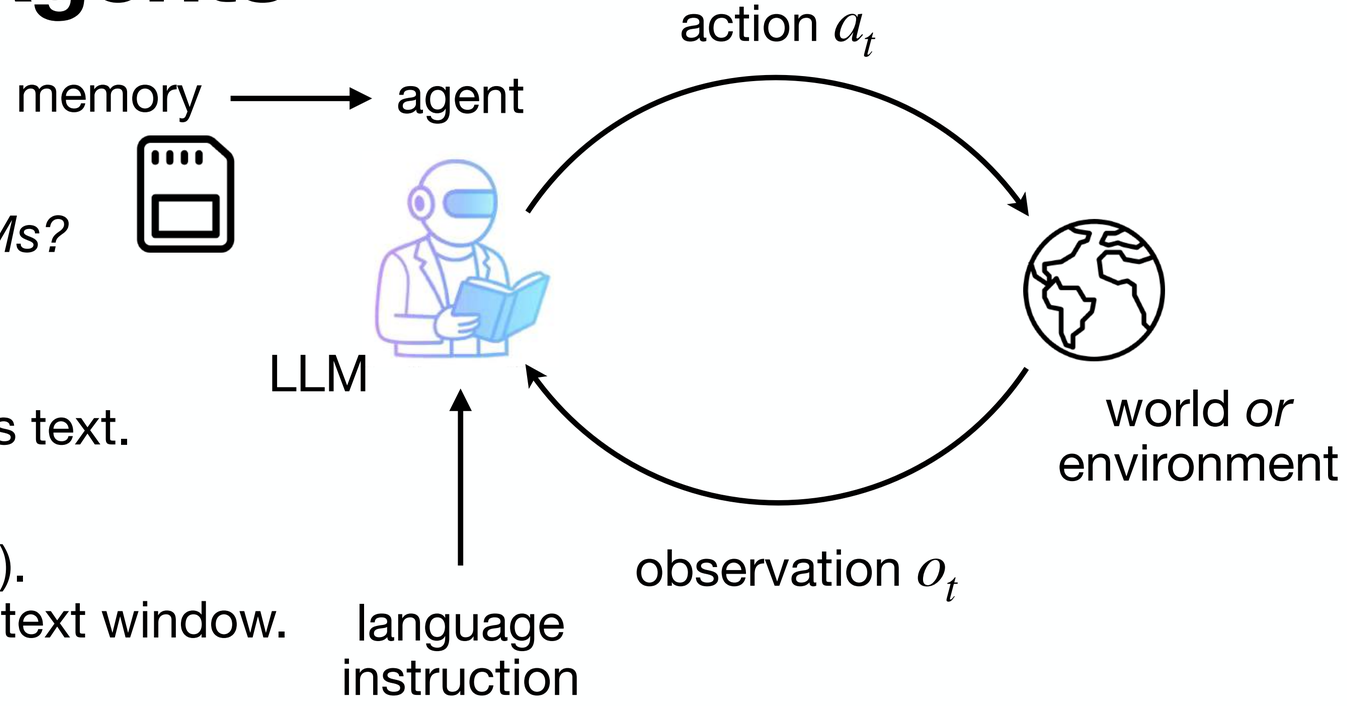
Minimal example in *RL terminology*: a system that, over time, maps observations o_t and internal state m_t to actions a_t to optimize a goal / reward.

Environment provides: observations o_t, transition dynamics $s_{t+1} = f\left(s_t, a_t\right)$, and optionally a reward r_t

Core loop: observe o_t → reason and plan → act a_t → log + update memory → repeat until stop.

Instruction-Following Agents

***Modern agents are* steered by large language models** *= augmented LLMs?*



Core idea:
Use LLM in a loop and specify action space as text.

Memory: Persist state across steps
(history, intermediate results, user preferences).
→ enables long-horizon tasks beyond the context window.

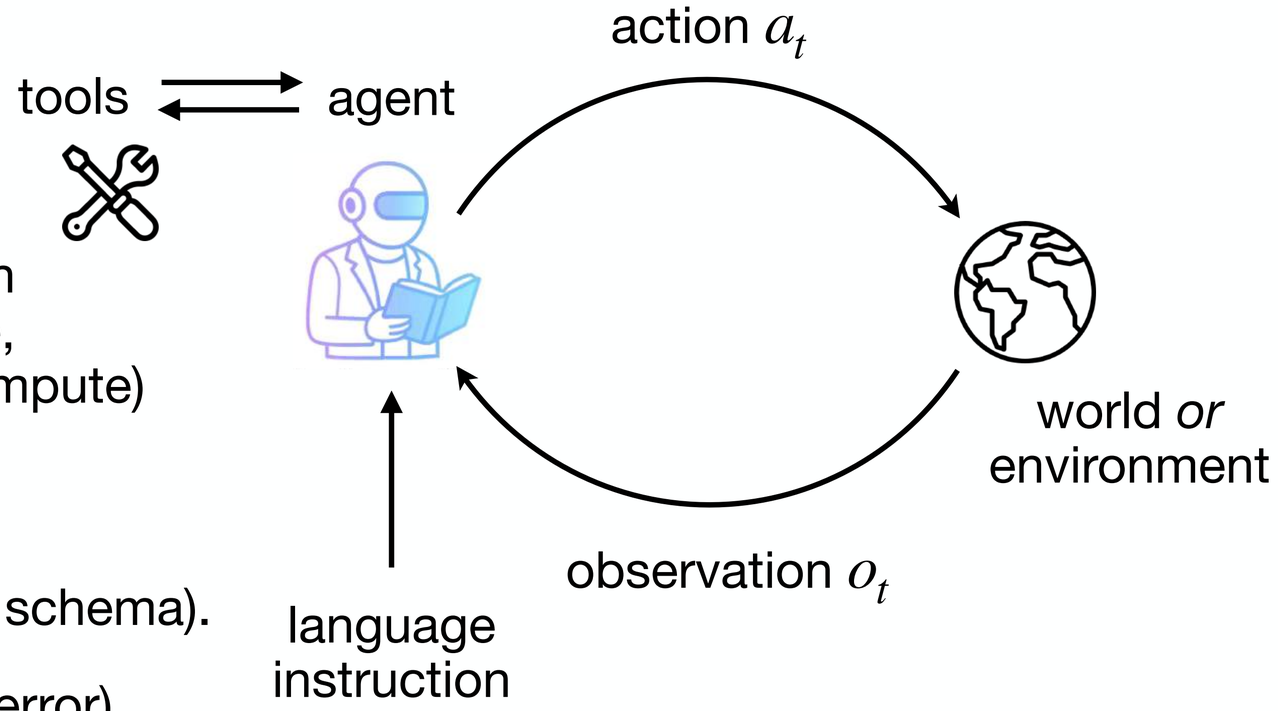
Why LLMs work as controllers?
Strong instruction following, flexible decomposition,
and language as a universal interface to tools.

Key challenge:
Long-horizon reliability: Errors compound (hallucinated steps, getting stuck)
→ need tool schemas and validation + verification + clear stop or abstain criteria.

Tools for Agents

What are Tools?

A tool is an external function or API the agent can invoke via a **structured call** to produce a reliable, verifiable effect; either information (retrieve or compute) or an environment change (actuate).



Action a_t: tool invocation (name and arguments, schema).

Observation o_t: tool output (structured result or error).

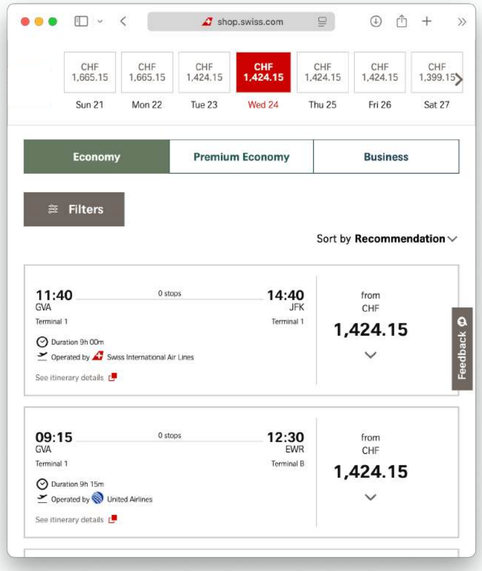
Tools provide **capabilities the LLM cannot do reliably**:
Exact computation, domain-specific models, real-world actions, etc.

Problem! **Tool overload:** Performance degrades less from the number of tools than from tool overlap; systems might handle >15 distinct tools but may fail with <10 similar ones.

Modern Agentic Systems

e.g., **Web Agents**:

- Instruction: “Book a flight from Geneva to New York.”
- Actions: `Type … on …`, `Click on …`, `Choose … from dropdown …`, etc.
- Observations:

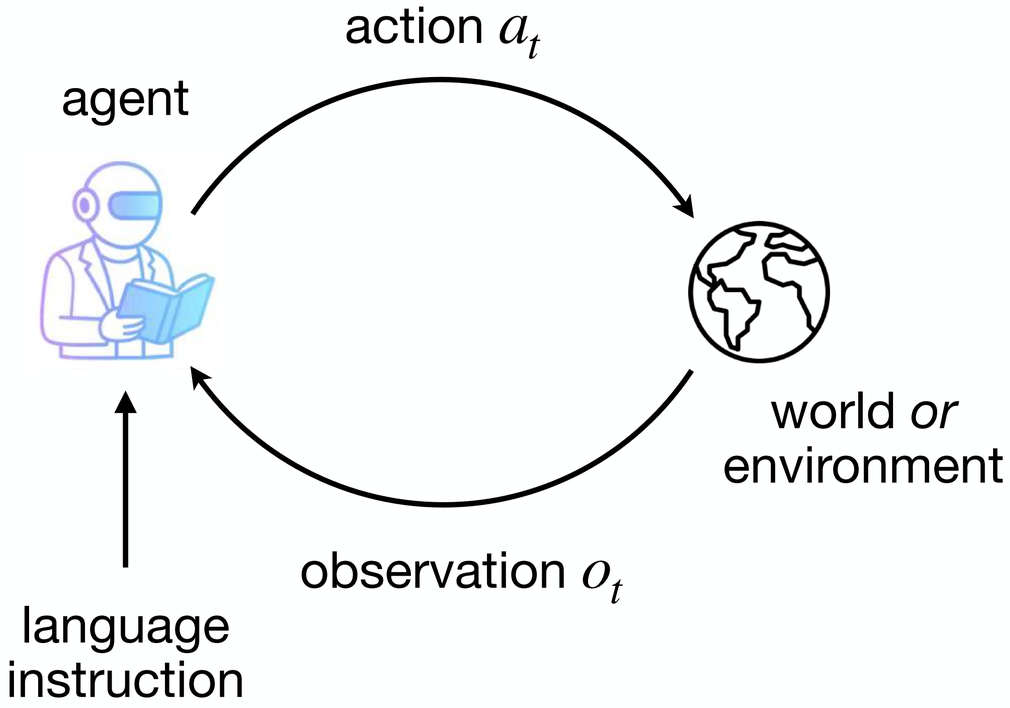


raw pixels
via a visual encoder.

```
<!doctype html>
<html style="font-size: 62.5%;" data-beasties-container>
 <head>
  <title>LHG booking</title>
  <link rel="shortcut icon" href="assets/icons/favicon.ico" type="image/x-icon">
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1"><base href="
  <style>body{margin:0}body > app > .loading-container{background:#fcfcfc;display:
  <link rel="stylesheet" href="assets/index.css" type="text/css" media="print" onl
  <noscript>
   <link rel="stylesheet" href="assets/index.css" type="text/css">
  </noscript>
  <link rel="stylesheet" href="assets/implementation.css" type="text/css" media="p
  <noscript>
   <link rel="stylesheet" href="assets/implementation.css" type="text/css">
  </noscript>
  <meta name="theme-color" content="#ffffff">
  <style>@font-face{font-family:LX-icons;src:url(LX-icons.fc0eec29f4b7127e.ttf) fo
  <link rel="stylesheet" href="styles.509866bbd8505f07.css" media="print" onload="
  <noscript>
   <link rel="stylesheet" href="styles.509866bbd8505f07.css">
  </noscript>
 </head>
 <body class="mat-typography" data-dynamicContentPath="/statics/applications/booki
  <app>
   <div class="loading-container">
    <img class="plane-gif" src="assets/img/common/animation-loader.gif">
   </div>
  </app>
  <noscript>
```

html

via LLM
with html
understanding.



