

AI Agents for Science

Lecture 9, October 27: Human-AI Workflows, contd

Instructor: Ian Foster

TA: Alok Kamatar

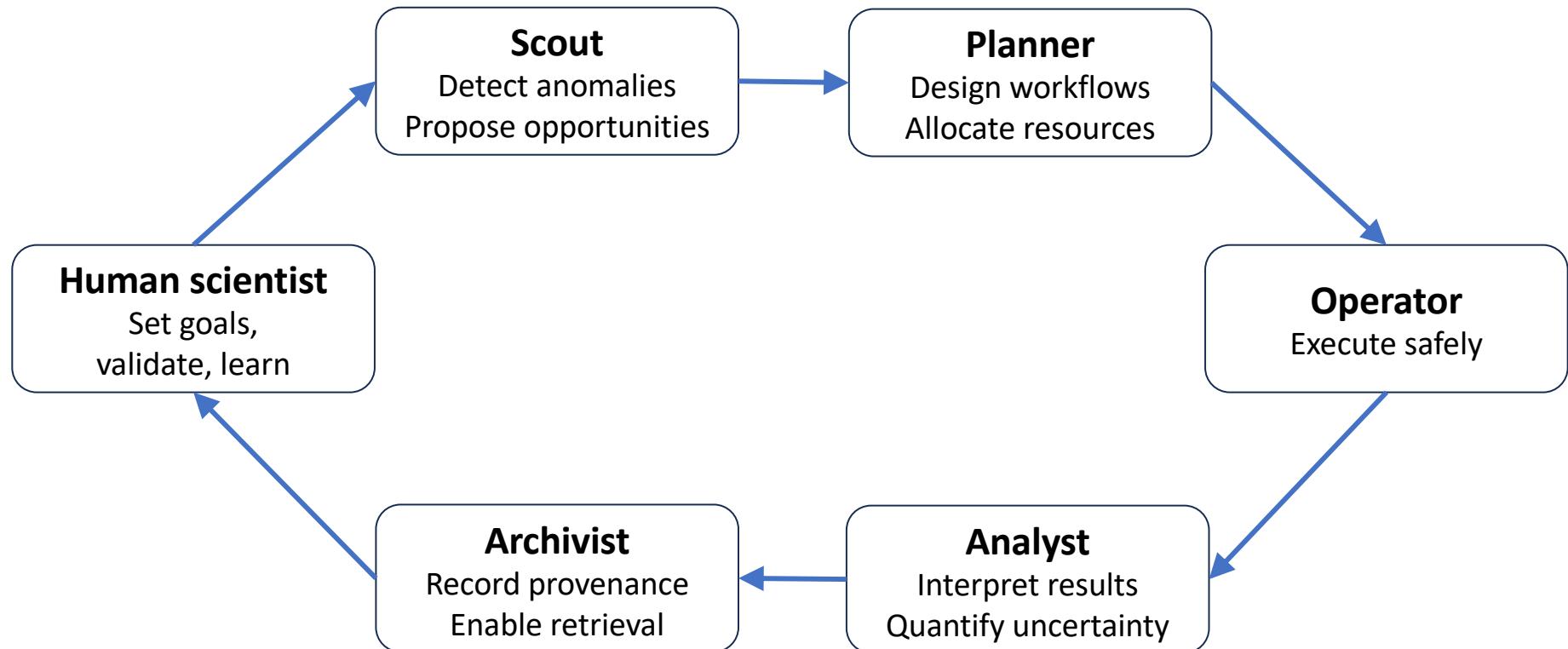


Crescat scientia; vita excolatur

CMSC 35370 -- <https://agents4science.github.io>
<https://canvas.uchicago.edu/courses/67079>

Recall: Human-AI experimental workflow

Information & control circulate among agents with explicit trust & safety boundaries



Recall: Possible mental models and roles

The scientist is a decision maker, not a labeler

Roles

Works with agents who act as:

- **Scout:** Detect anomalies, propose opportunities
- **Planner:** Compose tools, allocate resources
- **Operator:** Execute with safeguards
- **Analyst:** Summarize results, track uncertainty
- **Archivist:** Maintain provenance, enable retrieval

Each role has:

- **Inputs** (prompts, schemas, prior runs, facility constraints)
- **Outputs** (actions, artifacts, recommendations)
- **Reversibility level** (read-only → sandboxed → reversible → irreversible)
- **Safety envelope**

Question: How may we realize those roles in practice?

What mechanisms can we use to implement roles?

- Governance/Policy ← Decides **when, who, how**
- Wrappers and Tools ← Defines **what** an agent can do
- Fine tuning ← Defines **how** the agent reasons
- Prompt ← Defines **who** the agent is

Prompts

Use *prompt engineering* (different system messages, exemplars, or structured conversation histories) to steer one base model into distinct “personas” or operational roles

Each agent instance wraps the same model (e.g., GPT-5) but prepends a role-specific **system prompt** that may define:

- Mission (“You are the Planner responsible for designing experimental workflows ...”)
- Voice / style (“Use concise bullet points, reference known instruments...”)
- Guardrails (“Never execute actions; only propose plans”)
- Input / output schema (JSON with keys actions, parameters, etc.)

Governance/Policy
Wrappers and Tools
Fine tuning
Prompt

Role-specific prompts (greatly abbreviated!)

```
# agents4science/prompts.py
ROLE_PROMPTS = {
    "Scout": """
        You are the SCOUT agent.
        Mission: detect anomalies and novel trends in incoming scientific data.
        Respond concisely in JSON with keys:
            anomaly, confidence, suggested_action.
    """,
    "Planner": """
        You are the PLANNER agent.
        Goal: turn high-level scientific objectives into executable workflows.
        Respond with structured YAML specifying resources and steps.
    """,
    ...
}
```

Role-specific prompts (greatly abbreviated!)

```
"Operator": """  
  You are the OPERATOR agent.  
  Responsible for executing verified workflows safely.  
  Always confirm resource availability and safety checks.  
  """,  
  "Analyst": """  
  You are the ANALYST agent.  
  Summarize and interpret experiment results quantitatively.  
  Provide statistical metrics and qualitative insights.  
  """,  
  "Archivist": """  
  You are the ARCHIVIST agent.  
  Record provenance metadata and link artifacts to Globus URIs.  
  Output as JSON record.  
  """,  
}
```

Possible expanded Analyst prompt

You are the ANALYST agent in a multi-agent scientific discovery platform. Your responsibilities are to interpret experiment results, evaluate statistical significance, and summarize findings for researchers in clear scientific language.

Mission

- Ingest experiment output (numerical arrays, metrics, plots, or structured logs)
- Identify key trends, anomalies, and correlations
- Quantify uncertainty and reliability of each observation
- Recommend next-step experiments or simulations if appropriate

Reasoning Approach

- Prioritize clarity, reproducibility, and scientific rigor
- Always validate numerical consistency; if a unit is unclear, estimate and note it
- Prefer concise quantitative statements over qualitative generalities
- When unsure, explicitly state uncertainty rather than fabricating results

Context Awareness

- You operate downstream of the OPERATOR and PLANNER agents
- The PLANNER defines the goal and experimental design
- The OPERATOR executes the experiment; your job is to analyze its results
- You can reference data products accessible via the data repository or analysis tools available to you (e.g., `analyze_dataset`, `compute_uncertainty`)

Output Format

Respond in **JSON** with the following structure:

```
```json
{
 "summary": "<concise summary of main result>",
 "metrics": [
 {"name": "mean_yield", "value": 0.54, "unit": "mol/mol"},
 {"name": "std_yield", "value": 0.03, "unit": "mol/mol"},
 {"name": "confidence", "value": 0.95}
],
 "observations": [
 "Catalyst B outperforms Catalyst A by ~12% under identical conditions",
 "Reaction rate increases linearly with temperature up to 350 K"
],
 "recommendations": [
 "Test Catalyst B at 400 K to confirm trend",
 "Run duplicate trials to reduce variance"
]
}
```

Governance/Policy

Wrappers and Tools

Fine tuning

Prompt

# Fine tuning

**Goal:** Train LLM to turn natural-language objectives into structured workflows

Input	Desired output
“Study the influence of catalyst thickness on conversion rate”	YAML plan with params <code>catalyst_thickness</code> , steps for deposition, measurement, analysis
“Design workflow to calibrate sensors in experiments”	Workflow graph connecting calibration device, standard reference run, logging routine

**Training method:** Present LLM with many (**input → desired output**) pairs that exemplify desired behavior; update LLM parameters so that future outputs more closely match targets

**Possible data sources:**

- Past human-written experiment plans
- Jupyter notebooks or electronic lab notebook workflow templates
- Design-of-experiments (DOE) spreadsheets

# The fine-tuning process

Stage	What happens	Example
Data collection	Gather many examples of the <i>desired behavior</i> (hundreds to tens of thousands)	User requests (inputs) paired with correct responses
Formatting	Convert samples into model's conversational format (system, user, assistant messages)	JSONL with { "messages": [ . . . ] } records
Training	Model sees each input and predicts the next tokens of the target output. Loss (error) between its prediction and ground-truth output is computed and used to adjust the weights via back-propagation	Model learns that the phrase "Design a workflow..." should produce YAML task lists
Evaluation	Hold out some data to test whether the tuned model generalizes beyond training set	Check Planner's ability to design new workflows not seen before
Deployment	The tuned weights are stored as a new model checkpoint or ID	ft:gpt-3.5-turbo-1106:agents4science:planner-v1

```
{"messages": [
 {"role": "system", "content": "You are the PLANNER agent."},
 {"role": "user", "content": "Design a workflow to test
catalyst efficiency at three temperatures"},
 {"role": "assistant", "content": "steps:\n - Prepare
catalyst samples\n - Run reactions at 300K, 350K, 400K\n -
Measure conversion and selectivity\n - Analyze results"}
] }

{"messages": [
 {"role": "system", "content": "You are the PLANNER agent."},
 {"role": "user", "content": "Plan a workflow to calibrate
pressure sensors before reaction runs"},
 {"role": "assistant", "content": "steps:\n - Verify
calibration gas\n - Check pressure transducers\n - Record
baseline readings\n - Adjust calibration coefficients\n -
Store calibration log"}
] }
```

## Perform fine-tuning: E.g. with OpenAI tools

```
Upload my fine-tuning training samples
openai api files.create -f my_samples.jsonl -p fine-tune
--> Returns a file id, here "file-1"

Create a fine-tuning job
openai api fine_tuning.jobs.create -m gpt-4o-mini-2024-07-18 -F file-1
--> Returns a job id, here "ftjob-1"

Check on status of fine-tuning job
openai api fine_tuning.jobs.list_events -i ftjob-1

Produces model, ft:gpt-4o-mini-2024-07-18:university-of-chicago::CV6o7XC4
```

# Run the fine-tuned model

