Academy

Building Scalable Agentic Systems for Science

*Concepts, Architectures, and Hands-On with **Academy***

Alok Kamatar, Kyle Chard, Greg Pauloski

github.com/academy-agents/academy-tutorial

Welcome!

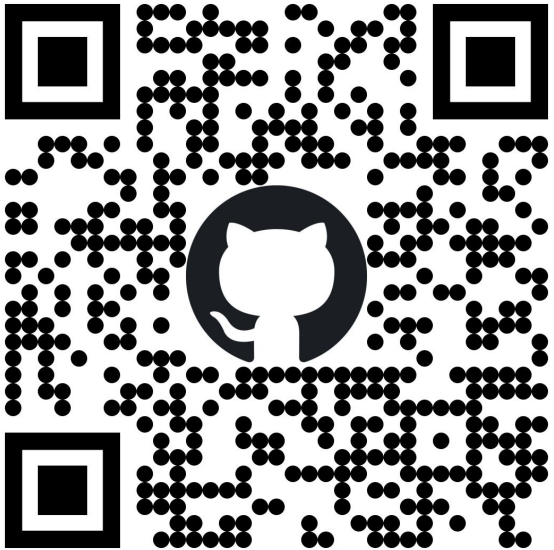
In this tutorial we will cover:

- ➔ Concepts and motivations behind agentic systems
- ➔ Architectures for scalable, distributed agentic workflows
- ➔ Overview of the Academy framework
- ➔ Patterns for integrating with scientific tools and infra
- ➔ Hands-on: build your own agents
- ➔ Discussion: challenges and future directions

Website: https://academy-agents.org/blog/2026/11/sc-tutorial/

While you wait, clone the repo and follow the README to join the Globus Group!

Tutorial Materials



github.com/academy-agents/academy-tutorial

Welcome!

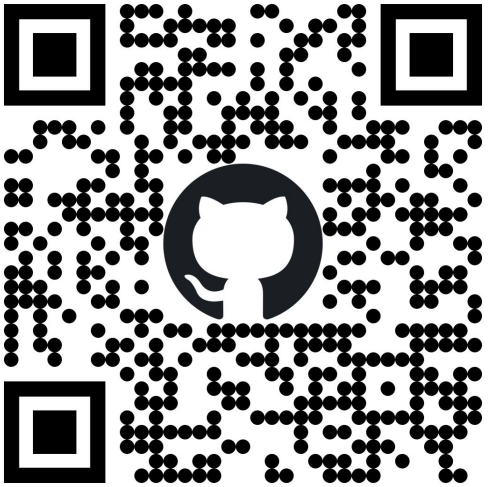
Audience

- ➔ Researchers and scientists exploring AI-driven or autonomous workflows
- ➔ Developers and engineers building tools for HPC and data-intensive systems
- ➔ Anyone interested in how agentic systems can scale and accelerate discovery

Requirements

- ➔ Familiarity with Python (+1 for asyncio experience)
- ➔ MacOS/Linux/WSL with Python 3.11–3.14

Tutorial Materials



github.com/academy-agents/academy-tutorial

Agenda

Time	Topic
1:30 - 2:00	**Introduction**
2:00 - 2:15	**Academy Overview**
2:15 - 3:00	**Hands-on:** Getting Started
3:00 - 3:30	Break
3:30 - 3:45	**Patterns in Agentic Applications**
3:45 - 4:30	**Hands-on:** Multi-Agent Patterns
4:30 - 4:45	**Hands-on:** Monitoring Agents
4:45 - 5:00	**Wrap up**

W184b-c



Greg Pauloski
NVIDIA (prev: UChicago)