The ReAct Framework

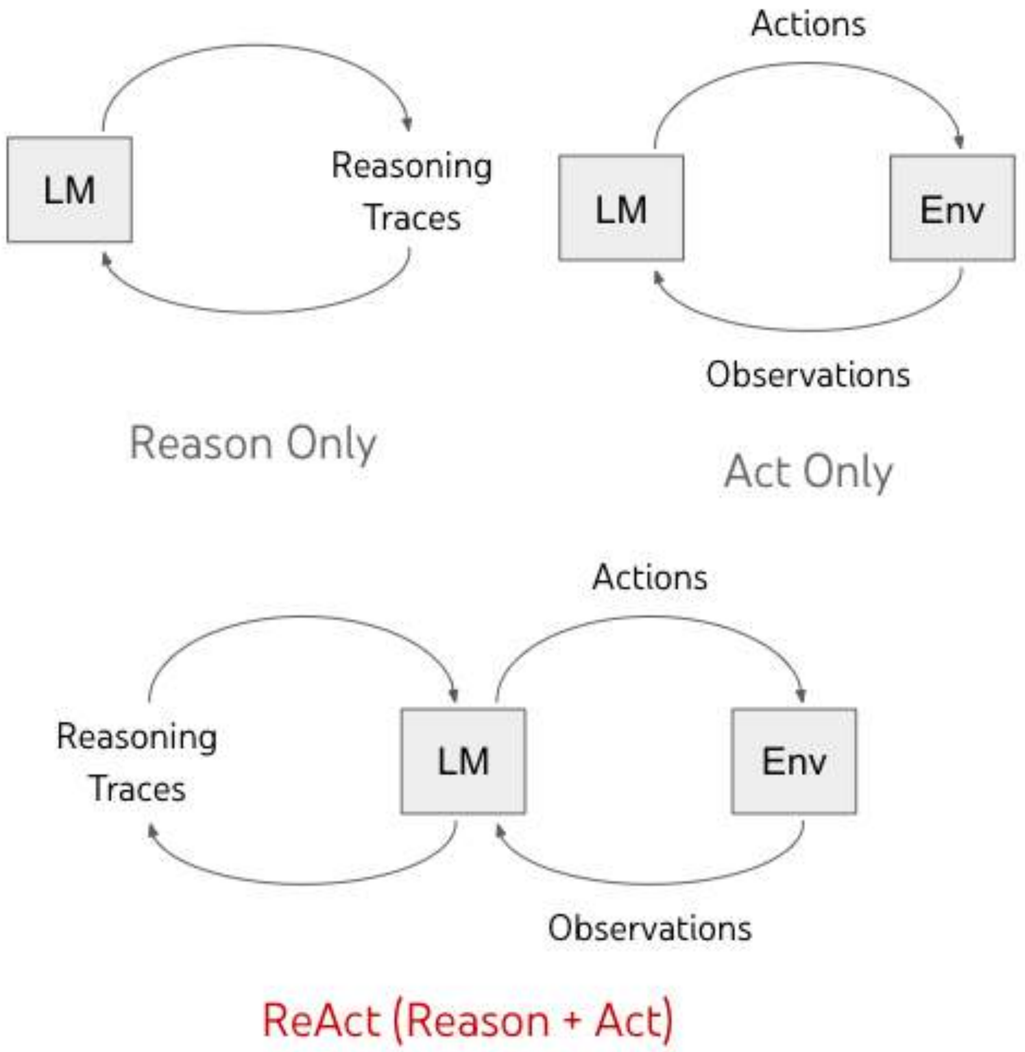
to synergize **reasoning** and **acting** in language models.

Idea:
Interleave Reasoning (a running “scratchpad”) with actions a_t (e.g., tool calls) and observations o_t (e.g., tool outputs).

Loop:
Thought → Act→ Observe → Thought … → Final Answer

Why it helps?
Reduces hallucination by grounding decisions in retrieved/computed evidence; supports multi-step planning without committing to a full plan upfront.

Yao et al., (2023)



→ ReAct paradigm systematically *outperforms* reasoning and acting only paradigms.

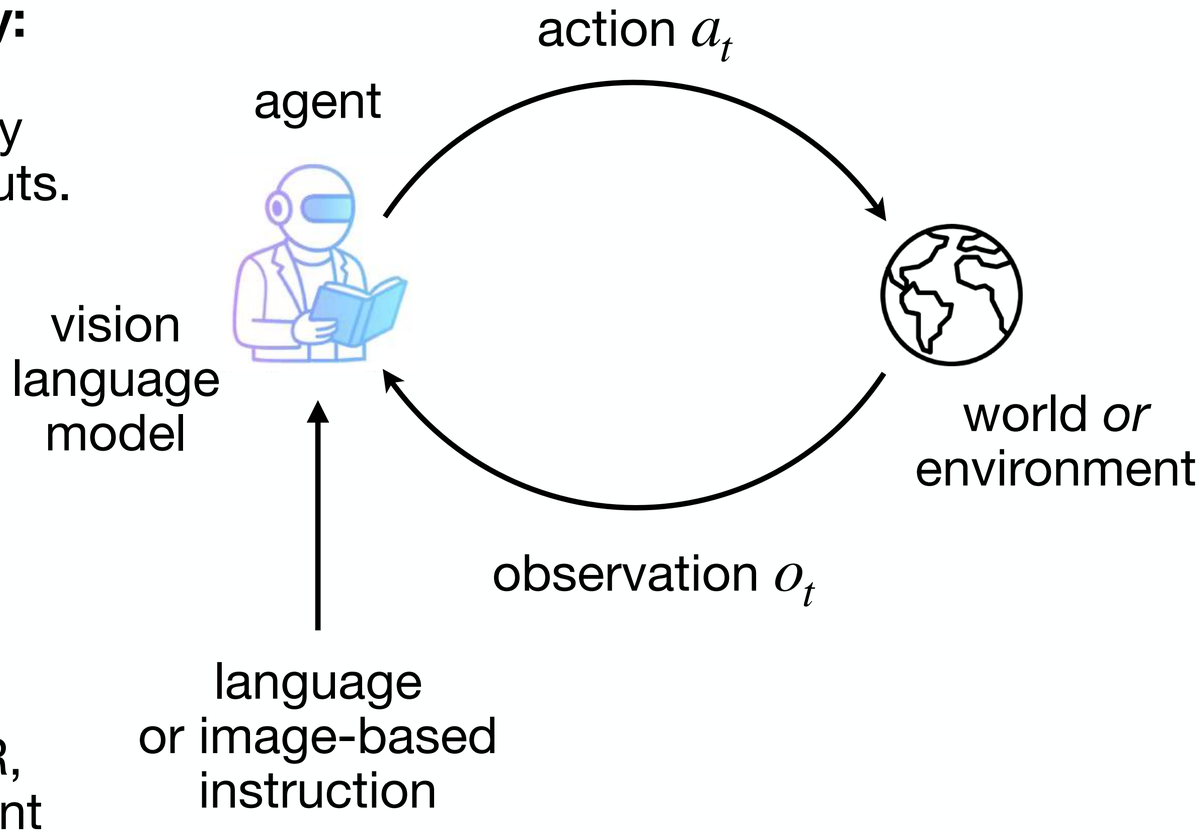
Multimodality: **Visual Language Models as AI Agents**

The central agent does not have to be text-only:

It can also be a vision-language model that directly reasons over images alongside text and tool outputs.

Vision: The central agent can be a multimodal foundation model that reasons over text plus domain signals (e.g., images, tables, time series, graphs, omics, structures).

In science, "observations" are often assays and measurements (e.g., microscopy, EHR, sequencing, proteomics), and the agent learns to integrate them into actionable hypotheses and decisions.



Operating System Layers for Agents

Tool libraries: Curated, typed interfaces (name, description, input schema, output schema). = syscalls

To make structured tool calls possible, every tool needs a machine-readable schemas.

→ Why do schemas matter? Reduce hallucinated tool calls; enable validation; improve reliability!

Important components:

- **Model Context Protocol (MCP):**
  Standardizes how agents interfaces or connects to tools and data. = device driver/ API standard
- **Agent Software Development Kits (SDK)**:
  Provide the orchestration loop, tool routing, retries, memory patterns, logging;
  so you do not rebuild infra each time.
  = runtime and scheduler

Agents Model Context Protocol (MCP)

A standard way for an agent to discover and call external tools and data.

Code Notebook 13

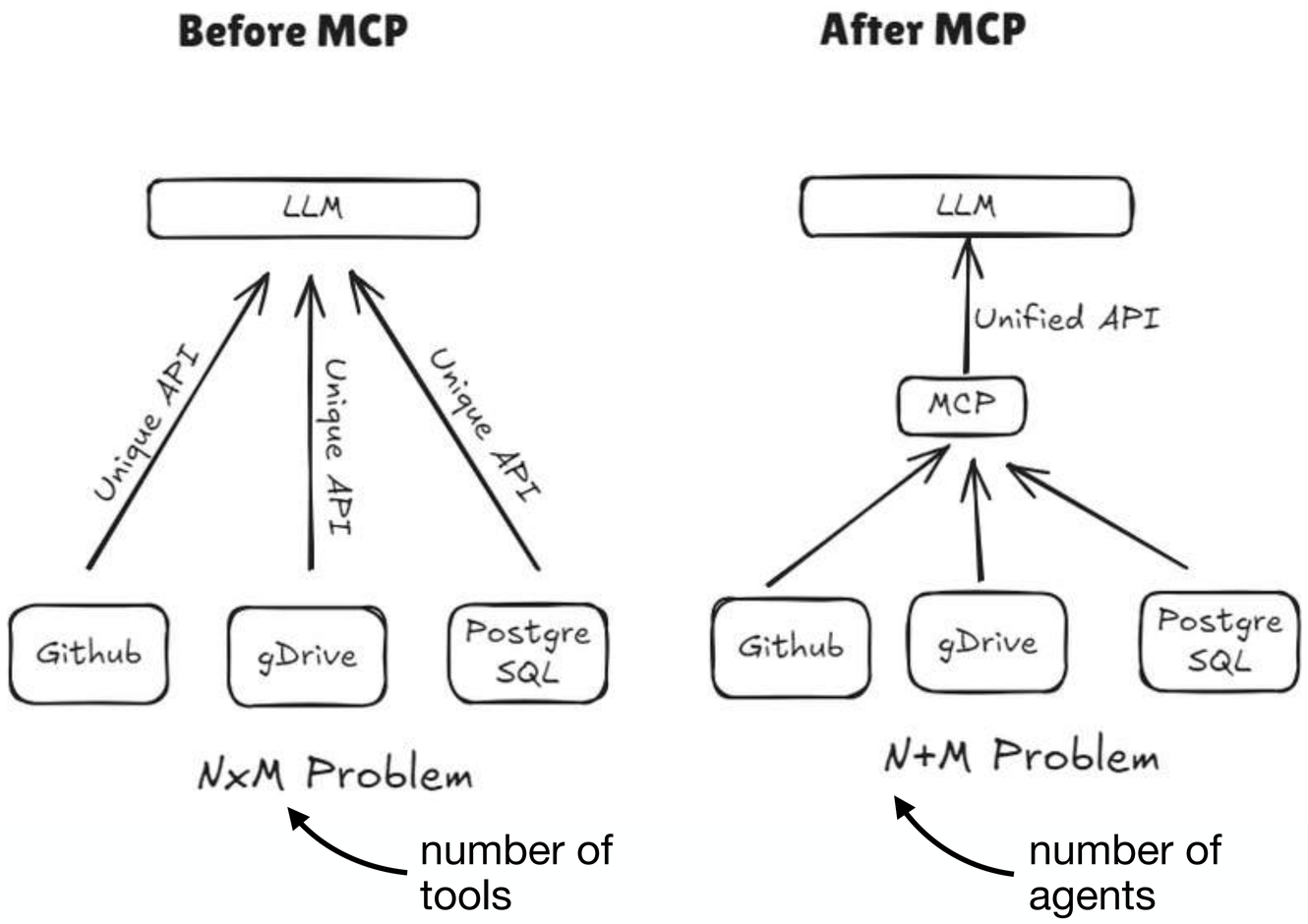
Problem: Every tool/API has a different interface → brittle, custom integrations.

Core idea: Tools expose machine-readable specs (name, description, input/output schema) over a standard protocol.

What MCP standardizes:

- **Tool discovery**: "What tools exist and what do they do?"
- **Invocation**: "Call tool X with arguments Y."
- **Results**: Return structured outputs or errors back into the agent context.

Why it matters? Swap tools or providers without rewriting agent logic.
Improves control via centralized permissions.



Agent Software Development Kits (SDK)

An SDK is a framework to build and run an agent:
It implements the agent loop (observe → think → act → observe), manages context/memory, and orchestrates tool routing and execution (incl. retries, timeouts, or logging.

- MCP standardizes the tool interface (discovery, schemas, invocation, and results).
  An SDK uses MCP (or other adapters) and adds the rest of the “plumbing” to ship a reliable agent.