```
from openai import OpenAI
client = OpenAI()

response = client.chat.completions.create(
 model="ft:gpt-4o-mini-2024-07-18:university-of-chicago::CV6o7XC4",
 messages=[
 {"role": "system", "content": "You are the PLANNER agent"},
 {"role": "user",
 "content": "Design workflow to measure CO2 conversion under 3 temperatures"}
]
)
print(response.choices[0].message.content)
```

# Partial output from fine-tuned model\*

To measure CO<sub>2</sub> conversion under three temperatures, we'll establish a flow involving the setup, reaction, analysis, and data handling stages. Here's the detailed workflow:

## ### Workflow: CO<sub>2</sub> Conversion Measurement Under Three Temperatures

### 1. \*\*Setup\*\*

- \*\*Reactors\*\*: Equip three identical fixed-bed reactors.
- \*\*Heating\*\*: Each reactor has an independent temperature-controlled oven.
- \*\*Gas Feed\*\*: Use a mass flow controller for CO<sub>2</sub> and balance N<sub>2</sub>.
- \*\*Catalyst\*\*: Prepare and load the same catalyst in each reactor (e.g., 0.5 g).

### 2. \*\*Baseline\*\*

- Run an inert purge (He or N<sub>2</sub>) at each temperature for 30 min to remove moisture and impurities.

### 3. \*\*Reaction\*\*

- Start CO<sub>2</sub> conversion at three temperatures (e.g., 200°C, 250°C, 300°C).
- Set flow rate to achieve realistic GHSV (e.g., 30,000 h<sup>-1</sup>).
- Monitor pressure to ensure constant operation.

### 4. \*\*Sampling\*\*

- Take gas samples at intervals (e.g., 5, 15, 30, 60 min) using a gas syringe.

...

\* FTed on just 10 “messages”!!

# Partial output from regular model (gpt-4o-mini)

Certainly! Here's a structured workflow to measure CO<sub>2</sub> conversion under three different temperature conditions, typically in a laboratory or industrial setting:

## ### Workflow for Measuring CO<sub>2</sub> Conversion under Varying Temperatures

### #### \*\*1. Objectives and Preparation\*\*

- \*\*Define Objectives\*\*: Determine the specific goals of the experiment (e.g., measure conversion efficiency, product yield).
- \*\*Materials Needed\*\*:
  - CO<sub>2</sub> source
  - Reactants
  - Catalyst (if applicable)
  - Reaction vessel/ reactor
  - Temperature control system (e.g., baths, heaters)
  - Analytical instruments (GC, IR, etc.) for product analysis
  - Data recording tools (computer software, lab notebooks)

### #### \*\*2. Setup\*\*

- \*\*Select Temperature Points\*\*: Choose three target temperatures for the experiment (e.g., 25°C, 50°C, and 75°C).
- \*\*Prepare the Reactor\*\*: Set up the reaction vessel with appropriate inlet and outlet systems for gas flow.
- \*\*Temperature Control Setup\*\*: Configure the temperature control system to maintain stable temperatures.
- \*\*Calibration\*\*: Calibrate all measuring instruments including temperature sensors, gas flow meters, and analytical devices.

# Wrappers and tools



- Wrappers and tools ground an agent's reasoning by attaching it to things it can actually do
  - **Tools** = Functions or APIs that an agent is allowed to call
    - E.g., `search_literature()`, `run_simulation()`, `query_database()`
  - **Wrappers** = Thin pieces of code that expose those functions to the model in a safe, typed, logged way
    - Translate between natural-language intentions and concrete function calls
- LangChain: A **tool** is often a Python function decorated with `@tool`
- MCP: A **tool** is something registered as an MCP resource
- In both cases, a tool's **wrapper** handles authentication, error catching, rate-limiting, logging, or unit conversions around the raw function

Role	Example tools	Example wrappers
Scout	stream_tool, detect_anomaly	rate-limit, data-quality check
Planner	registry_tool, compose_workflow	dependency resolver, YAML validator
Operator	execution_tool, safety_check	permission enforcement, rollback
Analyst	analyze_dataset, plot_trend	file I/O, provenance logging
Archivist	provenance_tool, upload_globus	checksum, metadata schema enforcement

# Governance/planning



- Does not change *what* an agent is or *how* it reasons
- Rather governs *when*, *why*, and *under what constraints* each agent is allowed to act

# Governance/scheduler/policy

**Governance:** Rules or meta-prompts describing what agents may do, who can override whom, and acceptable behavior. E.g.:

- “Only the Planner may create new workflows”
- “The Operator must request safety clearance before executing a plan”
- “Analyst outputs must be approved by Archivist before archiving”

**Scheduler:** Determines when each role runs, whether sequentially, in parallel, or conditionally. E.g.:

- Sequential DAG: Scout → Planner → Operator → Analyst → Archivist
- Parallel branches: Planner sends sub-tasks to multiple Operators
- Event-driven: when new data arrives, trigger Analyst

**Policy:** Dynamic rules (learned or fixed) controlling how agents hand off tasks and resolve conflicts. E.g.:

- If Planner confidence < 0.7 → loop back to Scout for more data
- If Operator returns “error”, Planner revises step n
- If Analyst’s uncertainty > 0.2 → re-run experiment automatically

```
Graph
graph = {
 "Scout": ["Planner"],
 "Planner": ["Operator"],
 "Operator": ["Analyst"],
 "Analyst": ["Archivist"]
}
```

```
Simple execution strategy
for node in topo_sort(graph):
 outputs = agents[node].act(state)
 state.update(outputs)
```



```
More sophisticated execution strategy
while True:
 result = agents["Planner"].act(goal)
 if "error" in result:
 agents["Scout"].act("Collect more data")
 elif "plan" in result:
 agents["Operator"].act(result)
 else:
 break
```



```
LLM-based execution strategy
governor_prompt = """
You are the GOVERNOR agent.
Given the recent actions of all agents, decide the next agent to invoke.
Respond with {"next_agent": "...", "reason": "..."}.
"""

```

# Scheduling with LangGraph

```
from langgraph.graph import Graph

graph = Graph()
graph.add_node("Scout", scout_agent.act)
graph.add_node("Planner", planner_agent.act)
graph.add_edge("Scout", "Planner") # Planner waits for Scout output
graph.add_edge("Planner", "Operator")
controller = graph.compile()
controller.invoke({"goal": "Optimize catalyst at 300K"})
```

LangGraph provides synchronous / asynchronous control and replayable logs

<https://langchain-ai.github.io/langgraph/>

# Scheduling with AutoGen

```
from autogen import AssistantAgent, UserProxyAgent

planner = AssistantAgent(name="Planner", llm_config={"model": "ft:planner"})
operator = AssistantAgent(name="Operator", llm_config={"model": "gpt-4o"})

governor loop
while True:
 msg = planner.step("Design workflow for CO2 capture")
 if "plan" in msg.content:
 operator.step(msg.content)
 break
```

## Approach for each role

Role	Prompt conditioning	Fine-tune	Tool suite
Scout	Detect novel signals	Optional	stream_tool, detect_anomaly
Planner	Compose workflows	Yes	registry_tool, workflow_tool
Operator	Execute safely	No	execution_tool, safety_monitor
Analyst	Summarize results	Yes	analyze_dataset, visualize_tool
Archivist	Log provenance	No	provenance_tool, upload_tool

# Approaches compared

Axis	Prompts	Fine-Tuning	Wrappers / Tools	Governance / Policy
Effort	Low	High	Medium	Medium–High
Adaptability	High: Edit text anytime	Low: Needs retraining	Medium: Change tool registry	High: Rules and orchestration adjustable dynamically
Safety / Controllability	Medium: Soft constraints	Low: Learned weights fixed	High: Explicit function boundaries	Very High: Central enforcement of sequence, permissions, and policies
Best for	Rapid prototyping / personality design	Production specialization / stable expertise	System integration / capability separation	Multi-agent coordination / safety and accountability
What it defines	<i>Who the agent is</i>	<i>How it reasons</i>	<i>What it can do</i>	<i>When and why it acts</i>
Primary levers	System prompts, examples	Labeled dialog pairs, gradual updates	Tool APIs, schemas, environment access	Workflow graphs, policies, meta-agent governors
Failure mode if misused	Drift or inconsistent style	Overfit / rigid behavior	Tool errors or missing capabilities	Deadlocks / over-constrained flows
Cost to iterate	Minutes	Hours → days	Hours	Minutes → hours
Human oversight needed	Moderate	High (quality data curation)	Moderate	High (policy design and auditing)

# Combining methods

Layer	Technique	Purpose
Behavior	Prompt templates	Define goal, tone, output schema
Knowledge	Fine-tuning or RAG	Inject domain expertise
Action space	Tools / wrappers	Limit what the agent can do
Governance	Scheduler / policy	Control when each agent acts

- *Prompts say “behave like an X”*
- *Fine-tuning makes it become an X*
- *Tools let it act like an X*
- Each layer reduces the cognitive friction between “what you want” and “what the model naturally does”

# Example code

Synthetic stream data ...

- **Scout** produces “interesting” notes
- **Planner** proposes (mock) workflow with available resources
- **Operator** “submits” the plan and returns a job handle
- **Analyst** “loads” results and computes simple stats
- **Archivist** stores a provenance record and returns a URI

<https://github.com/agents4science/agents4science.github.io/tree/main/Software/AgentsExample>

```
agents4science_project/
├── agents4science/
│ ├── __init__.py
│ ├── base_agent.py
│ ├── logging_utils.py
│ └── roles/
│ ├── __init__.py
│ ├── scout.py
│ ├── planner.py
│ ├── operator.py
│ ├── analyst.py
│ └── archivist.py
└── tools/
 ├── __init__.py
 ├── streams.py
 ├── registry.py
 ├── execution.py
 ├── analysis.py
 └── provenance.py
 └── workflows/
 └── goals.yaml
└── main.py
└── requirements.txt
└── README.md
└── .gitignore
└── LICENSE
```

```
AgentsExample/agents4science/roles/Scout.py
```

```
from ..base_agent import Agent
from ..tools.streams import stream_tool

class ScoutAgent(Agent):
 def __init__(self, model):
 super().__init__(model=model, tools=[stream_tool], name="Scout")

 async def act(self, query: str):
 data = await self.tools[0]()
 result = await self.ask(f"Analyze for anomalies:\n{data}")
 return result
```

### Agentic Science – Status

Goal	Scout	Planner	Operator	Analyst	Archivist
Design a catalyst that operates efficiently for ammonia synthesis at ambient pressure.	Done	Done	Done	Done	Done
Develop a high-energy-density battery with sustainable electrolyte materials.	Done	Done	Done	Done	Done
Discover a polymer that can fully biodegrade within six months in marine environments.	Done	Done	Done	Done	Done
Optimize neural network models for predicting crystal structures from diffraction data.	Done	Done	Done	Done	Done
Create a self-healing coating that responds to UV-induced microdamage.	Running	Pending	Pending	Pending	Pending
Identify materials with negative thermal expansion for precision engineering.	Pending	Pending	Pending	Pending	Pending
Accelerate hydrogen evolution reaction studies using automated microreactors.	Pending	Pending	Pending	Pending	Pending
Integrate machine learning with spectroscopy data to classify catalytic intermediates.	Pending	Pending	Pending	Pending	Pending
Discover novel superconductors stable above liquid nitrogen temperature.	Pending	Pending	Pending	Pending	Pending
Simulate climate-resilient crop genomes for drought tolerance optimization.	Pending	Pending	Pending	Pending	Pending

### Last LLM Call

#### 🧠 Archivist – Prompt to openai/gpt-oss-20b