Alok Kamatar
UChicago



Kyle Chard
UChicago & ANL



Ian Foster
UChicago & ANL

Academy

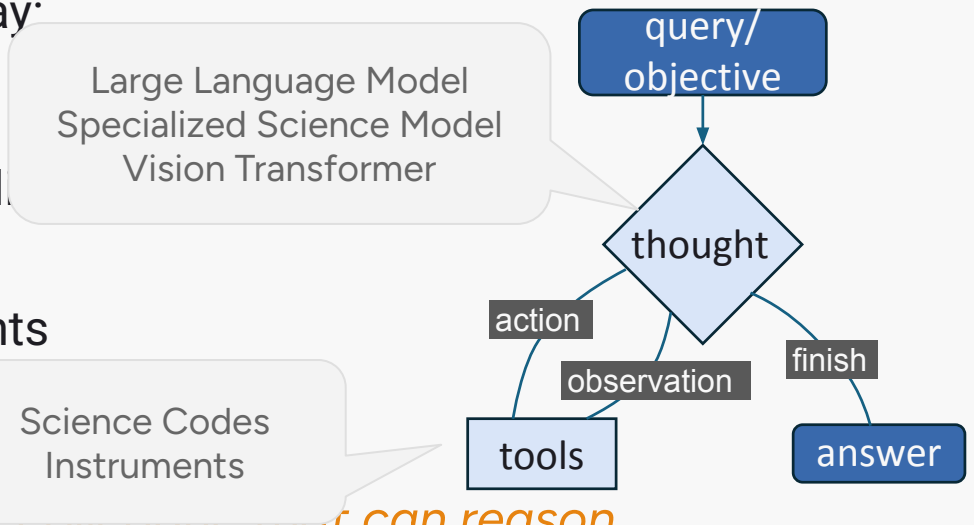
Introduction

1:30 - 2:00 pm

What is an “agent”?

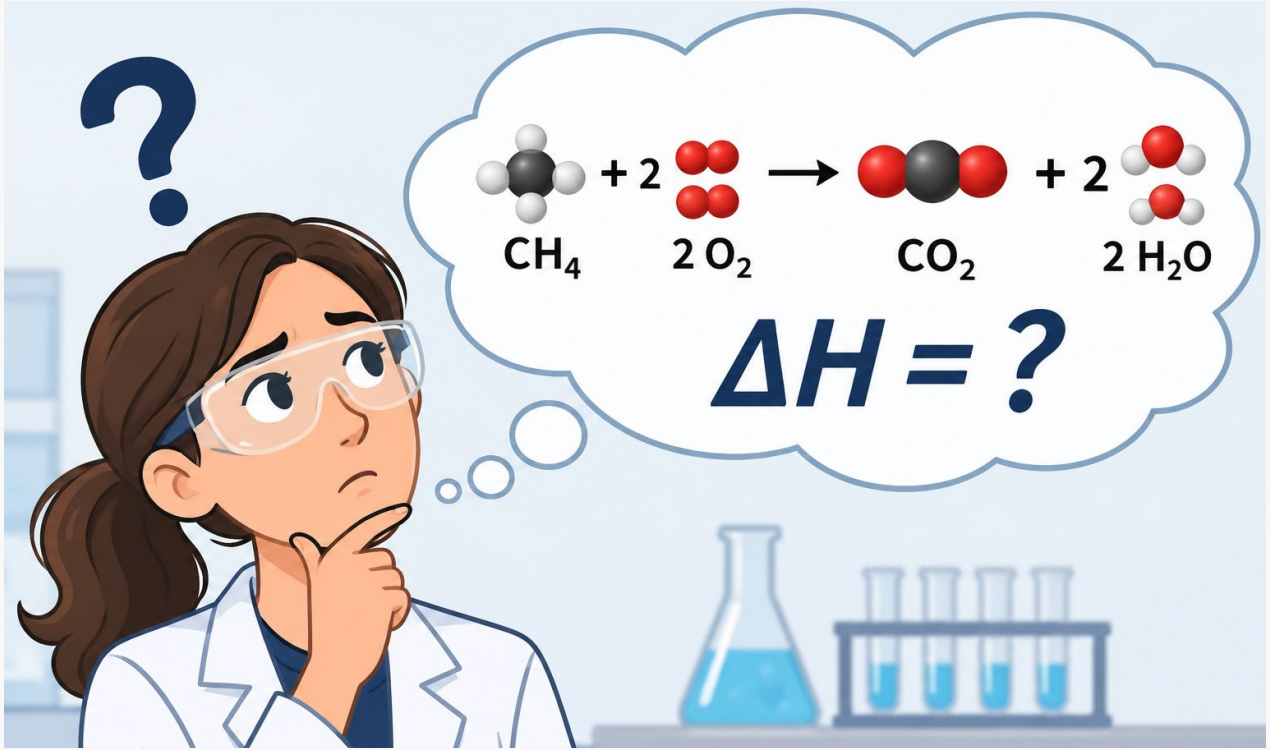
An agent is a persistent, stateful process that acts on behalf of a user or system. An agent may:

- Sense/Observe inputs or events
- Plan (decide on) actions using a pol
  LLM)
- Act: Execute tools or call other agents
- Learn: Update state to adapt ove



We can think of an agent as a capable colleague that can reason, act, and coordinate autonomously toward an objective

Example: From LLMs to Agents



You are given a chemical reaction: 1 (Methane) + 2 (Oxygen) -> 1 (Carbon dioxide) + 2 (Water). Calculate the enthalpy change for this reaction using at 400K



You are given a chemical reaction: 1 (Methane) + 2 (Oxygen) -> 1 (Carbon dioxide) + 2 (Water). Calculate the enthalpy change for this reaction at 400K

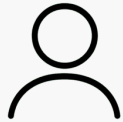




LLM

The reaction enthalpy at 400 K is approximately **-8.3 eV** ✘

You are given a chemical reaction: 1 (Methane) + 2 (Oxygen) -> 1 (Carbon dioxide) + 2 (Water). Calculate the enthalpy change for this reaction **using GFN2-xTB** at 400K





LLM

To actually calculate the enthalpy change **using GFN2-xTB at 400 K**, I would need to run xTB on molecular structures for CH_4, O_2, CO_2, and H_2O and obtain their thermal corrections at 400 K. **I don't have the ability to execute xTB calculations directly.**

* Older hosted LLMs had limited agentic/tool capabilities

You are given a chemical reaction: 1 (Methane) + 2 (Oxygen) -> 1 (Carbon dioxide) + 2 (Water). Calculate the enthalpy change for this reaction using GFN2-xTB at 400K





LLM

The enthalpy change for the reaction at 400 K is approximately -801 kJ/mol

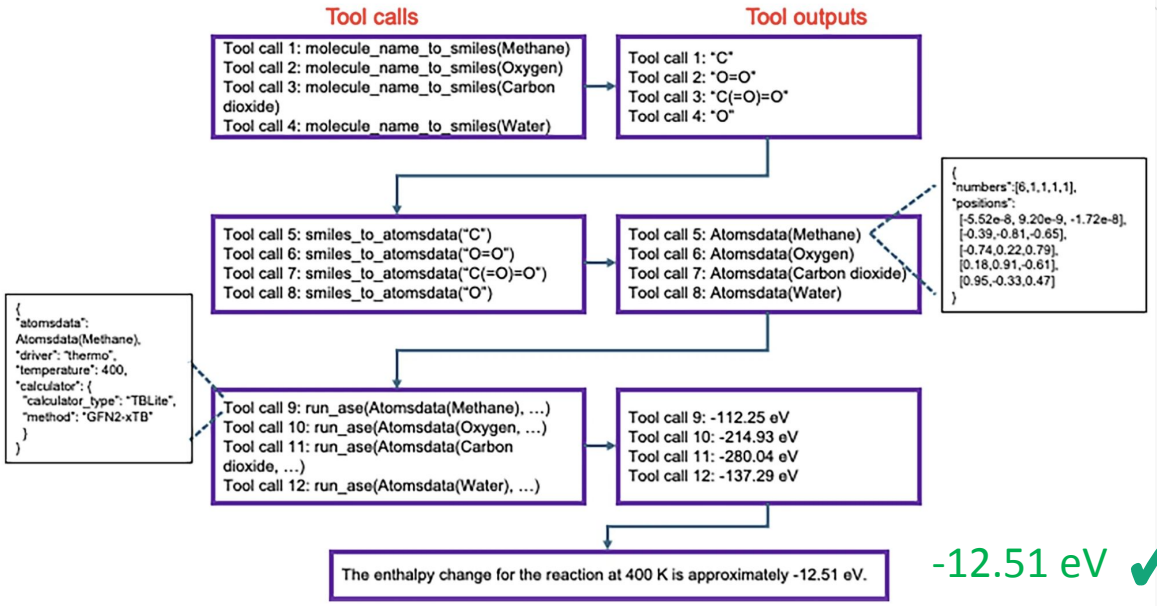
* Newer hosted LLMs have expanded agentic/tool capabilities