Examples:

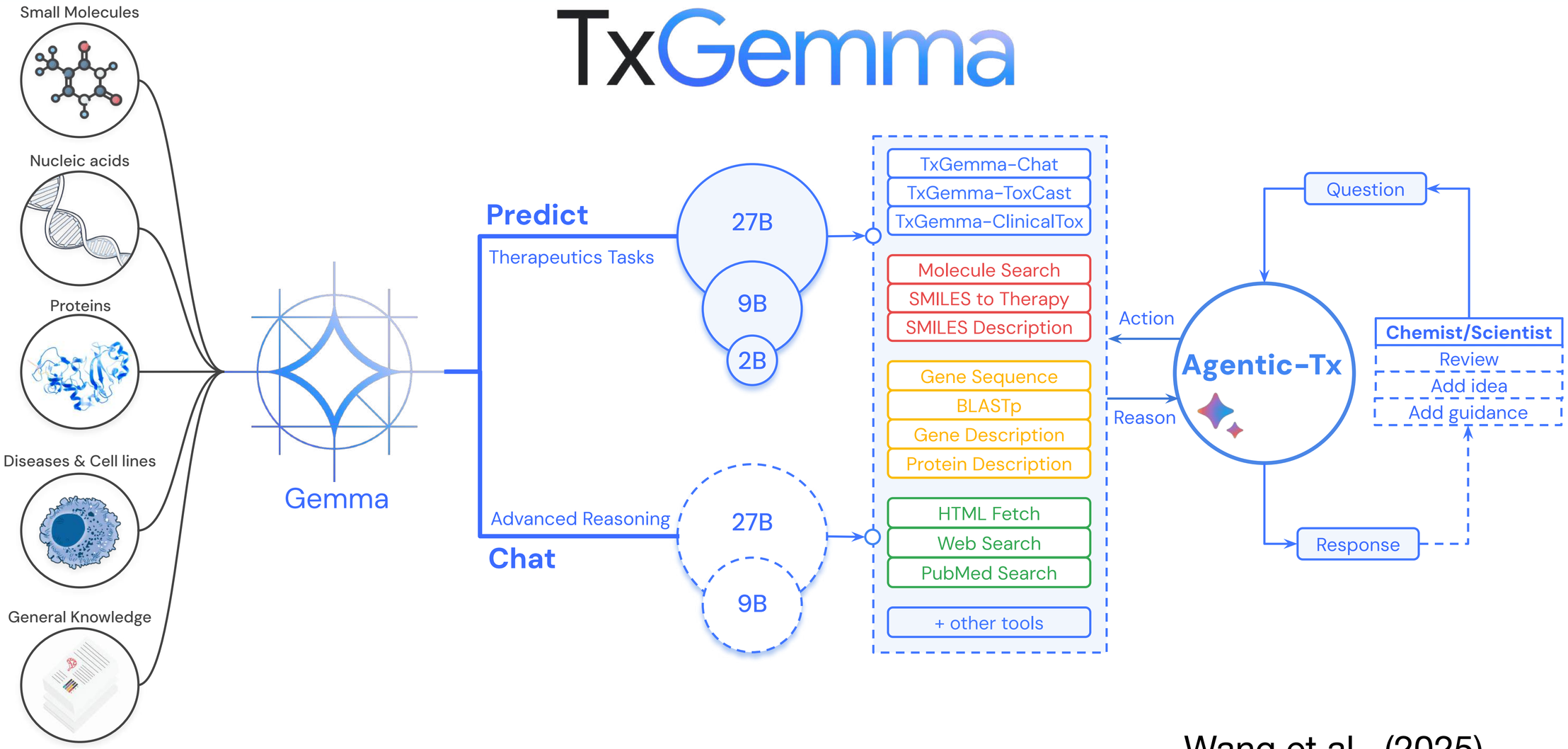
- OpenAI Agents SDK
- Anthropic Claude Agent SDK, etc.

```python
from agents import Agent, Runner

agent = Agent(name="Assistant", instructions="You are a helpful assistant")

result = Runner.run_sync(agent, "Write a haiku about recursion in programming.")
print(result.final_output)
```

Output:

```
Code within the code,
Functions calling themselves,
Infinite loop's dance.
```

Example: **Agents for Therapeutics**



Wang et al., (2025)

Example: **Agents for Therapeutics**

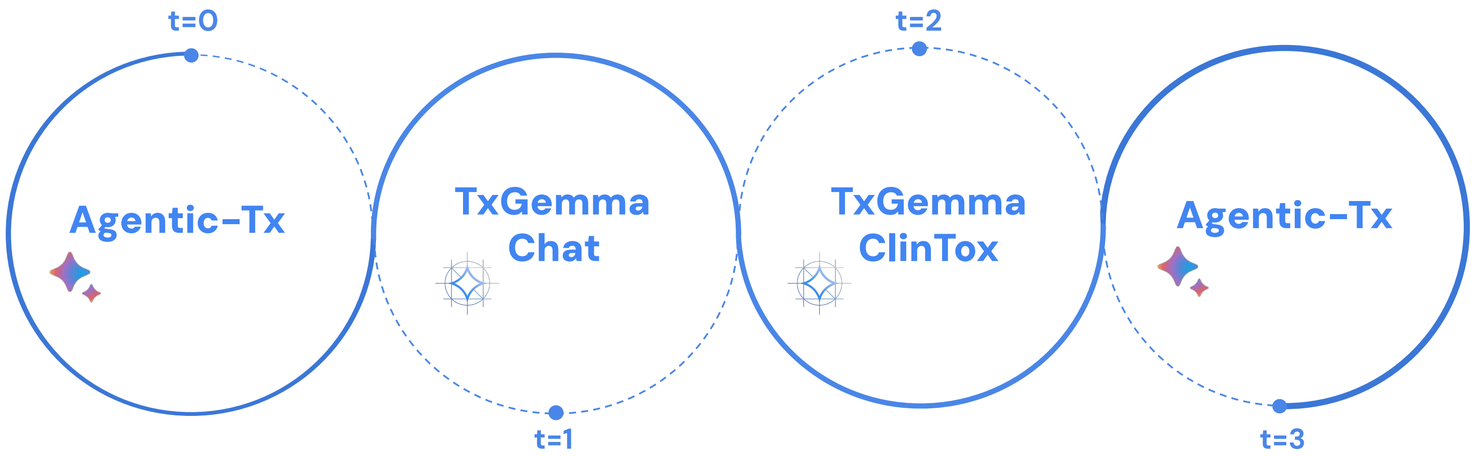
Prompt: Imagine an early virtual screening campaign setting. Which of the following two candidates would you prefer for further development?

<Molecule 1: SMILES>

<Molecule 2: SMILES>

Agent → TxGemma-ClinTox: Is the following toxic?
<Molecule 1: SMILES>
TxGemma ClinTox: Toxic

Agent → TxGemma-ClinTox: Is the following toxic?
<Molecule 2: SMILES>
TxGemma ClinTox: Non-toxic



Agent → TxGemma-Chat: Given two drug candidates, what factors would influence your decision to prioritize one over the other in an early virtual screening campaign?

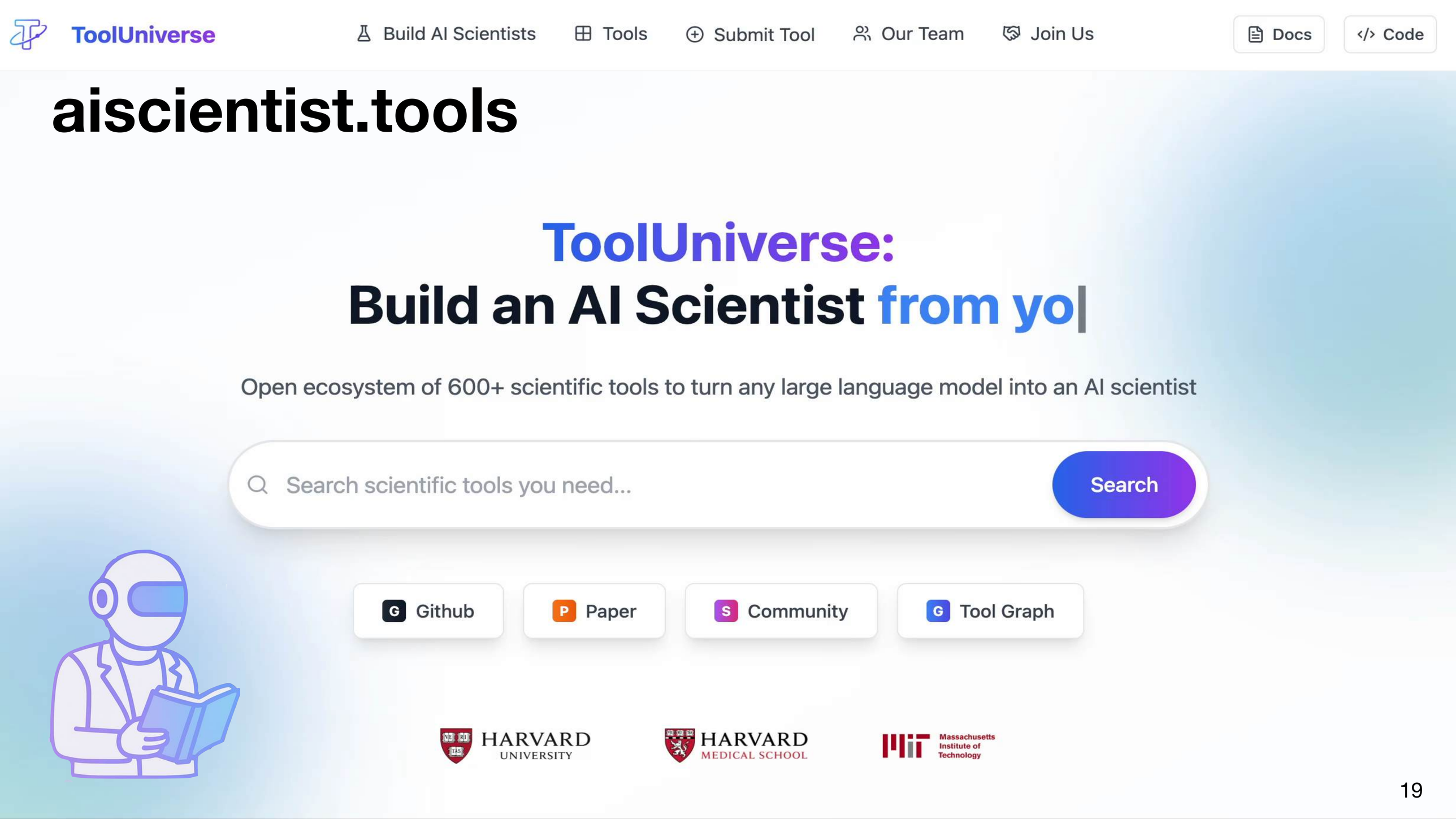
TxGemma-Chat: Investigate whether the drugs would pass through a clinical trial, based on properties such as toxicity.

Agent: Final answer: <Molecule 2: SMILES>

Natural Language Input → Reason + Action → Tool-use → Final Answer

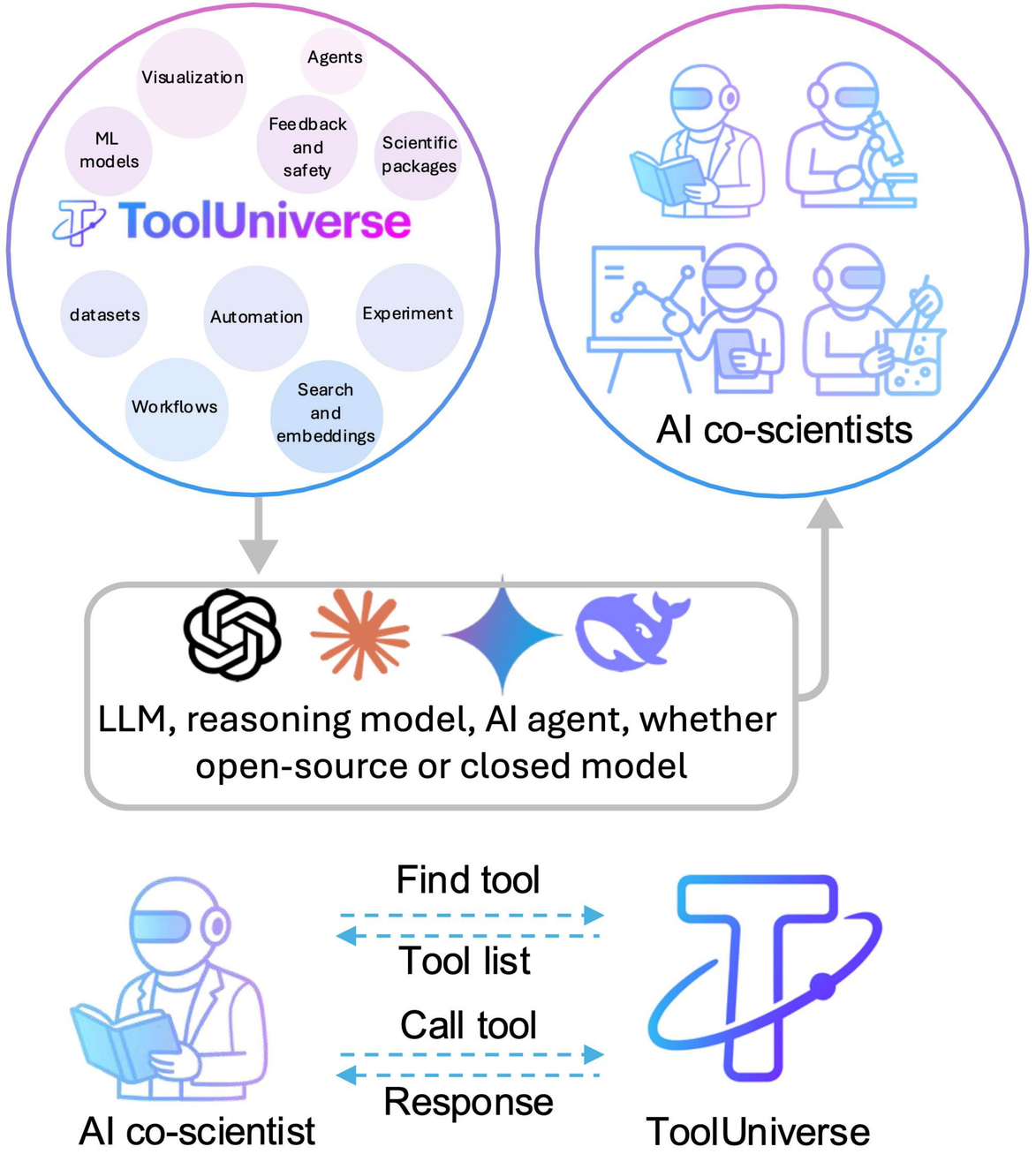
Wang et al., (2025)