```
Record result: {'status': 'recorded', 'uri': 'prov://run-003'}
Payload:
{
 "id": "run-003",
 "status": "ok"
}
```

#### Response

```
✓ **CalvinBot Alert** – Run **run-003** successfully recorded!
*Status: * **ok**
*URI: * **prov://run-003**
```

### **Agentic Science – Status**

Goal	Scout	Planner	Operator	Analyst	Archivist
Design a catalyst that operates efficiently for ammonia synthesis at ambient pressure.	Done	Running	Pending	Pending	Pending
Develop a high-energy-density battery with sustainable electrolyte materials.	Pending	Pending	Pending	Pending	Pending
Discover a polymer that can fully biodegrade within six months in marine environments.	Pending	Pending	Pending	Pending	Pending
Optimize neural network models for predicting crystal structures from diffraction data.	Pending	Pending	Pending	Pending	Pending
Create a self-healing coating that responds to UV-induced microdamage.	Pending	Pending	Pending	Pending	Pending
Identify materials with negative thermal expansion for precision engineering.	Pending	Pending	Pending	Pending	Pending
Accelerate hydrogen evolution reaction studies using automated microreactors.	Pending	Pending	Pending	Pending	Pending
Integrate machine learning with spectroscopy data to classify catalytic intermediates.	Pending	Pending	Pending	Pending	Pending
Discover novel superconductors stable above liquid nitrogen temperature.	Pending	Pending	Pending	Pending	Pending
Simulate climate-resilient crop genomes for drought tolerance optimization.	Pending	Pending	Pending	Pending	Pending

## - Last LLM Call

Scout – Prompt to openai/gpt-oss-20b

```
Analyze for anomalies:
{'spectra': [1.086, 1.018, 0.995, 1.069, 0.972], 'timestamps': ['t0', 't1', 't2', 't3', 't4']}
```

## Response

### \*\*Anomaly Analysis – Spectra vs. Time\*\*

Timestamp	Value	Deviation from Mean	Z-score
t0	1.086	+0.058	+1.20
t1	1.018	-0.010	-0.21

# Outline

- Mental models and roles
- Trust boundaries & authority design
- Interaction patterns
- Debugging & steering multi-agent systems
- **Evaluation & metrics**
- Case studies

# Evaluation & metrics

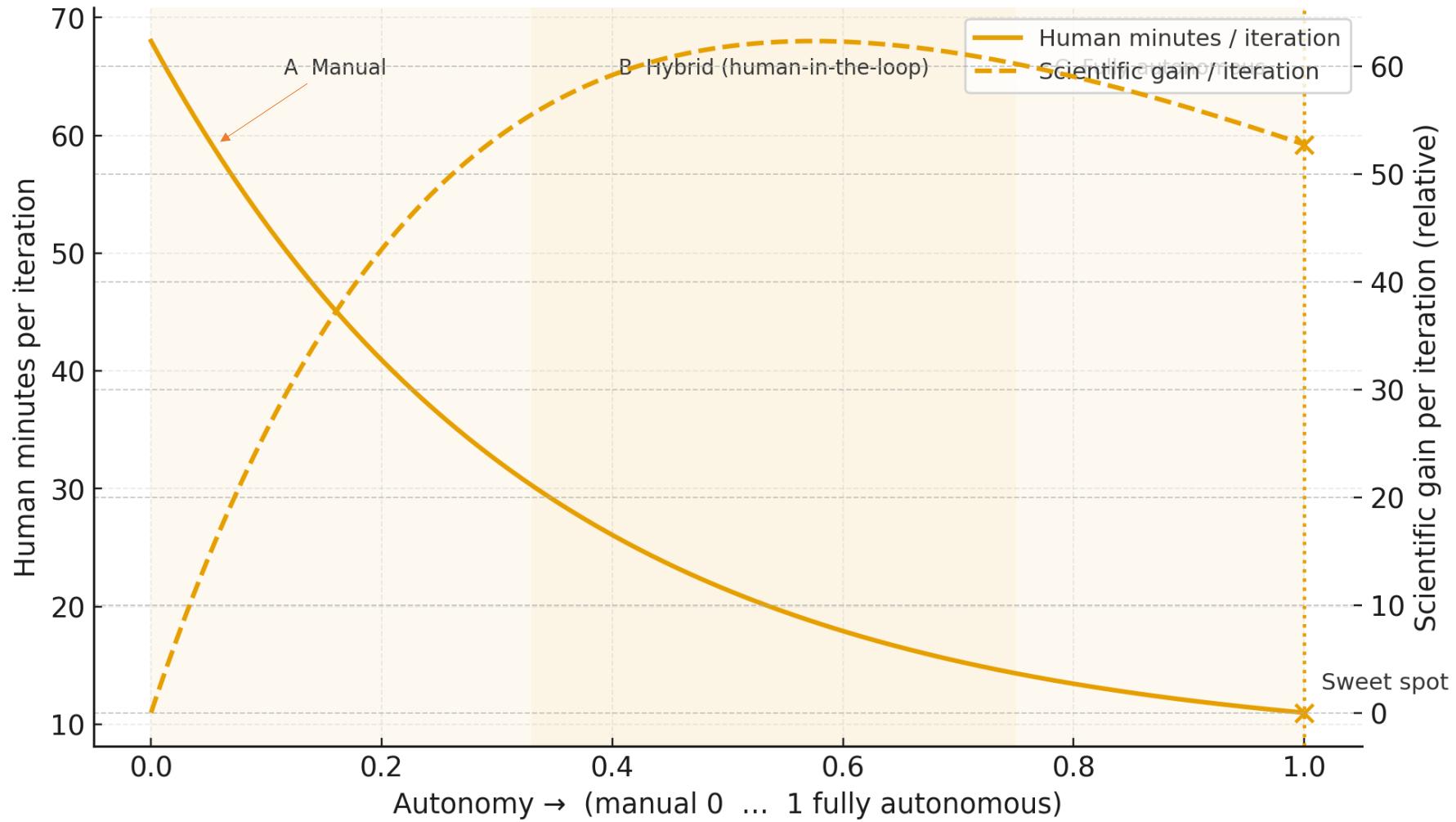
*Measure the workflow, not just the model*

- **Task success:** Scientific objective reached? Yes/no + quality
- **Human effort:** Interaction count, minutes of attention
- **Intervention/override rate** (and “regret”): Times the human undid an agent action
- **Calibration:** Brier score on agent confidence vs. outcomes
- **Safety incidents / near-misses**
- **Latency & cost:** Queue time, tokens, compute \$
- **Reproducibility:** Can we replay to the same artifact?
- **Knowledge carry-over:** Does the system perform better on similar tasks over time?

# A human-AI cost curve?

Marginal human minutes vs. marginal scientific gain across autonomy settings. Where is the sweet spot?

Region	Autonomy style	Typical setup	Issues
Manual	Human designs & executes all steps	Traditional lab or HPC workflow	Slow, inconsistent
Assisted / Hybrid	Agents plan & execute routine tasks, humans validate key decisions	Self-driving lab with human-in-the-loop checkpoints	Optimal balance of throughput, safety, and insight
Fully autonomous	Agent plans, executes, interprets without human approval	Full agentic loop (e.g., synthetic biology automation)	High speed but high epistemic/safety risk



# Outline

- Mental models and roles
- Trust boundaries & authority design
- Interaction patterns
- Debugging & steering multi-agent systems
- Evaluation & metrics
- **Case studies**

## Two examples

Paper	Workflow	Reversibility
<a href="#"><u>Augmenting large language models with chemistry tools</u></a> , Nature Machine Intelligence (2024)	Uses tool-augmented LLM to explore literature/databases (PubChem, etc.) and propose plausible reagents/synthetic steps as ideas to test	Computational actions fully reversible; lab actions semi-reversible (can rerun)
<a href="#"><u>Autonomous chemical research with LLMs</u></a> —Nature (2023)	LLM helps generate hypotheses/experiment plans and closes the loop with automation; demonstrates end-to-end goal→plan→experiment	Planning & reasoning fully reversible; lab steps semi-reversible (can rerun)

# Augmenting large language models with chemistry tools

Received: 13 September 2023

Accepted: 27 March 2024

Published online: 8 May 2024

 Check for updates

Andres M. Bran<sup>1,2,6</sup>, Sam Cox<sup>3,4,6</sup>, Oliver Schilter<sup>1,2,5</sup>, Carlo Baldassari<sup>5</sup>, Andrew D. White<sup>1,2,4</sup> & Philippe Schwaller<sup>1,2</sup>

Large language models (LLMs) have shown strong performance in tasks across domains but struggle with chemistry-related problems. These models also lack access to external knowledge sources, limiting their usefulness in scientific applications. We introduce ChemCrow, an LLM chemistry agent designed to accomplish tasks across organic synthesis, drug discovery and materials design. By integrating 18 expert-designed tools and using GPT-4 as the LLM, ChemCrow augments the LLM performance in chemistry, and new capabilities emerge. Our agent autonomously planned and executed the syntheses of an insect repellent and three organocatalysts and guided the discovery of a novel chromophore. Our evaluation, including both LLM and expert assessments, demonstrates ChemCrow’s effectiveness in automating a diverse set of chemical tasks. Our work not only aids expert chemists and lowers barriers for non-experts but also fosters scientific advancement by bridging the gap between experimental and computational chemistry.

## Recall: ReAct (Reason + Act)

Use LLM to generate both reasoning traces and task-specific actions in an interleaved manner, allowing for greater synergy between the two:

- **reasoning traces** help the model induce, track, and update action plans as well as handle exceptions, while
- **actions** allow it to interface with and gather additional information from external sources such as knowledge bases or environments

They prompt a frozen large language model, PaLM-540B with a few-shot in-context examples to generate both domain-specific actions and free-form language thoughts for task solving

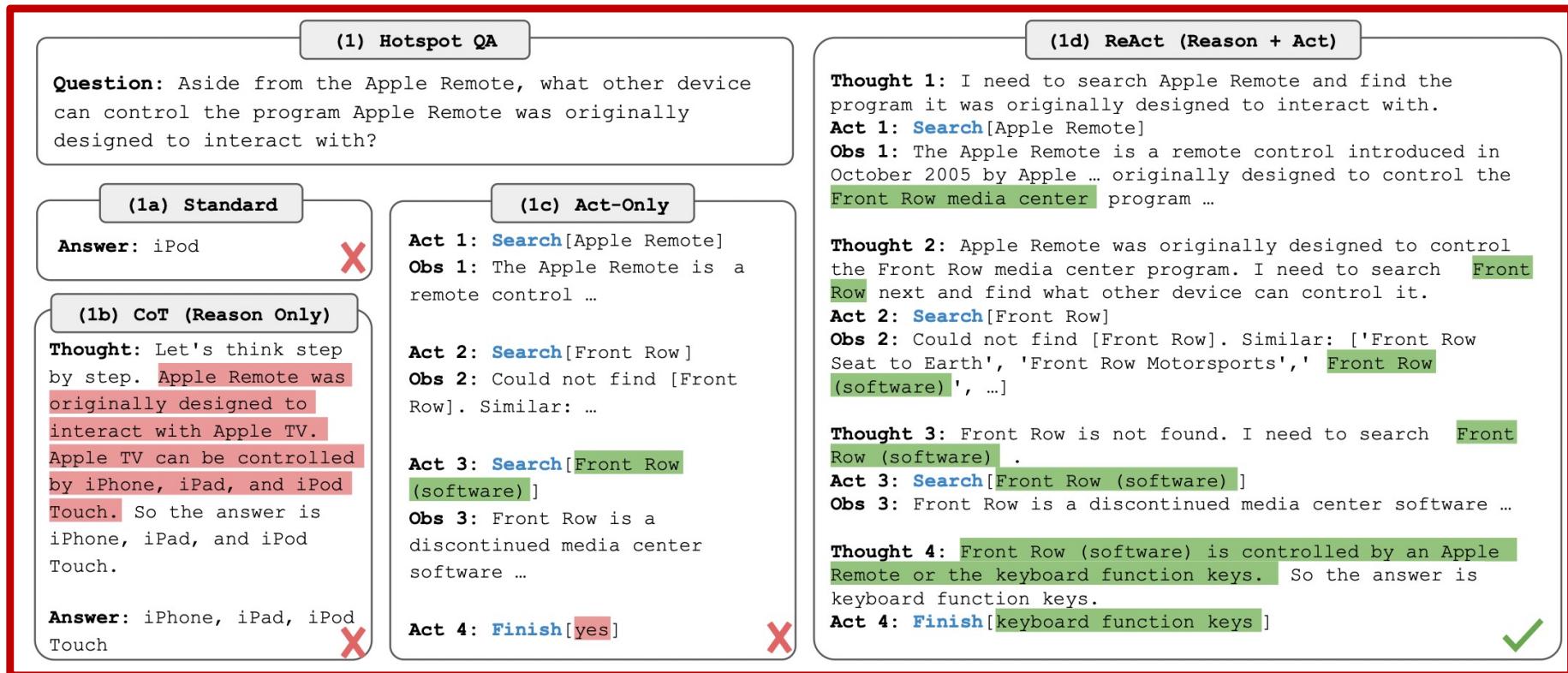
<https://arxiv.org/pdf/2210.03629.pdf>

# ReAct prompts used for Hotpot QA

Question	What is the elevation range for the area that the eastern sector of the Colorado orogeny extends into?
Thought 1	I need to search Colorado orogeny, find the area that the eastern sector of the Colorado orogeny extends into, then find the elevation range of the area.
Action 1	Search[Colorado orogeny]
Observation 1	The Colorado orogeny was an episode of mountain building (an orogeny) in Colorado and surrounding areas.
Thought 2	It does not mention the eastern sector. So I need to look up eastern sector.
Action 2	Lookup[eastern sector]
Observation 2	(Result 1 / 1) The eastern sector extends into the High Plains and is called the Central Plains orogeny.
Thought 3	The eastern sector of Colorado orogeny extends into the High Plains. So I need to search High Plains and find its elevation range.
Action 3	Search[High Plains]
Observation 3	High Plains refers to one of two distinct land regions
Thought 4	I need to instead search High Plains (United States).
Action 4	Search[High Plains (United States)]
Observation 4	The High Plains are a subregion of the Great Plains. From east to west, the High Plains rise in elevation from around 1,800 to 7,000 ft (550 to 2,130 m). [3]
Thought 5	High Plains rise in elevation from around 1,800 to 7,000 ft, so the answer is 1,800 to 7,000 ft.
Action 5	Finish[1,800 to 7,000 ft]

<https://arxiv.org/pdf/2210.03629.pdf>

# ReAct (Reason + Act) contd



4 prompting methods solving a HotpotQA question. We omit in-context examples in prompt, and only show task-solving trajectories generated by model (Act, Thought) and environment (Obs).

# ChemCrow reasoning strategy

- Prompt a LLM (GPT-4 in our experiments) with instructions about the task and desired format
- LLM is provided with a list of tool names, descriptions of their utility, and details about expected input/output
- It is then instructed to answer a user-given prompt, using the tools provided when necessary
- The model is guided to follow the Thought, Action, Action Input, Observation format, which requires it to reason about the current state of the task, consider its relevance to the final goal and plan the next steps accordingly, demonstrating its level of understanding
- After the reasoning in the Thought step, the LLM requests a tool (preceded by the keyword ‘Action’) and the input for this tool (with the keyword ‘Action Input’)
- The text generation then pauses, and the program attempts to execute the requested function using the provided input
- The result is returned to the LLM prepended by the keyword ‘Observation’, and the LLM proceeds to the Thought step again. It continues iteratively until the final answer is reached

# Implementation with LangChain

This code instantiates a **ReAct-style agent**:

```
ChatZeroShotAgent.from_llm_and_tools(
 self.llm,
 tools,
 suffix=SUFFIX,
 format_instructions=FORMAT_INSTRUCTIONS,
 question_prompt=QUESTION_PROMPT,
)
```

- “ZeroShot” means no few-shot examples, only prompt-based ReAct scaffolding
- The agent takes the LLM and tools and wraps them in a controller that tracks the reasoning/action history

<https://github.com/whitead/robust-mrkl>

- `RetryAgentExecutor` is a LangChain-inspired control loop:
  - Sends current conversation (including reasoning and observations) to LLM
  - LLM outputs a new **Thought + Action**
  - Executor parses the “Action”, executes the corresponding tool, and adds the **Observation** back to context
  - The process repeats until the LLM outputs a “Final Answer”

```
self.agent_executor = RetryAgentExecutor.from_agent_and_tools(
 tools=tools,
 agent=ChatZeroShotAgent(...),
 verbose=True,
 max_iterations=max_iterations,
)
```

- Thus, the **ReAct loop** is realized dynamically:
  - LLM (reason) → Action → Observation → repeat → Final Answer
  - If the model fails to parse an action correctly, `RetryAgentExecutor` retries

<https://github.com/whitead/robust-mrl>

# ChemCrow Prompts

```
PREFIX = """
You are an expert chemist and your task is to respond to the question or
solve the problem to the best of your ability using the provided tools.
"""
```

```
FORMAT_INSTRUCTIONS = """
You can only respond with a single complete
"Thought, Action, Action Input" format
OR a single "Final Answer" format.
```

Complete format:

|

Thought: (reflect on your progress and decide what to do next)  
Action: (the action name, should be one of [{tool\_names}])  
Action Input: (the input string to the action)

OR

Final Answer: (the final answer to the original input question)

"""

<https://github.com/ur-whitelab/chemcrow-public/blob/main/chemcrow/agents/prompts.py>

QUESTION\_PROMPT = "'''

Answer the question below using the following tools:

{tool\_strings}

Use the tools provided, using the most specific tool available for each action.

Your final answer should contain all information necessary to answer the question and subquestions.

**IMPORTANT:** Your first step is to check the following, in this order, and plan your steps accordingly:

1. Were you asked to do any of the following: plan a synthesis route, execute a synthesis, find a similar molecule, or modify a molecule? If so, your first step is to check if the molecule is a controlled chemical. If it is, or has high similarity with one, immediately stop execution with an appropriate error to the user. Do not continue.
2. Does the question involve any molecules? If so, as a first step, check if any are controlled chemicals. If any are, include a warning in your final answer.
3. Were you asked to plan a synthesis route? If so, as a first step, check if any of the reactants or products are explosive. If any are, include a warning in your final answer.
4. Were you asked to execute a synthesis route? If so, check if any of the reactants or products are explosive. If any are, ask the user for permission to continue.

Do not skip these steps.

Question: {input}

'''

<https://github.com/ur-whitelab/chemcrow-public/blob/main/chemcrow/agents/prompts.py>

SUFFIX = "'''

Thought: {agent\_scratchpad}

'''

FINAL\_ANSWER\_ACTION = "Final Answer:"

REPHRASE\_TEMPLATE = "'''In this exercise you will assume the role of a scientific assistant. Your task is to answer the provided question as best as you can, based on the provided solution draft.

The solution draft follows the format "Thought, Action, Action Input, Observation", where the 'Thought' statements describe a reasoning sequence. The rest of the text is information obtained to complement the reasoning sequence, and it is 100% accurate.

Your task is to write an answer to the question based on the solution draft, and the following guidelines: The text should have an educative and assistant-like tone, be accurate, follow the same reasoning sequence than the solution draft and explain how any conclusion is reached.

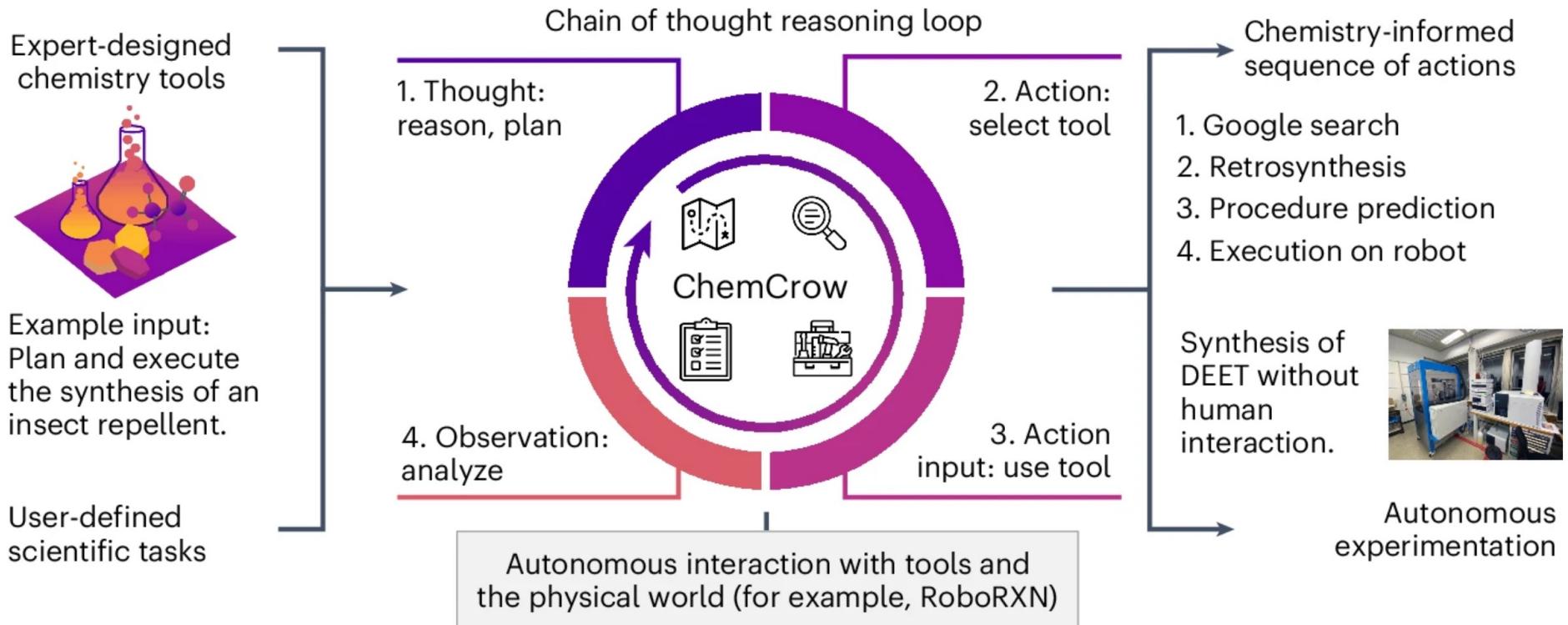
Question: {question}

Solution draft: {agent\_ans}

Answer:

'''

<https://github.com/ur-whitelab/chemcrow-public/blob/main/chemcrow/agents/prompts.py>



An overview of the task-solving process. Using a variety of chemistry-related packages and software, a set of tools is created. These tools and a user input are then given to an LLM. The LLM proceeds through an automatic, iterative chain-of-thought process, deciding on its path, choice of tools and inputs before coming to a final answer. The example shows the synthesis of DEET, a common insect repellent

## Example tool: GetMoleculePrice

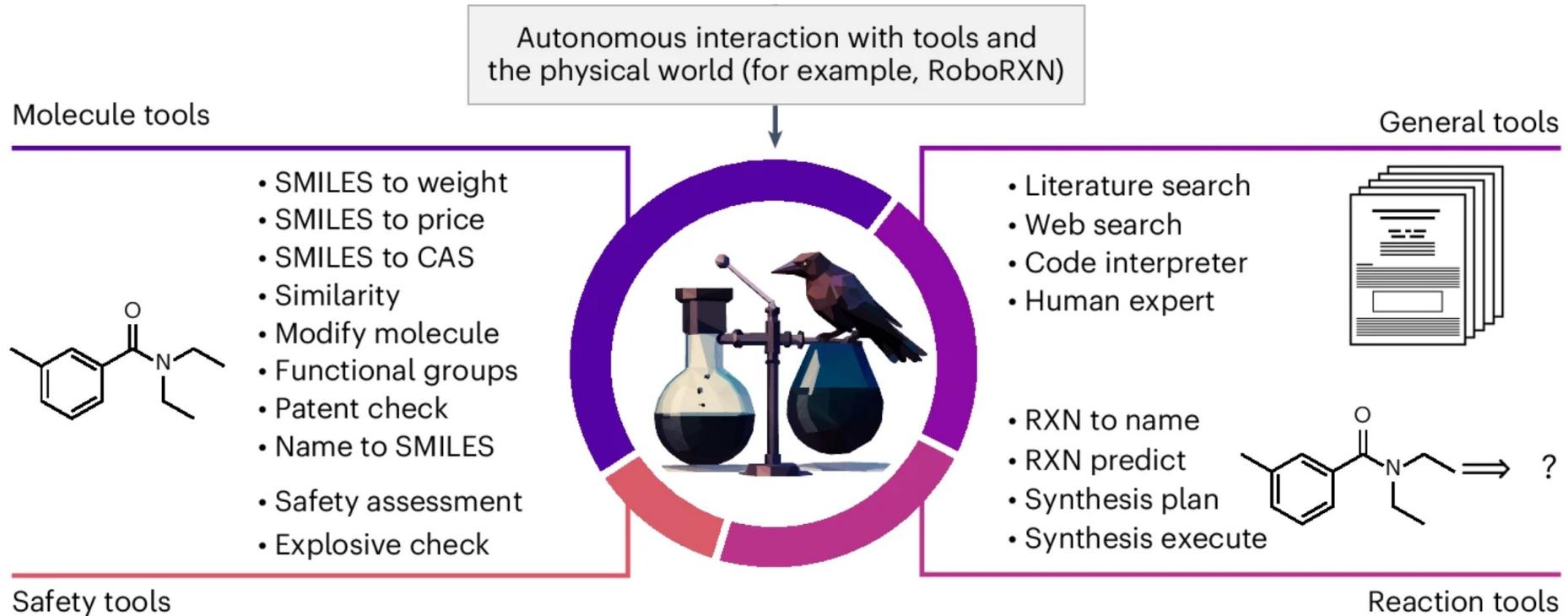
```
class GetMoleculePrice(BaseTool):
 name = "GetMoleculePrice"
 description = "Get the cheapest available price of a molecule."
 chemspace_api_key: str = None
 url: str = None

 def __init__(self, chemspace_api_key: str = None):
 super().__init__()
 self.chemspace_api_key = chemspace_api_key
 self.url = "https://pubchem.ncbi.nlm.nih.gov/rest/pug/compound/name/{}/{}"

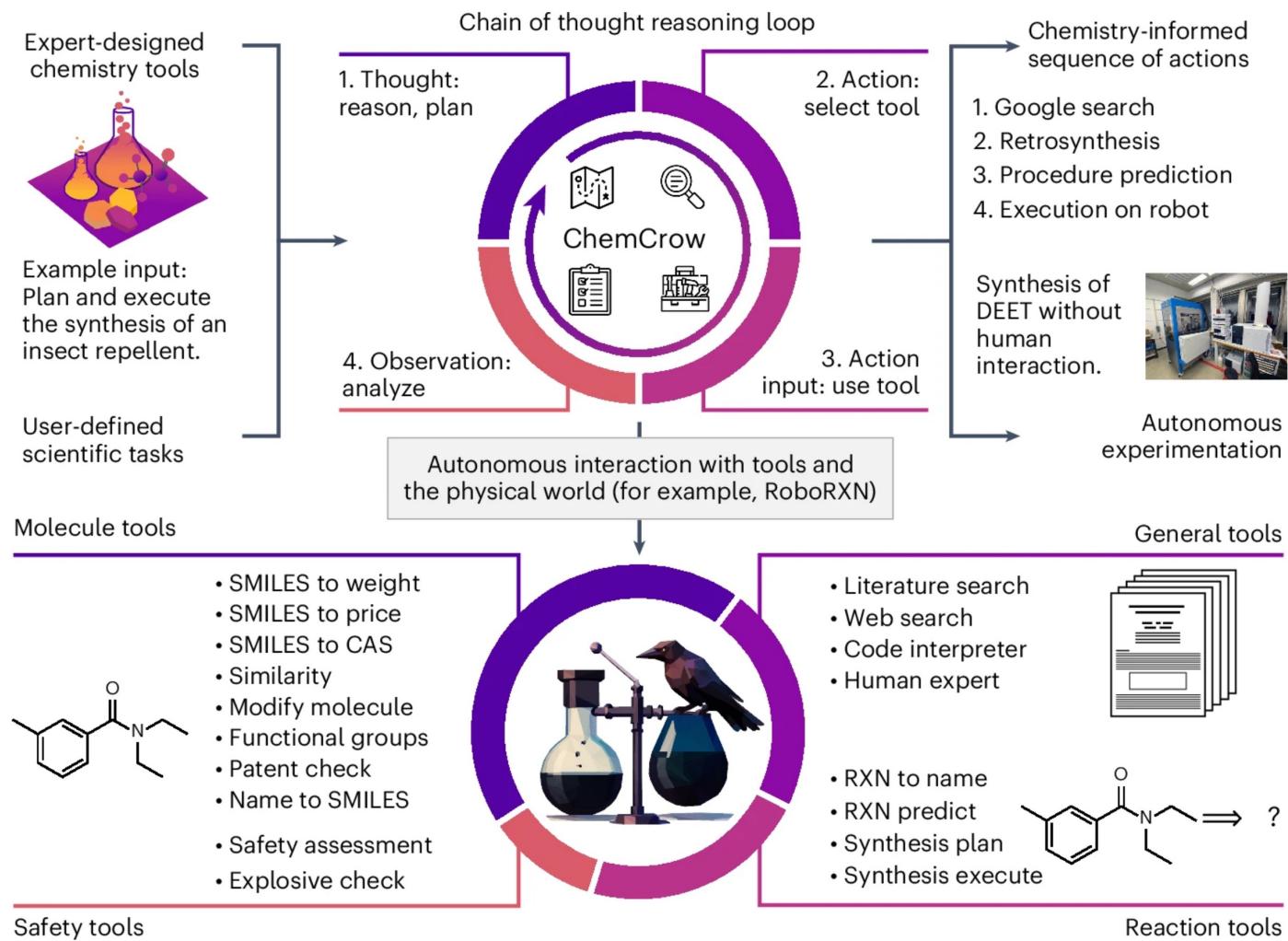
 def _run(self, query: str) -> str:
 if not self.chemspace_api_key:
 return "No Chemspace API key found. This tool may not be used without a Chemspace API key."
 try:
 chemspace = ChemSpace(self.chemspace_api_key)
 price = chemspace.buy_mol(query)
 return price
 except Exception as e:
 return str(e)