aiscientist.tools

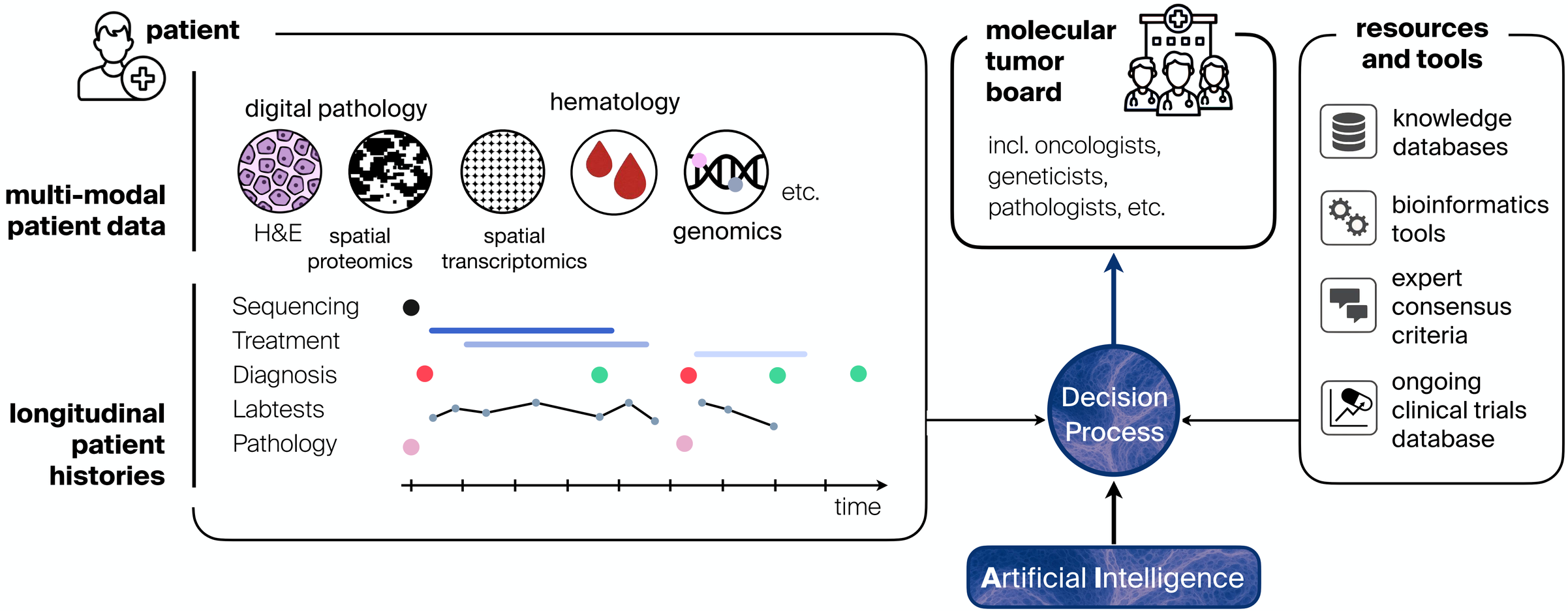


aiscientist.tools

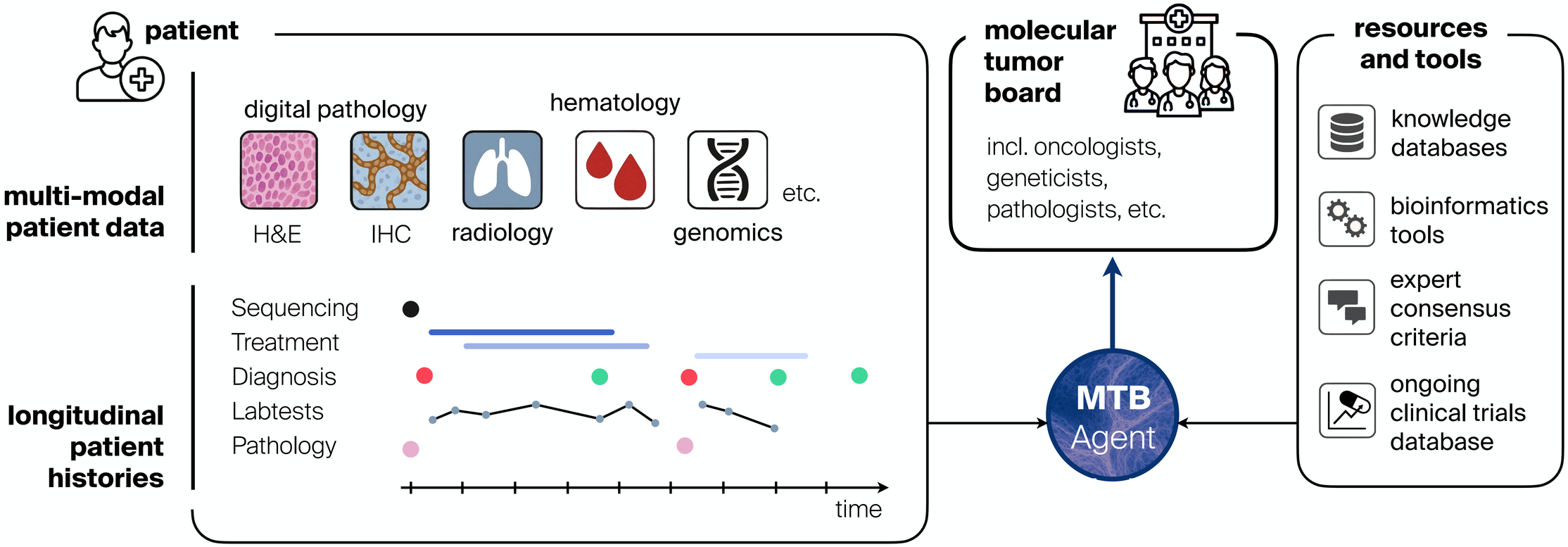
1. Select an LLM, whether open or closed.
2. ToolUniverse links LLM to data, models, tools.
3. AI cancer co-scientist is ready.



Example: **Agents for Molecular Tumor Boards**

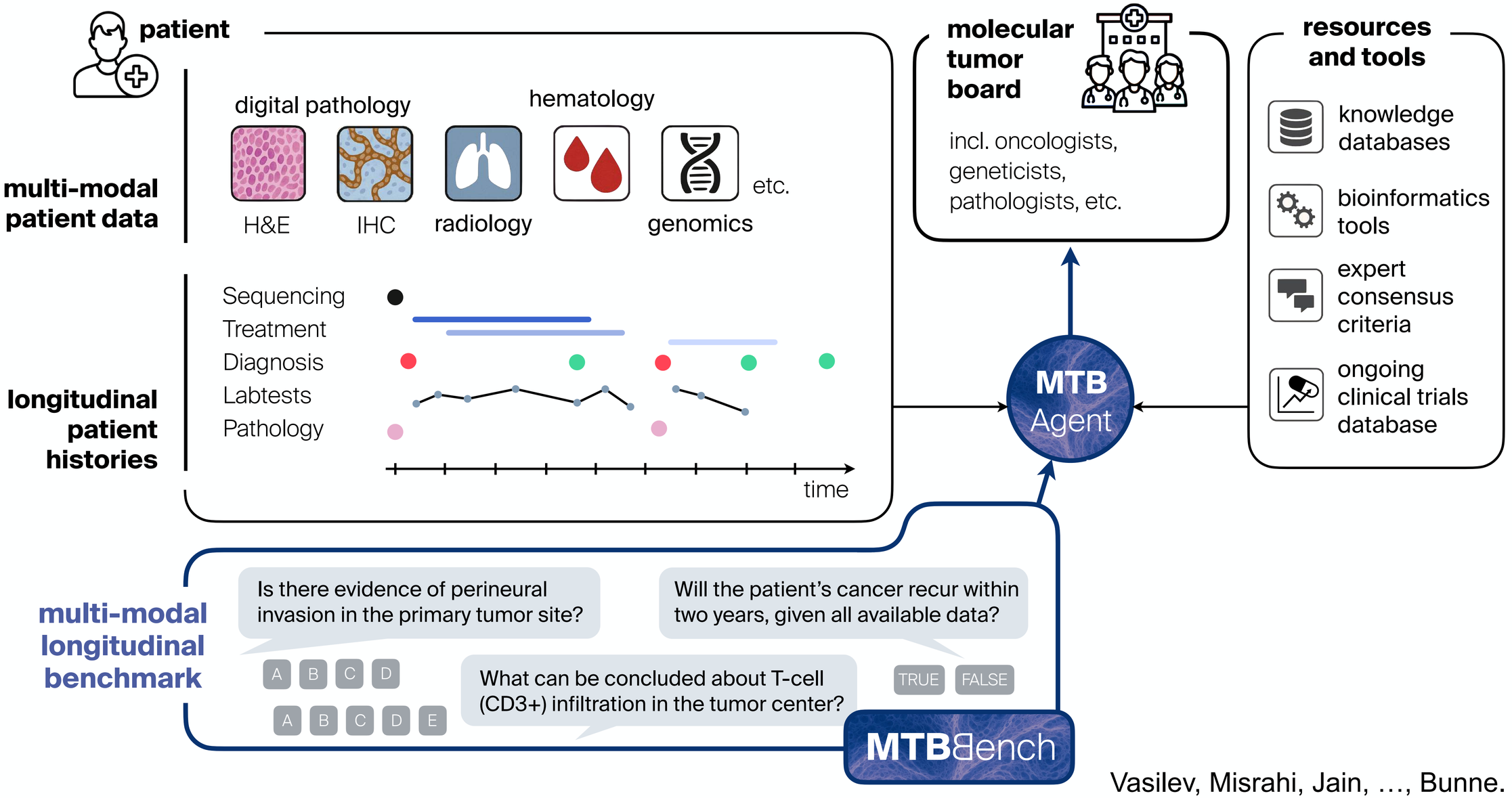


Toward Agents and FMs in Molecular Tumor Boards



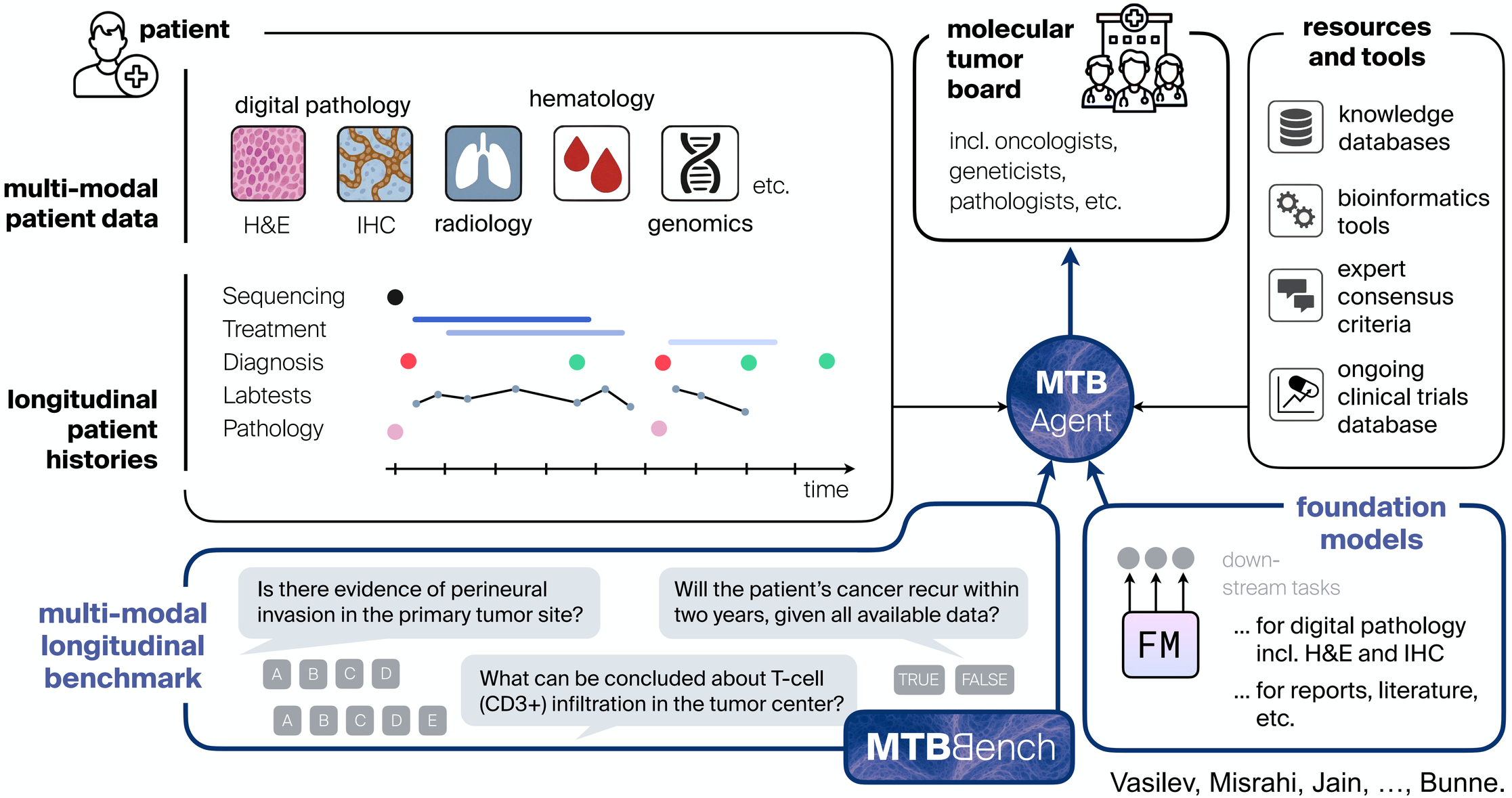
Vasilev, Misrahi, Jain, …, Bunne.
NeurIPS (2025)

Toward Agents and FMs in Molecular Tumor Boards



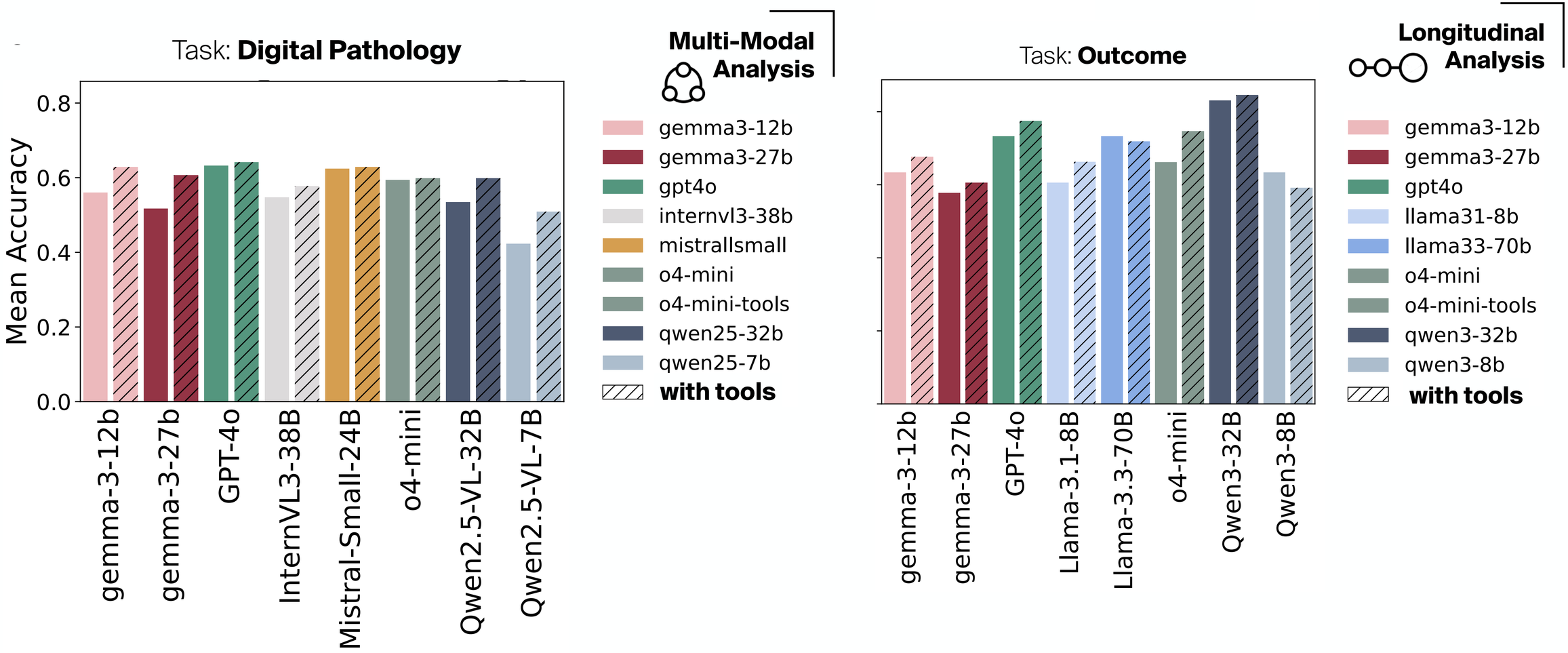
Vasilev, Misrahi, Jain, …, Bunne. **NeurIPS** (2025)

Toward Agents and FMs in Molecular Tumor Boards



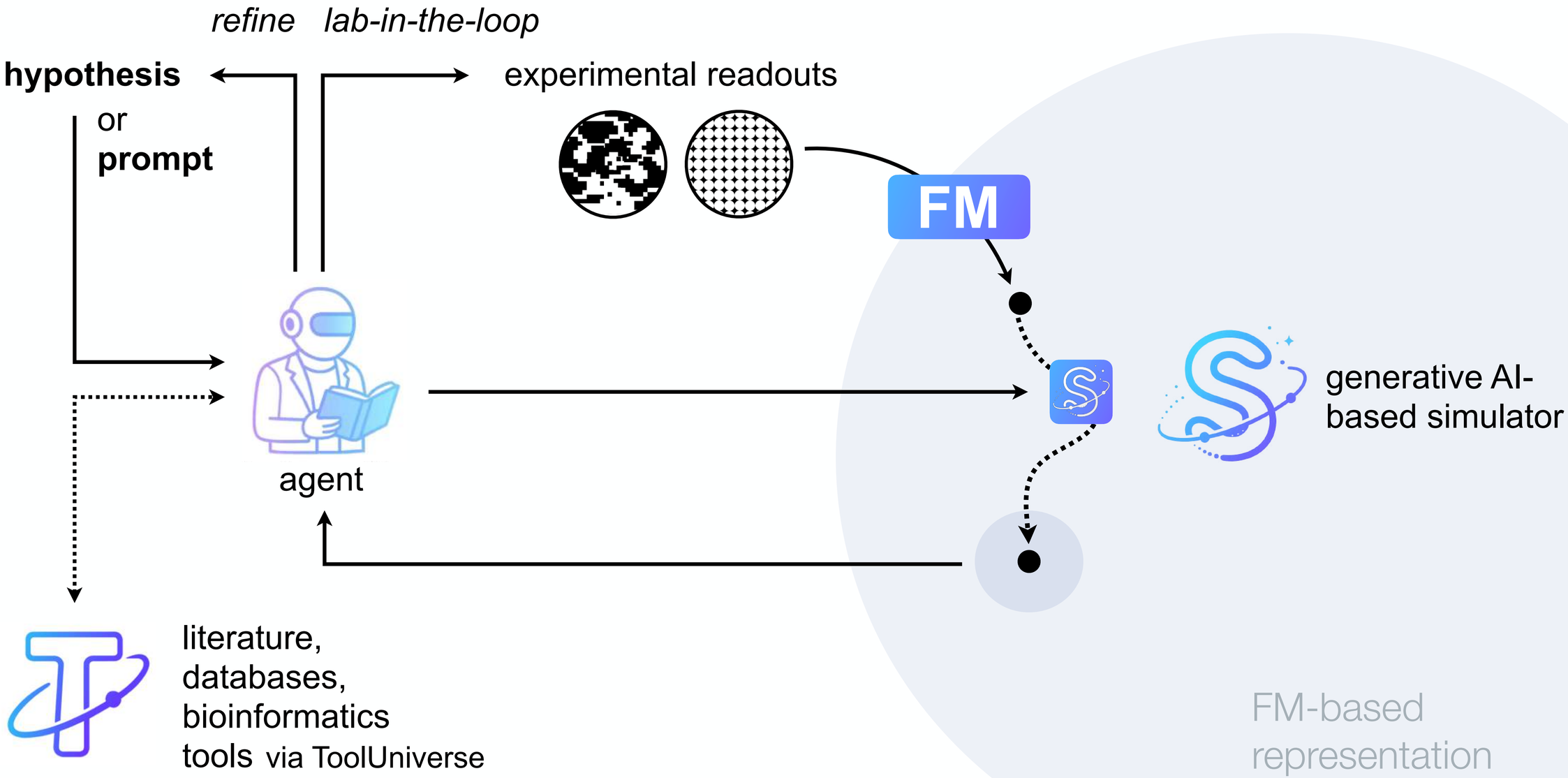
Vasilev, Misrahi, Jain, …, Bunne. **NeurIPS** (2025)

FM-Powered Agents for Molecular Tumor Boards



Vasilev, Misrahi, Jain, …, Bunne.
NeurIPS (2025)

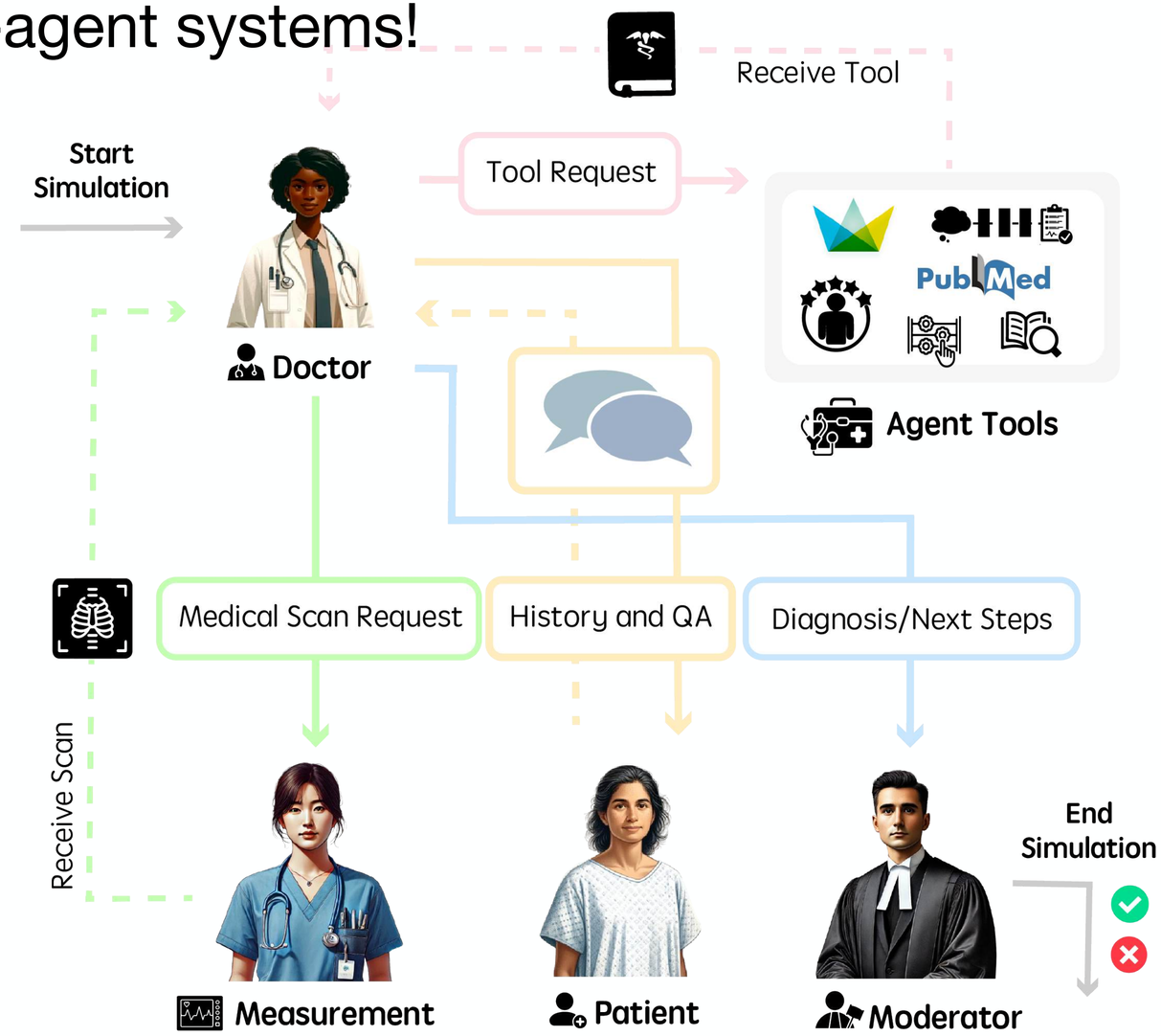
Accelerating Science through Agentic Scientific Reasoning