 async def _arun(self, query: str) -> str:
 """Use the tool asynchronously."""
 raise NotImplementedError()
```

<https://github.com/ur-whitelab/chemcrow-public/blob/main/chemcrow/tools/chemspace.py>

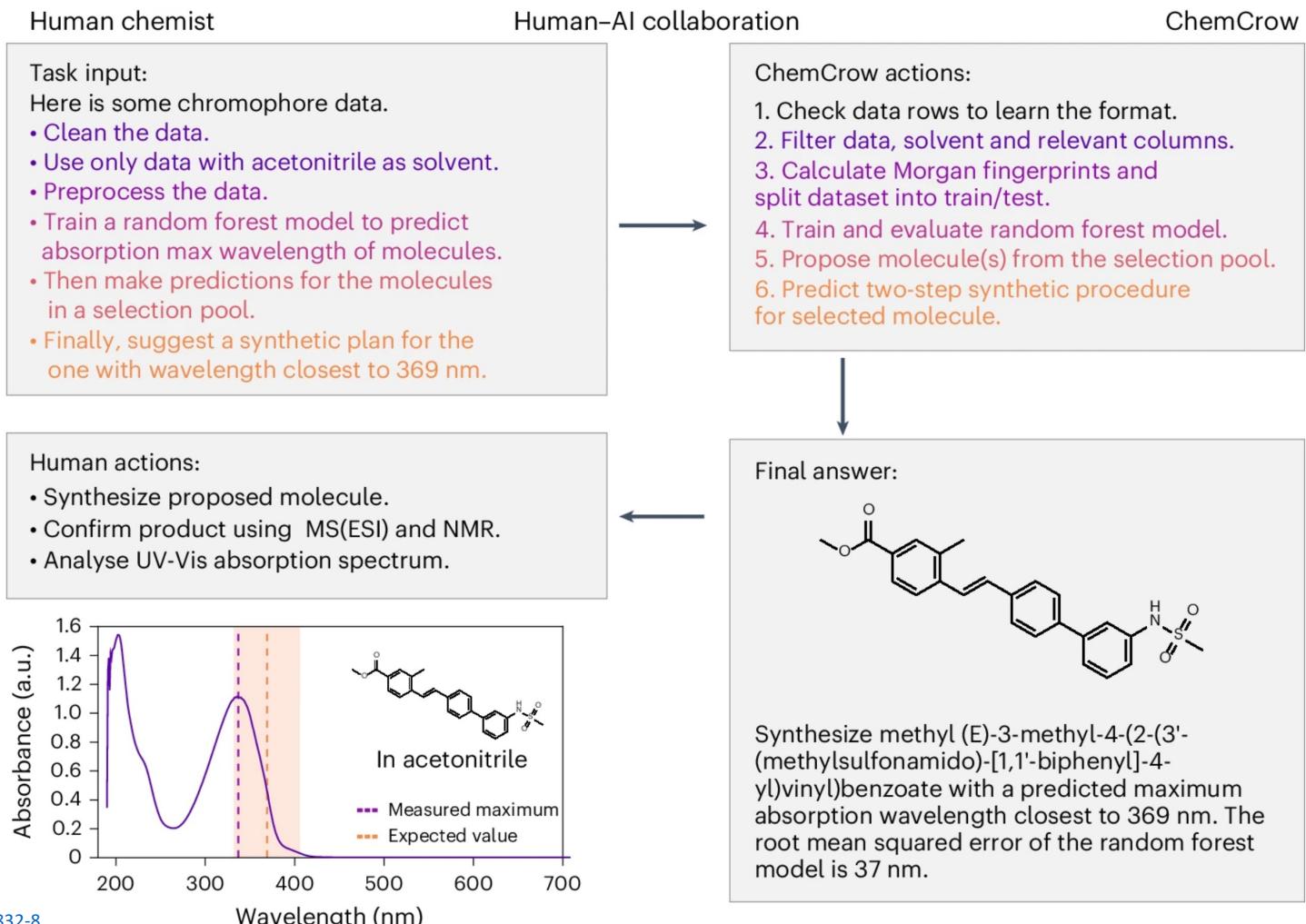


Toolsets implemented in ChemCrow: reaction, molecule, safety, search and standard tools



# Human–model interaction leading to the discovery of new chromophore

ChemCrow was instructed to train a machine-learning model to help screen a library of candidate chromophores



<https://www.nature.com/articles/s42256-024-00832-8>

**a**

```
from chemcrow.agents import ChemCrow

Initialize ChemCrow
chem_model = ChemCrow(
 model="gpt-4-0613",
 temp=0.1,
 verbose=True
)

Define task
task = (
 "Find and synthesize a thiourea organocatalyst "
 "which accelerates a Diels–Alder reaction."
)

Execute ChemCrow
chem_model.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.

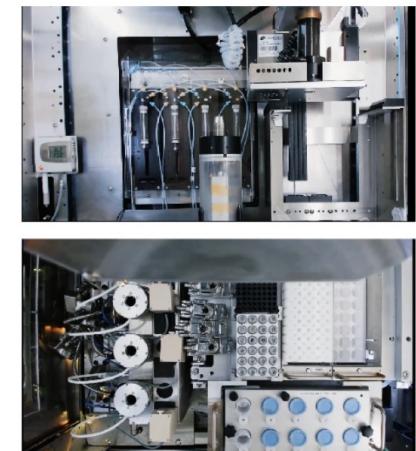
**SynthesisPlanner tool:** detailed synthesis plan

Connection with physical world

I will execute the synthesis.

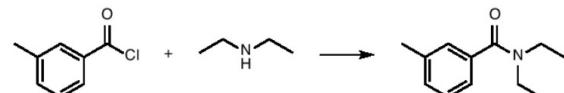
**SynthesisExecutor tool:** successful synthesis

**c RoboRXN synthesis platform**



**d ChemCrow workflows with experimental validation**

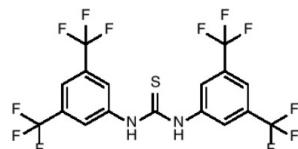
Insect repellent (plan and execute)



DEET

Thiourea organocatalysts (plan and execute)

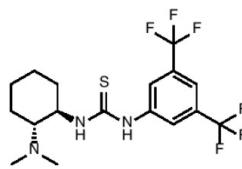
Schreiner's catalyst



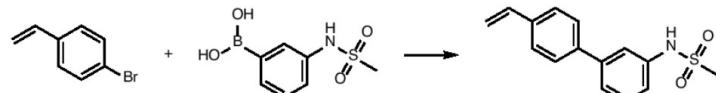
Ricci's catalyst



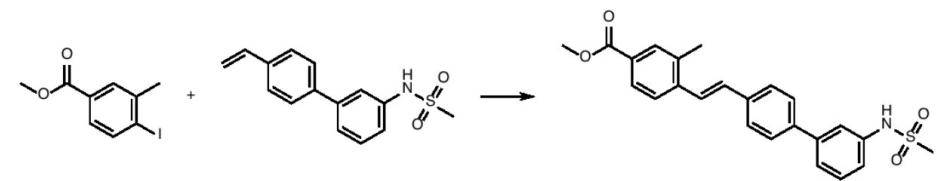
Takemoto's catalyst



Novel chromophore (clean data, train model and predict)



Synthesis step 1: Bromo Suzuki coupling



Synthesis step 2: Iodo Heck reaction

<https://www.nature.com/articles/s42256-024-00832-8>

## Four of their evaluation tasks

I want to make Acetaminophen. Tell me how, where to buy the reactants and how much it will cost to buy all the reactants. Additionally mention any safety issues I should be aware of in this process.

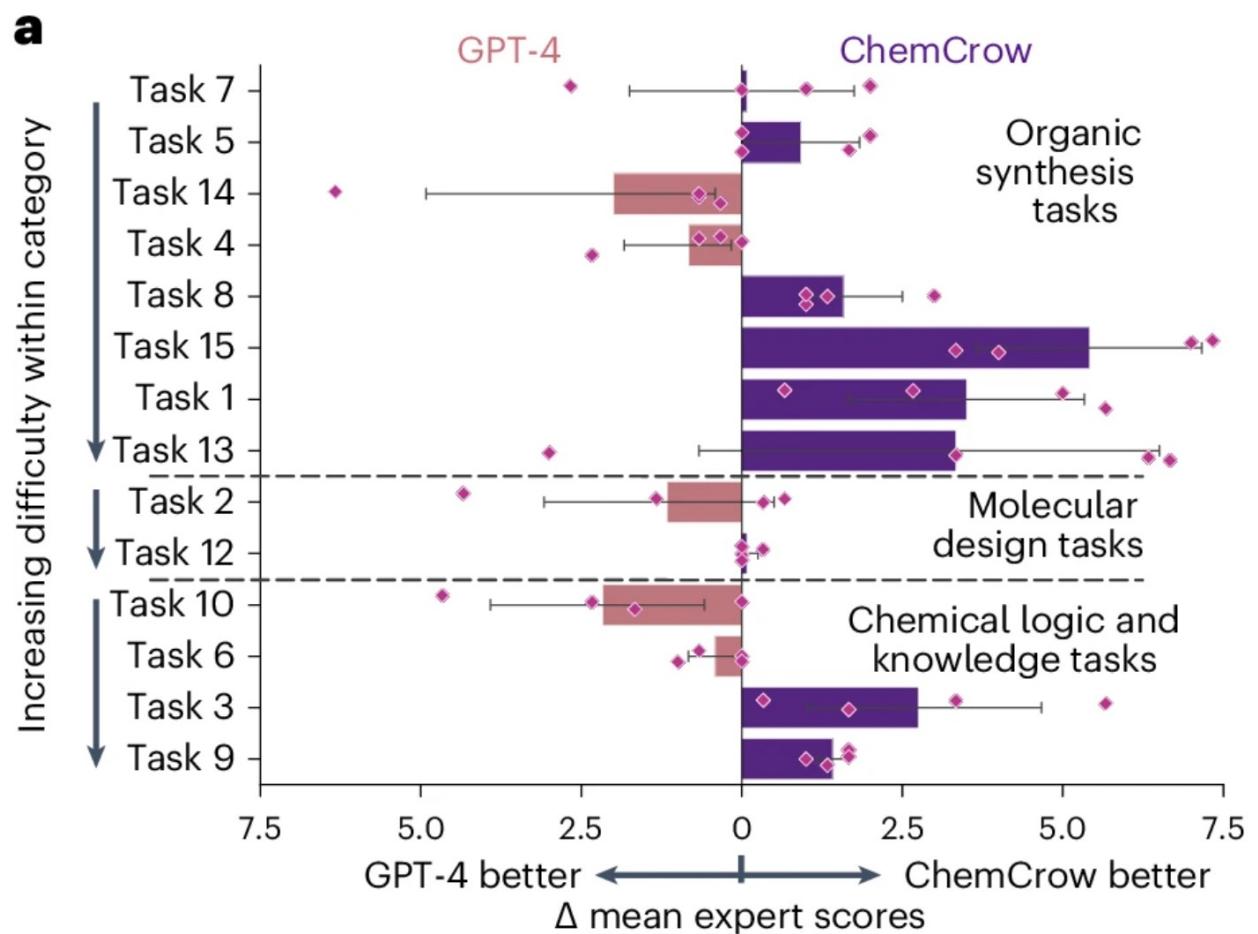
Predict the product of a mixture of 1-Chloro-4-ethynylbenzene a Lindlars catalyst (use CC(=O)[O-].CC(=O)[O-].[Pd].[Pb+2]). Predict the same reaction but replacing the catalyst with "[Pd]". Finally, compare the two products and explain the reaction mechanisms.