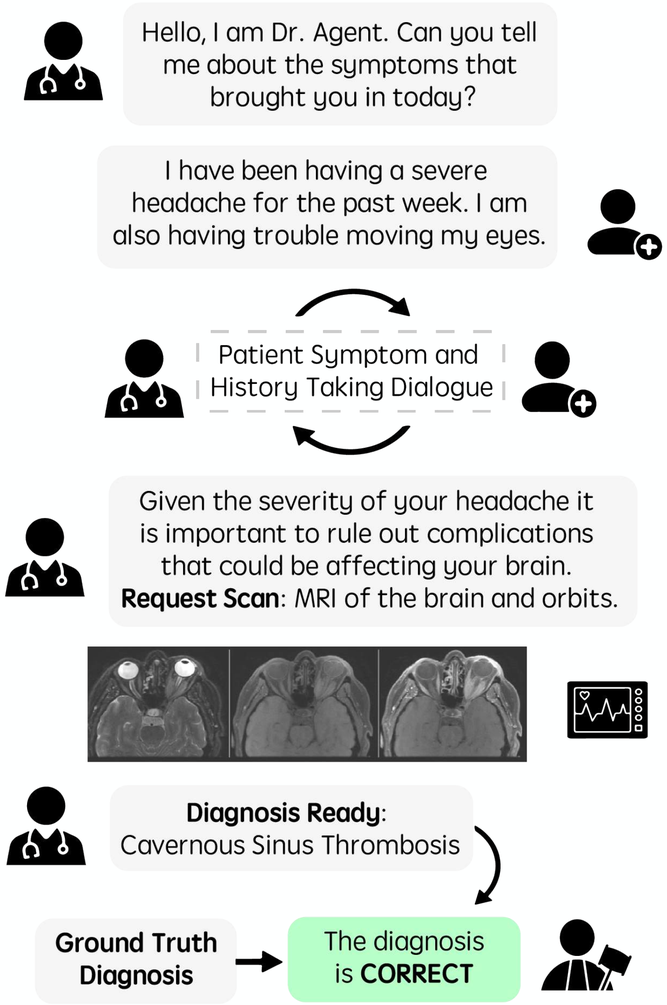
Example: **Multimodal AgentClinic**

Schmidgall et al., (2025)

Multi-agent systems!



Example AgentClinic **Simulation**



From World Models to Embodied Agents

World models: learn representations and dynamics to *predict* / *imagine* future states.

EMBODIED AGENT

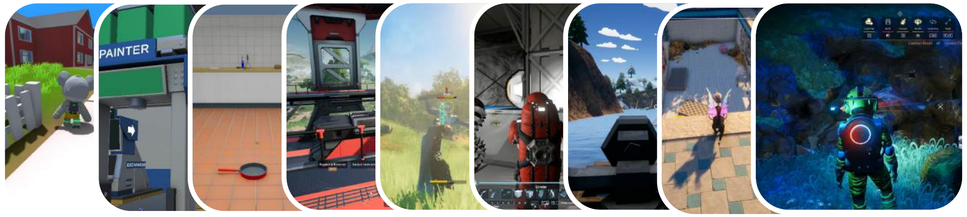
An agent that is situated in an environment and can act, creating a closed perception–action loop.

Key shift:
From “model the world” to “do things in the world”:
Mapping instruction + multimodal observations → action sequences.

What becomes challenging?
Learning grounded, recoverable behavior in a closed loop where feedback is delayed and possibly sparse, errors compound, and actions must respect affordances and safety.

Example: **Scalable Instructable Multiworld Agent**

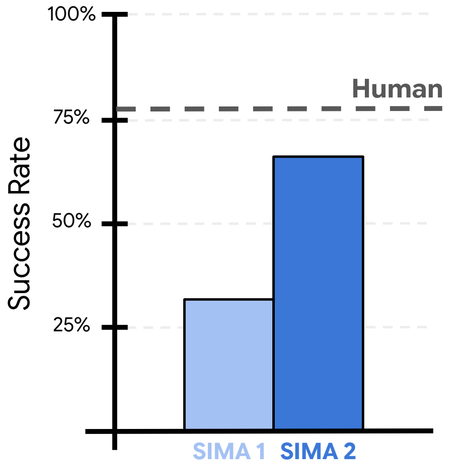
Dialog, Reasoning, and Embodied Acting Capabilities



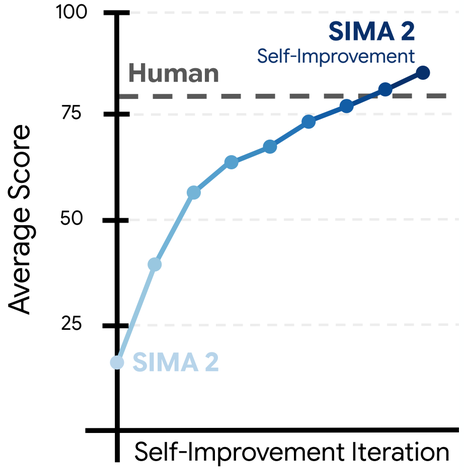
Training Environments

Held-Out Environments

Improved Performance



Self-Improvement



Example: **Scalable Instructable Multiworld Agent** (SIMA)

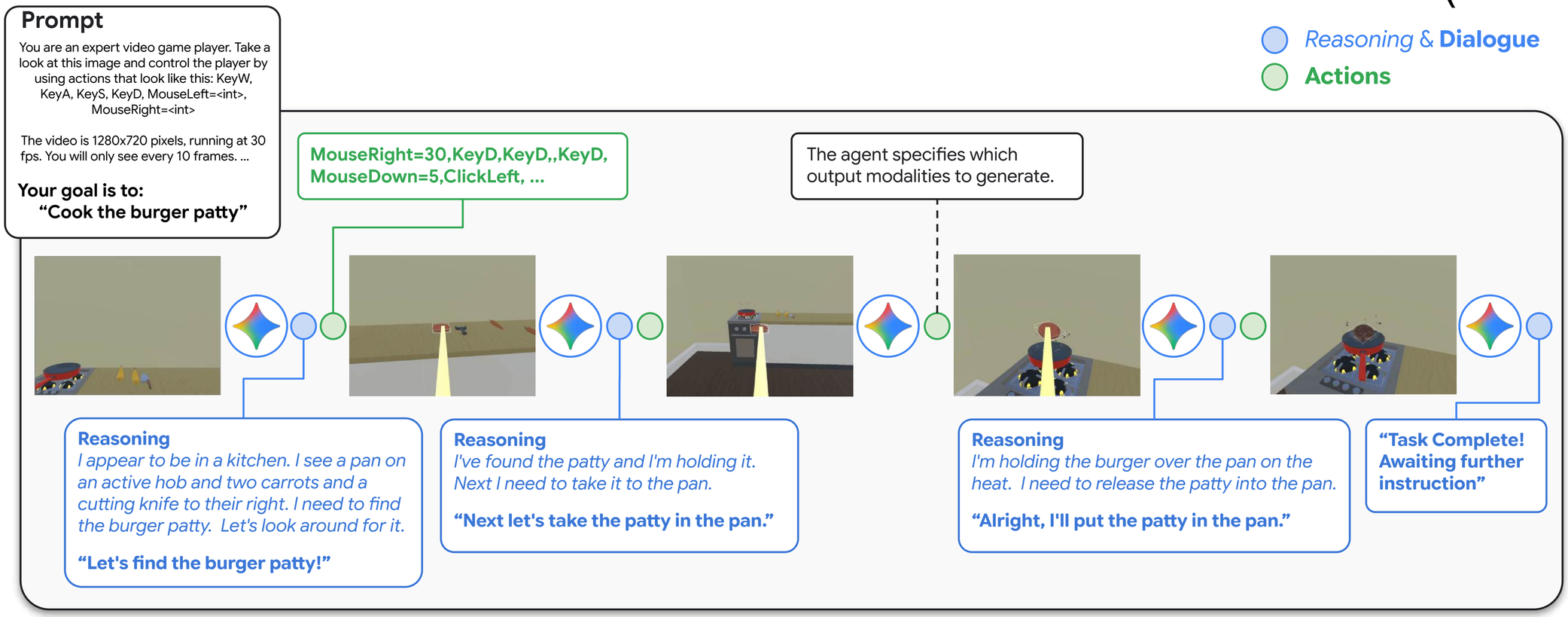


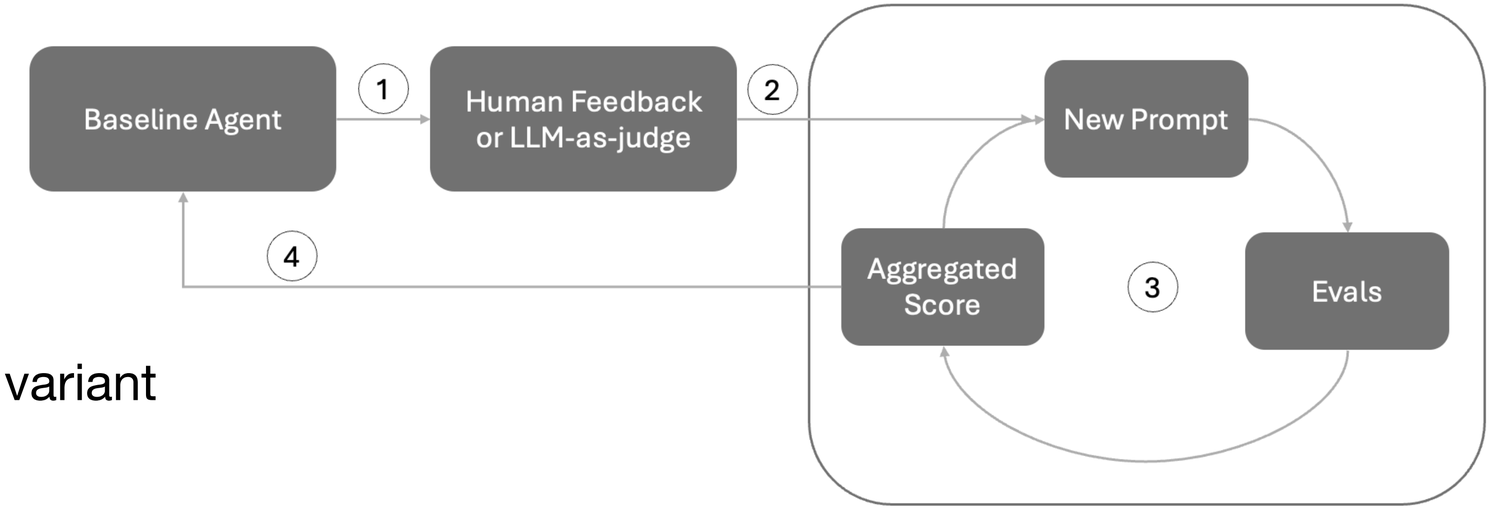
Figure 3 | **Agent-Environment Interface**. The agent receives a prompt that includes the current instruction. Conditioning on recent frames, the agent outputs internal reasoning, dialogue, and actions, with the agent specifying which modalities to produce at any given step.

Self-Evolving Agents

Core idea:
An agent can improve itself by running an automated propose → evaluate → select → update loop.

How?

1. Run baseline agent: execute tasks in the environment and log traces (actions, tool calls, failures).
2. Get a training signal: human feedback or LLM-as-judge scores the behavior (success, quality, safety).
3. Optimize "what the agent does": propose variants (new prompt or policy, tool-use rules, skills, etc.) and benchmark them with evals → aggregated score.
4. Update the agent: deploy the best variant back into the baseline agent.