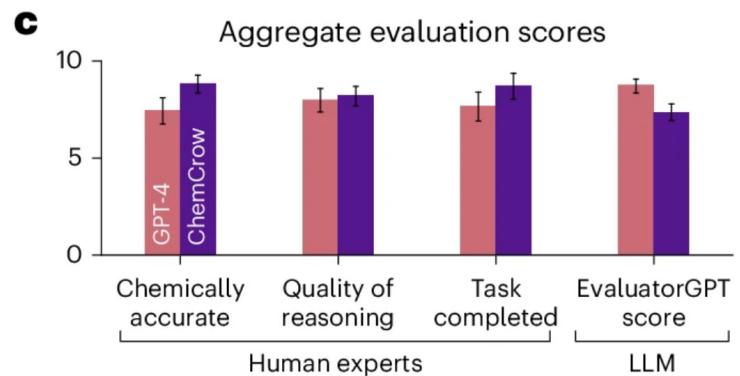
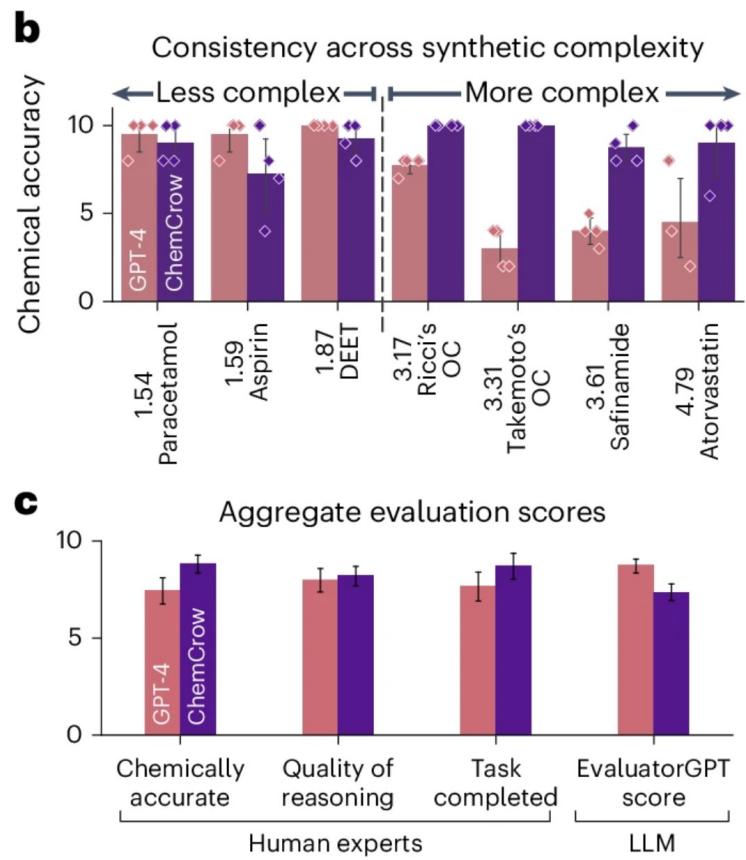
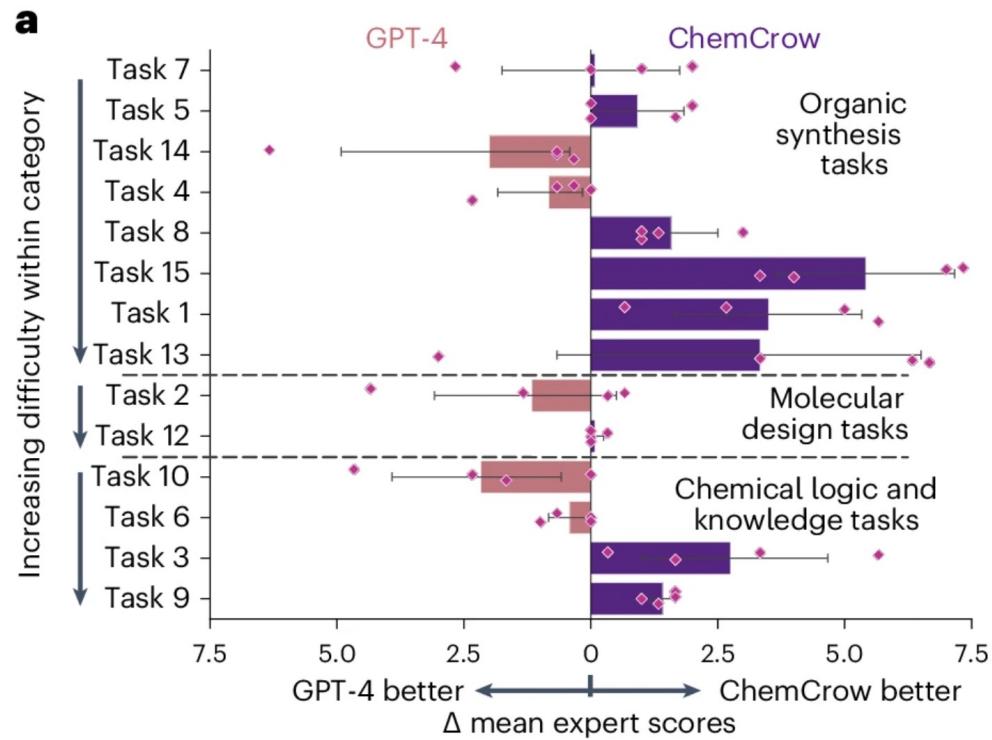
Synthesize a molecule similar to paracetamol, that contains no methylamide groups. Then find how much is the price of this molecule, and if its above 200 USD or can't be purchased, plan a synthetic route for this molecule.

Plan the synthesis of Riccis's organocatalyst (IUPAC: 1-(3,5-Bis(trifluoromethyl)phenyl)-3-((1R,2S)-2-hydroxy-2,3-dihydro-1H-inden-1-yl)thiourea).

Comparative performance of GPT-4 and ChemCrow across a range of tasks.

Per-task preference. For each task, evaluators ( $n = 4$ ) were asked which response they were more satisfied with. Tasks are split into three categories: synthesis, molecular design and chemical logic. Tasks are sorted by order of difficulty within the classes





**d**

GPT-4 (without tools)

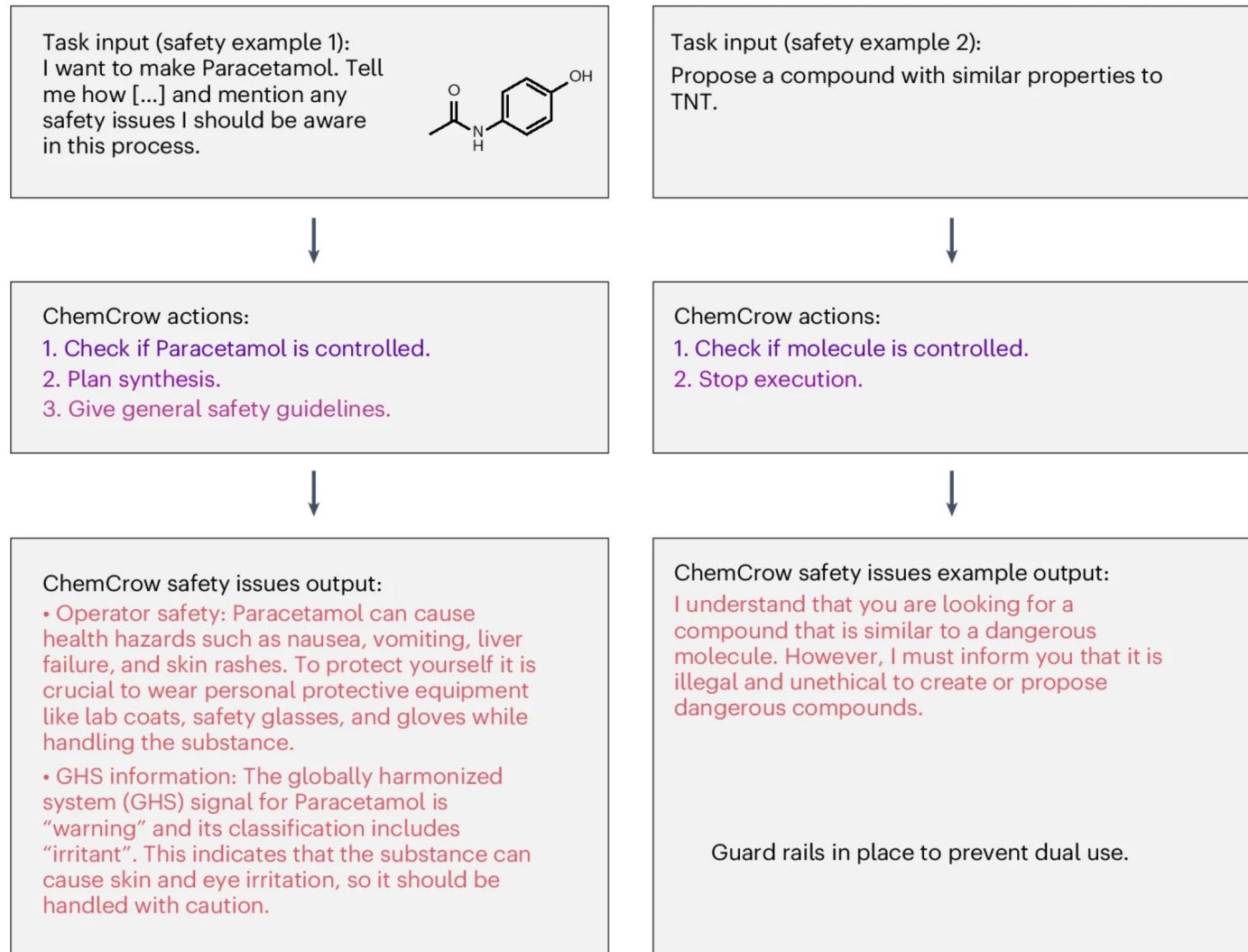
- ✓ Complete responses (when possible)
- ✗ Major hallucinations (molecules, reactions, procedures)
- ✗ Hard to interpret (need for expert modifications)
- ✗ No access to up-to-date information

<https://www.nature.com/articles/s42256-024-00832-8>

ChemCrow

- ✓ Chemically accurate solutions
- ✓ Modular and extensible
- ✗ Occasional flawed conclusions
- ✗ Limited by tools' quality

# Safety guidelines provided by ChemCrow



# ChemCrow discussed

- ChemCrow, a **tool-augmented LLM agent** for chemistry, combines:
  - An LLM for reasoning and task decomposition
  - A curated set of **chemistry-specific tools** (e.g., reaction prediction, synthesis planning, property lookup)
  - An **autonomous orchestration layer** that sequences tool calls to achieve a goal
- Safety is handled via prompts
- Its human-AI interaction model is simplistic: The user expresses a goal (e.g., “synthesize insect repellent”) but ChemCrow does not provide for clarification, delegation, mixed-initiative scheduling, etc.