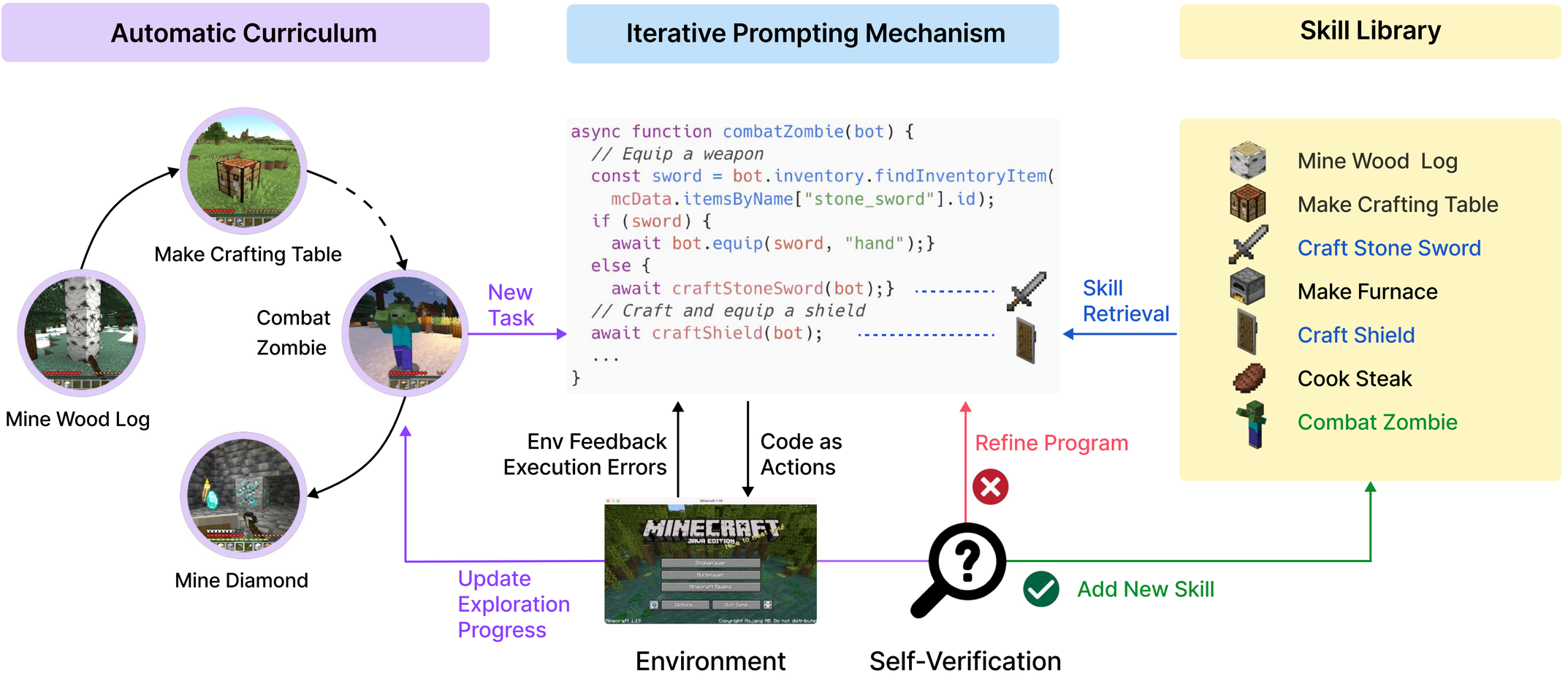


OpenAI Cookbook

Self-Evolving Agents

Wang et al., (2025)

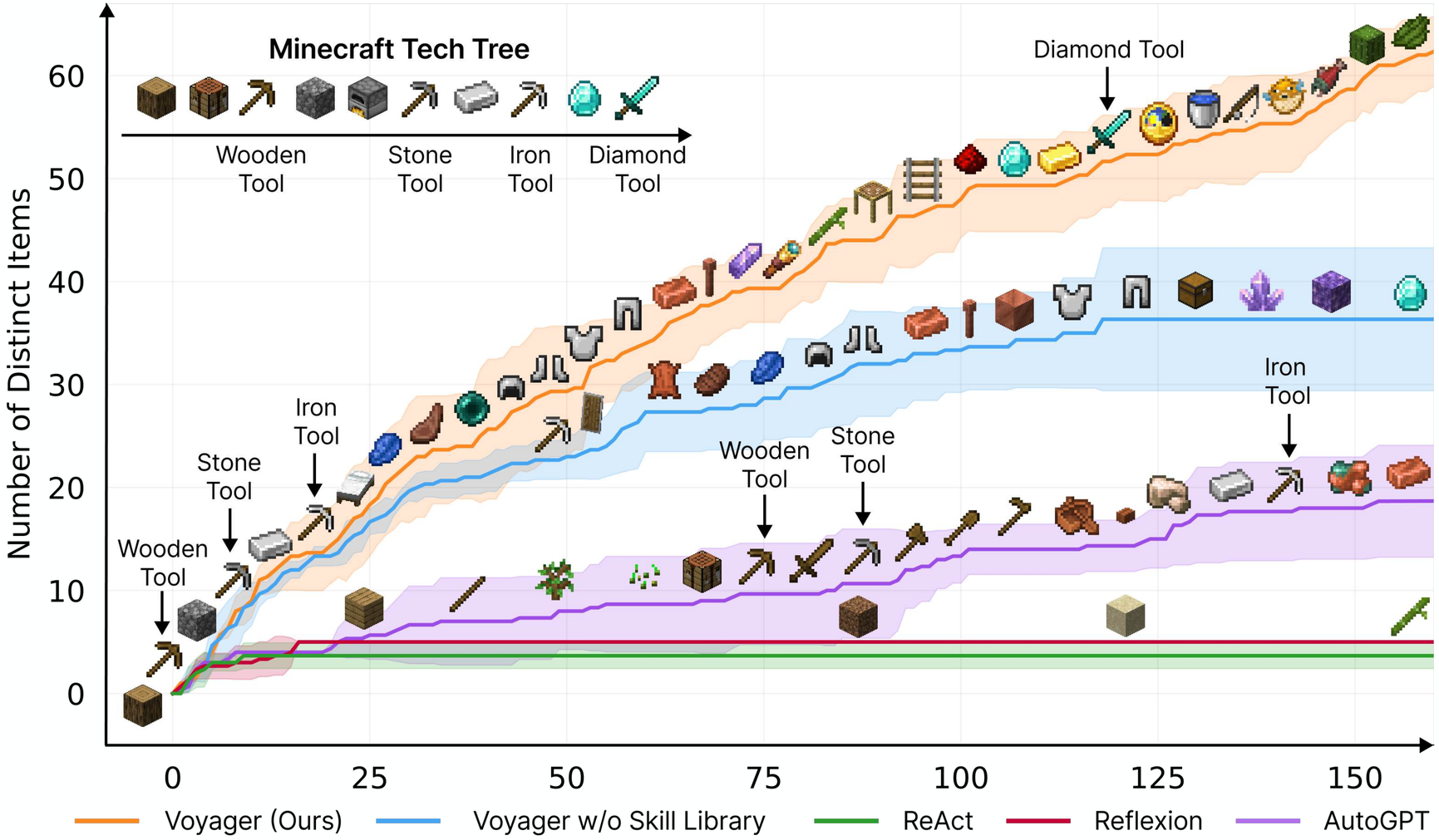
Example: **Voyager**, an Open-Ended Embodied LLM Agent



Self-Evolving Agents

Wang et al., (2025)

Example: **Voyager**, an Open-Ended Embodied LLM Agent



Self-Improving Artificial Intelligence

Beyond self-evolving agents:
Not just tuning prompts/skills, but improving the capabilities of the whole AI system over time.

What improves? The agent, its tools, its training data, its world models, an its evaluators, i.e., better tests, simulators, judges.

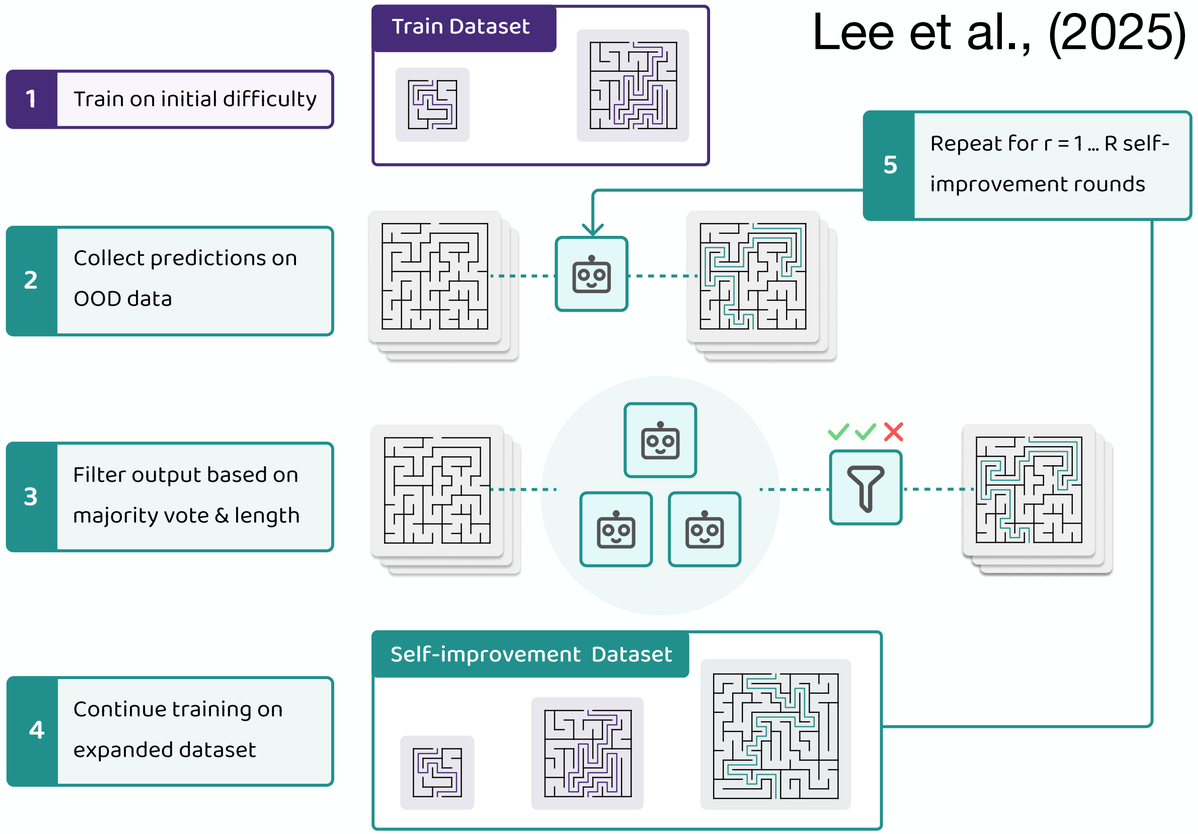
Active research area!

... e.g., self-improving Transformer, where the model iteratively generates its own training data and progressively learns from harder examples.

... other approaches use reinforcement learning.

... let's see what the next years will bring!

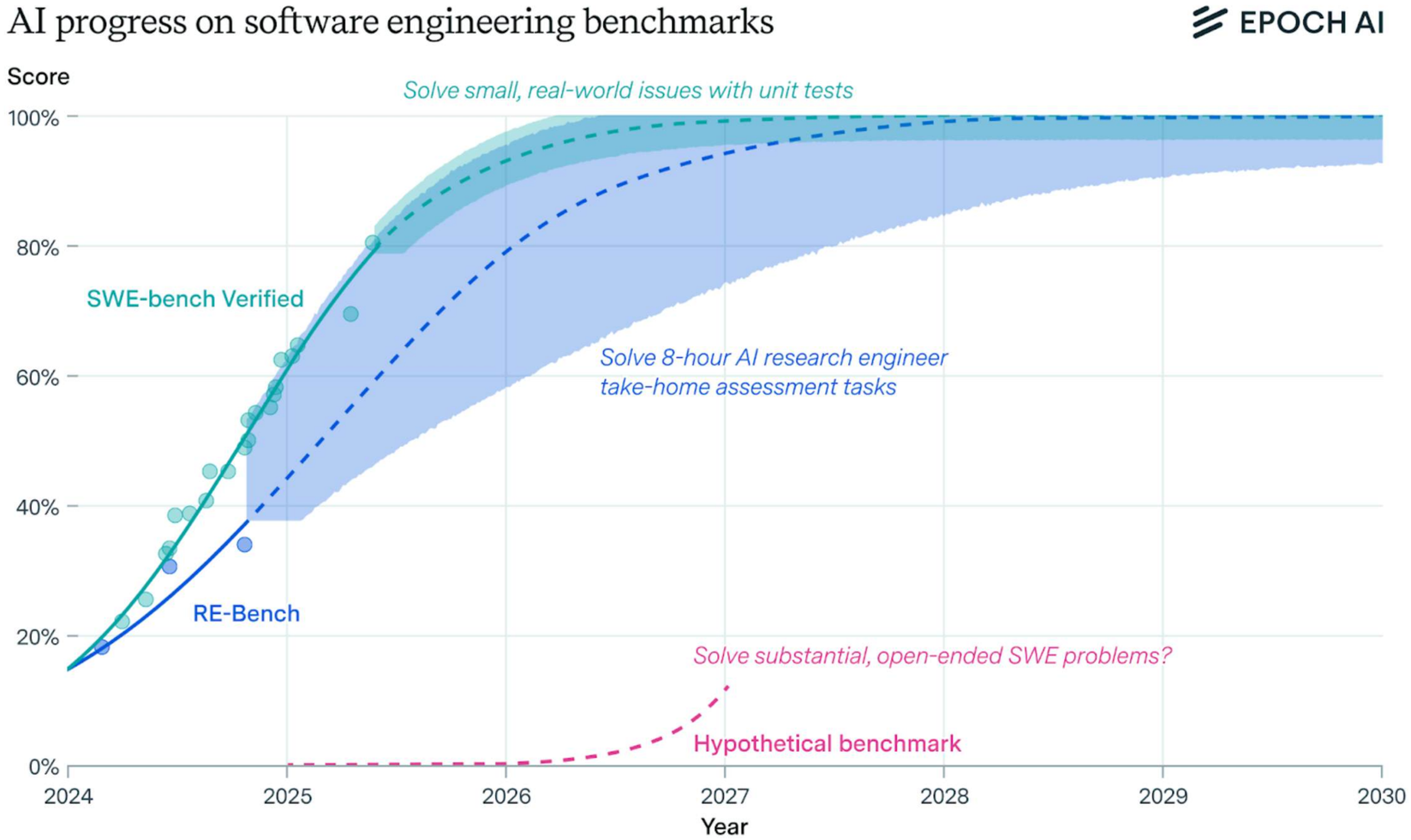
Lee et al., (2025)



How Will the Future Look Like?