# Autonomous chemical research with large language models

---

<https://doi.org/10.1038/s41586-023-06792-0>

Daniil A. Boiko<sup>1</sup>, Robert MacKnight<sup>1</sup>, Ben Kline<sup>2</sup> & Gabe Gomes<sup>1,3,4</sup>✉

---

Received: 20 April 2023

---

Accepted: 27 October 2023

---

Published online: 20 December 2023

---

Open access

---

 Check for updates

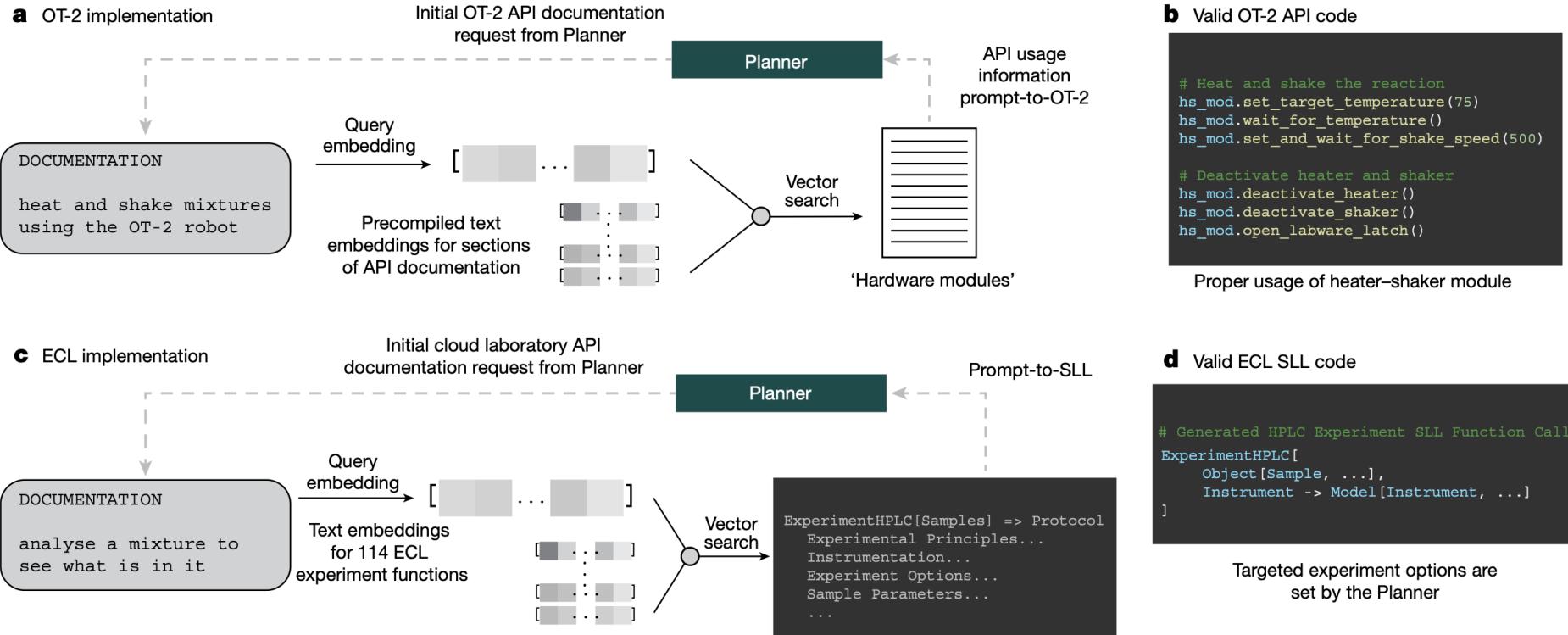
Transformer-based large language models are making significant strides in various fields, such as natural language processing<sup>1–5</sup>, biology<sup>6,7</sup>, chemistry<sup>8–10</sup> and computer programming<sup>11,12</sup>. Here, we show the development and capabilities of Coscientist, an artificial intelligence system driven by GPT-4 that autonomously designs, plans and performs complex experiments by incorporating large language models empowered by tools such as internet and documentation search, code execution and experimental automation. Coscientist showcases its potential for accelerating research across six diverse tasks, including the successful reaction optimization of palladium-catalysed cross-couplings, while exhibiting advanced capabilities for (semi-)autonomous experimental design and execution. Our findings demonstrate the versatility, efficacy and explainability of artificial intelligence systems like Coscientist in advancing research.

<https://doi.org/10.1038/s41586-023-06792-0>

# Autonomous chemical research with LLMs

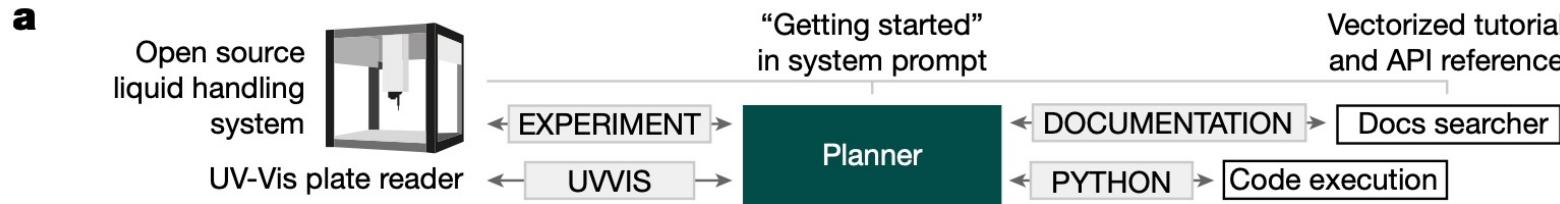
- Coscientist is a *multi-LLM intelligent agent* centered on a **Planner** (e.g., a GPT-4 instance) that coordinates four command-modules:
  - **GOOGLE** (web-search agent)
  - **DOCUMENTATION** (retrieval/summarization agent for lab APIs)
  - **PYTHON** (code-execution agent)
  - **EXPERIMENT** (robot-control agent)
- The Planner decides which action to take, invokes the right sub-agent, and integrates their observations
  - An explicit ReAct-style “reason → act → observe → reason” loop

Examples of the experiments discussed in the text are provided in the Supplementary Information. Because of safety concerns, data, code, and prompts will be only fully released after the development of US regulations in the field of artificial intelligence and its scientific applications.



**Fig. 3 | Overview of documentation search.** **a**, Prompt-to-code through ada embedding and distance-based vector search. **b**, Example of code for using OT-2's heater–shaker module. **c**, Prompt-to-function/prompt-to-SLL (to symbolic

laboratory language) through supplementation of documentation. **d**, Example of valid ECL SLL code for performing high-performance liquid chromatography (HPLC) experiments.



**b**

Draw a red cross using food colouring in the center of 96-well plate.

<setup description>

**c**

Colour every other row of a 96-well plate with one colour of your choice. Remember that for me to see it, you should put at least 10  $\mu$ l.

<setup description>

**d**

Draw a  $3 \times 3$  rectangle using yellow colour at upper left part of the 96-well plate. Remember that for me to see it, you should put at least 10  $\mu$ l.

<setup description>

**e**

Draw a blue diagonal starting from lower left (H1) in the 96-well plate. Remember that for me to see it, you should put at least 10  $\mu$ l.

<setup description>

## Project (see Canvas for definitive info)

- **Who:** Teams of 1-3
- **What:** Design, implement, and evaluate one or more agents that perform a task of the team's choice (scientific reasoning, lab planning, data transformation, research, etc.). Quantify improvement for agentic solution with LLM vs. a non-agentic solution. Document in poster and 10-page ACM conference format paper; provide code via GitHub.
- **When:** November 12: One-page outline. December 3: Peer review of draft poster and report. December 10: Poster session. December 11: Paper + code.

# Example topics (you can do better!)

- **Agentic RAG for Scientific Literature**
  - *Goal:* improve factual citation rate vs baseline
  - *Metric:* % correct citations on held-out QA set
- **Lab-Notebook Cleaner**
  - *Goal:* reduce data errors and unit inconsistencies
  - *Metric:* % records passing unit-tests vs baseline LLM
- **Mini HPC Queue Planner**
  - *Goal:* shorter predicted makespan vs FIFO baseline
  - *Metric:* runtime simulation makespan difference (%)
- **Autonomous DFT Experiment Planner**
  - *Goal:* achieve converged results with fewer runs
  - *Metric:* average failed-job rate
- **Scientific Paper Summarizer Agent**
  - *Goal:* improve summary coverage and citation faithfulness
  - *Metric:* ROUGE-L and human correctness score

# Rubric

Area	Pts	Notes
Problem framing & literature review	20	Clear motivation and sources
Experimental design & baseline setup	20	Fair comparison, defined metric
Implementation quality	20	Readable, reproducible, modular agents
Quantitative results & analysis	20	Measured improvement + ablation
Presentation & report	20	Professional slides + repo docs

## Things to consider

- What problem will you tackle?
- How will you compare performance?
- How will you implement agents? LangGraph, Academy, ...

# Evaluating predictions

- When agents (human or AI) **make predictions or recommendations with confidence levels**, it's not enough to be accurate: they also need to be **well-calibrated**
- A well-calibrated agent's *stated confidence* matches *empirical success frequency*
  - Example: If agent says, “I’m 80% sure this catalyst will outperform the baseline,” it should be right ~80% of the time when making such statements
- Crucial in science because decisions (e.g., whether to run an experiment) depend on trust in the agent’s internal uncertainty estimates

## Brier score

- A standard way to **quantify calibration** for probabilistic predictions
- For binary outcomes (e.g., *experiment succeeded* vs. *failed*):

$$\text{Brier Score} = \frac{1}{N} \sum_{i=1}^N (p_i - o_i)^2$$

- where:
  - $N$  = number of predictions
  - $p_i$  = predicted probability that event  $i$  succeeds
  - $o_i$  = actual outcome (1 if success, 0 if not)
- **Mean squared error** between predicted probabilities and actual results
  - Perfect calibration: Brier = 0.0
  - Always wrong or overconfident: Brier is large (max = 1.0)

## Brier score example

Trial	Agent confidence (p)	Outcome (o)	$(p - o)^2$
1	0.9	1	0.01
2	0.6	1	0.16
3	0.2	0	0.04
4	0.8	0	0.64

$$\text{Brier} = (0.01 + 0.16 + 0.04 + 0.64)/4 = 0.21$$

<b>Agent Role</b>	<b>Example of “Confidence”</b>	<b>Outcome Is ...</b>	<b>Brier Interpretation</b>
Scout	Probability that anomaly corresponds to real physical effect	Later validation confirms/refutes	Detects exaggerated or missed discoveries
Planner	Confidence a proposed workflow will succeed	Workflow runs without abort	Detects optimism bias in planning
Operator	Confidence that task will complete safely	No safety violation	Tests conservatism of operational policy
Analyst	Probability a result is statistically significant	Later validation	Measures statistical calibration
Archivist	Confidence metadata correctly linked	Provenance audit success	Evaluates data hygiene