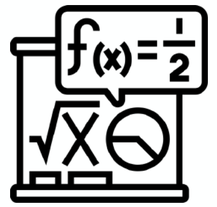
What will AI look like in 2030?

"Scaling is likely to continue to 2030."



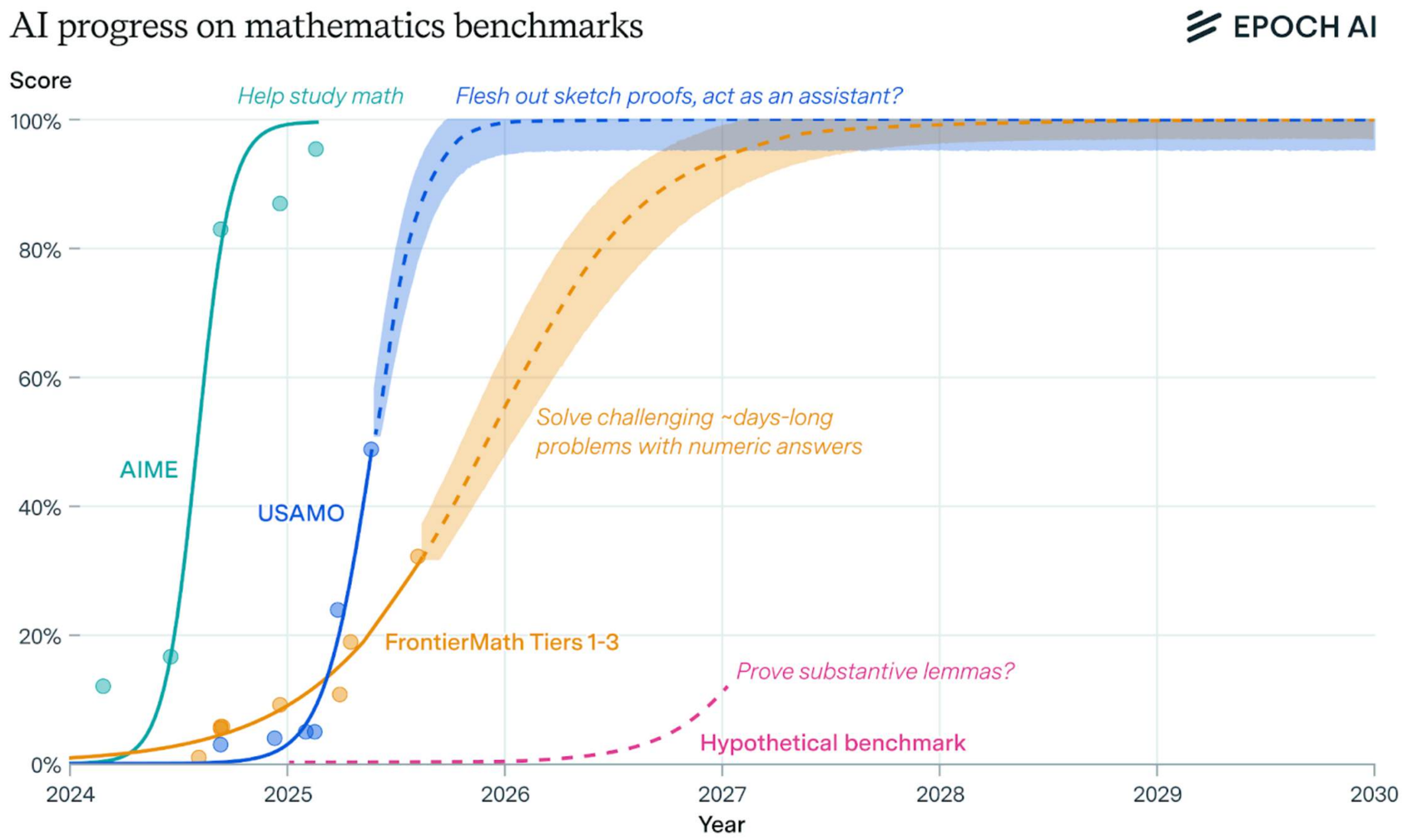
How Will the Future Look Like?

math



What will AI look like in 2030?

"AI may soon act as a research assistant, fleshing out proof sketches or intuitions. Early accounts already document AI being helpful in mathematicians' work."



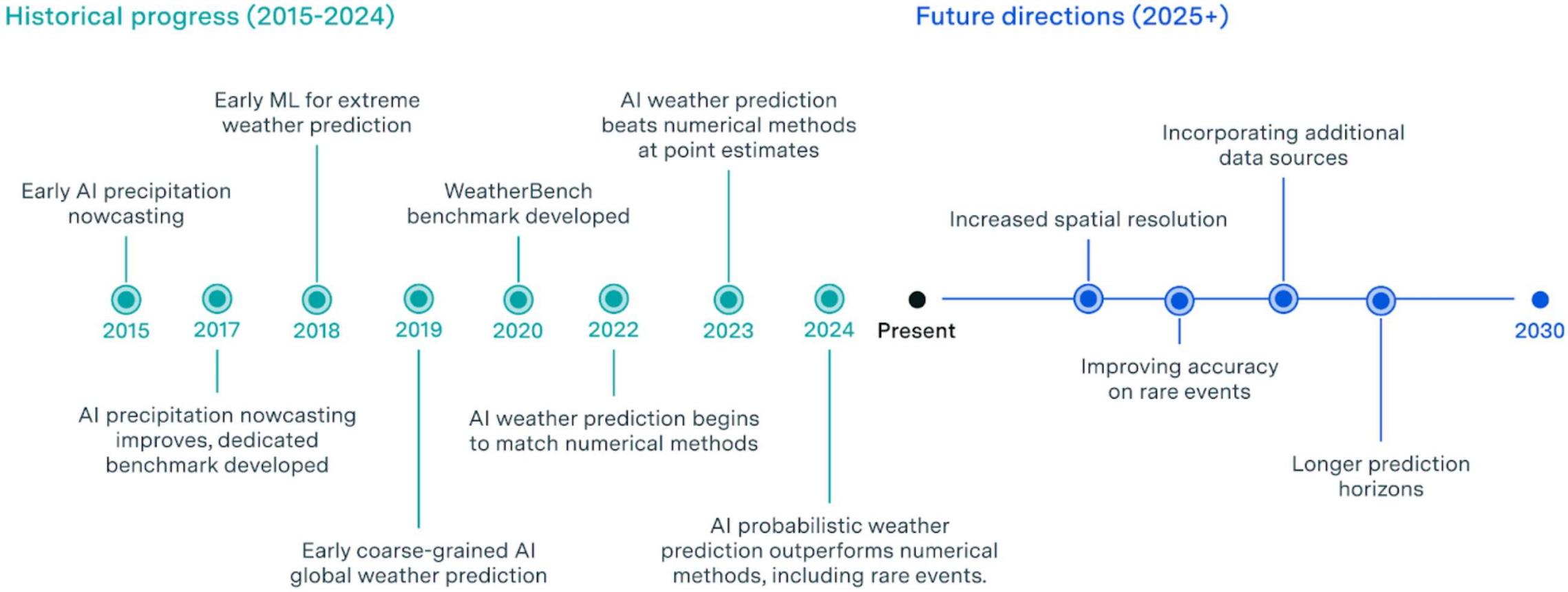
How Will the Future Look Like?

weather

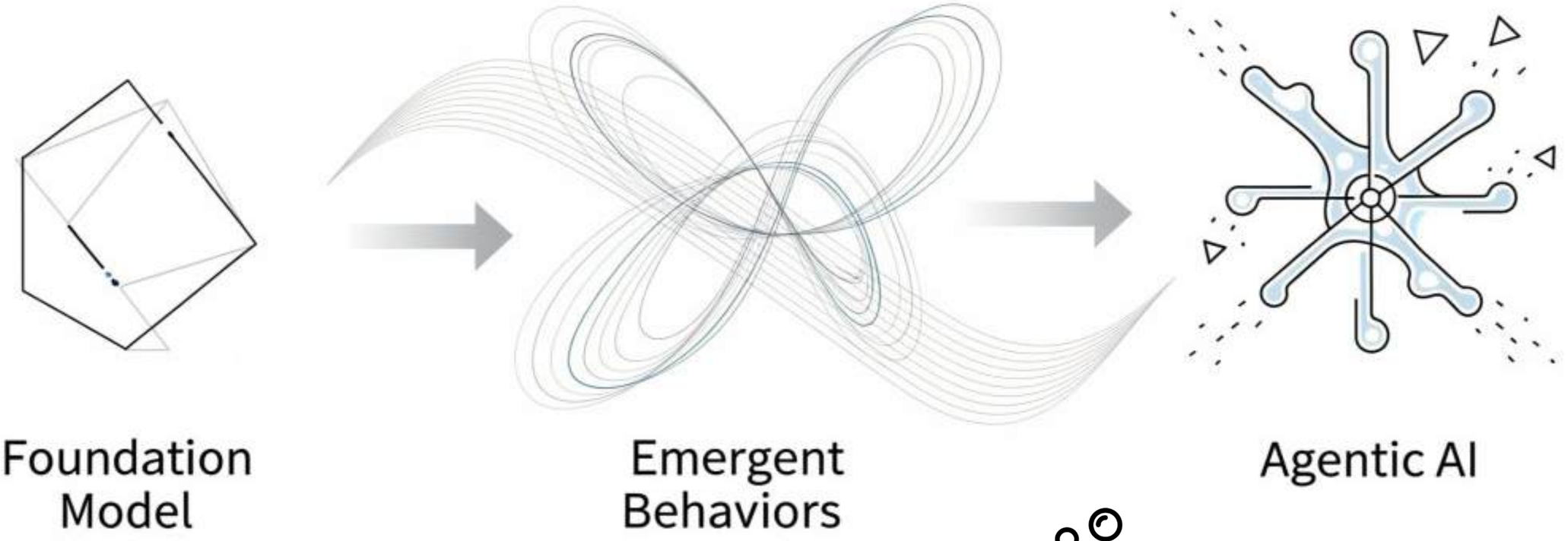
"AI weather prediction can already improve on traditional methods from hours up to weeks. Moreover, AI methods are cost-effective to run, and could improve further with more data."

AI progress on weather prediction

EPOCH AI

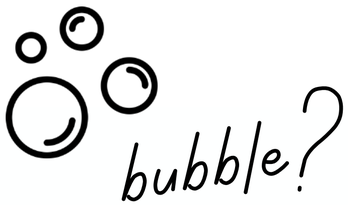


Emergent Behavior of Large-Scale AI



… we are definitely not there yet.

… thought-provoking timeline by AI 2027. 🤯



Blog Post by
Julian Schrittwieser

This Week's Exercise Sheet

Code Notebook 13

You will design an implement an agent loop that coordinates interactions between the LLM and tools, handling reasoning, tool execution, and iterative feedback until a final answer is produced.

Your goal would be to build and integrate local tools with proper schemas, validation, safety mechanisms, and tests.

Further, you will create a local MCP server and integrate local tools with it, along with play around with external MCP servers

CS-461

Foundation Models and Generative AI

It's a Wrap!