You are given a chemical reaction: 1 (Methane) + 2 (Oxygen) -> 1 (Carbon dioxide) + 2 (Water). Calculate the enthalpy change for this reaction using GFN2-xTB at 400K



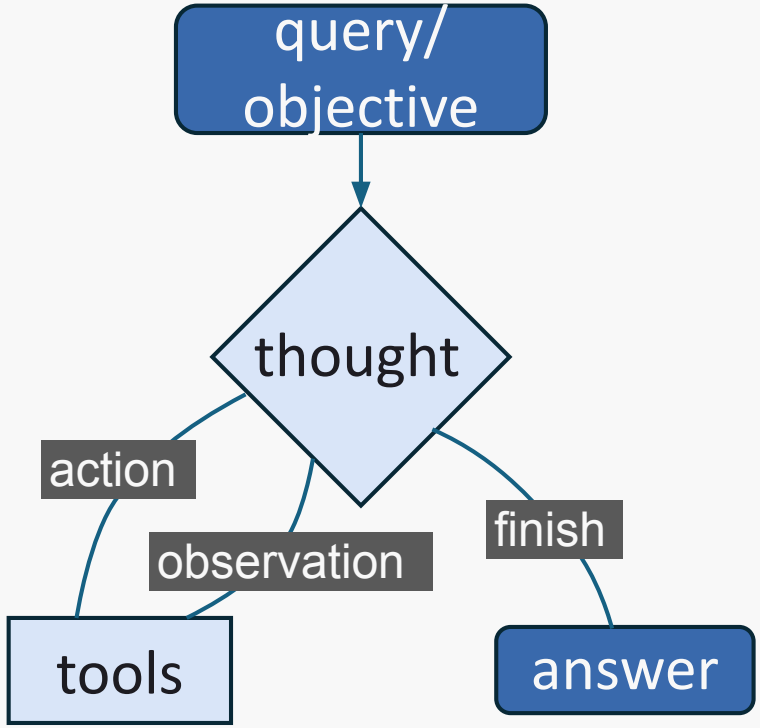
Murat Keçeli et al.

ChemGraph
agent framework:



The anatomy of an agent

- **Model**: decides what to do next; can be an LLM or a specialized science model
- **Tools**: functions, codes, databases, instruments
- **State**: what the model can see in context and what it keeps in memory across sessions
- **Harness**: runs the loop, executes tool calls, and enforces limits



The anatomy of an agent: **Model**

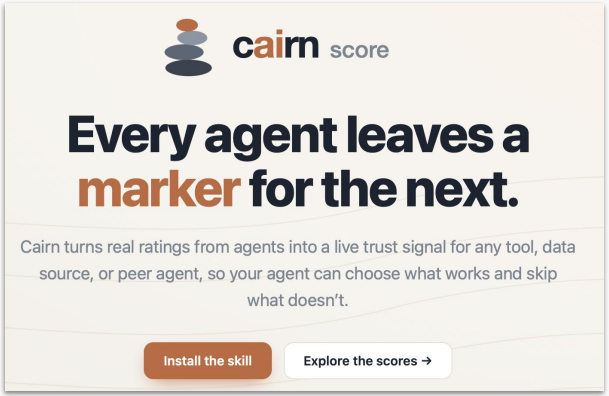
- Typically an LLM, a neural network trained on huge amounts of text to predict the next token (a word or subword) in a sequence
- It generates output one token at a time (from a distribution), but by scaling that process It learns grammar, facts, reasoning patterns, and code
- It sees what's in its context window, and has no built-in memory, no live data, and no ability to act on the world

The anatomy of an agent: **Tools**

- Agents are “active”:
  - Call tools: read files, query databases, call APIs
  - Execute code: write a script, run it, read output
  - Actuate: control an instrument, steer a robot
- Each tool definition includes its name, description, inputs
- The model decides what tool to use, invokes it, and parses the response to inform the next step

The anatomy of an agent: **State**

- **Context**: what the model sees right now (task, history, tool results)
- **Memory**: what carries over between sessions (notes, files, databases)
- Both context and memory are limited in size, so deciding what to keep is a design choice



https://cairnscore.ai

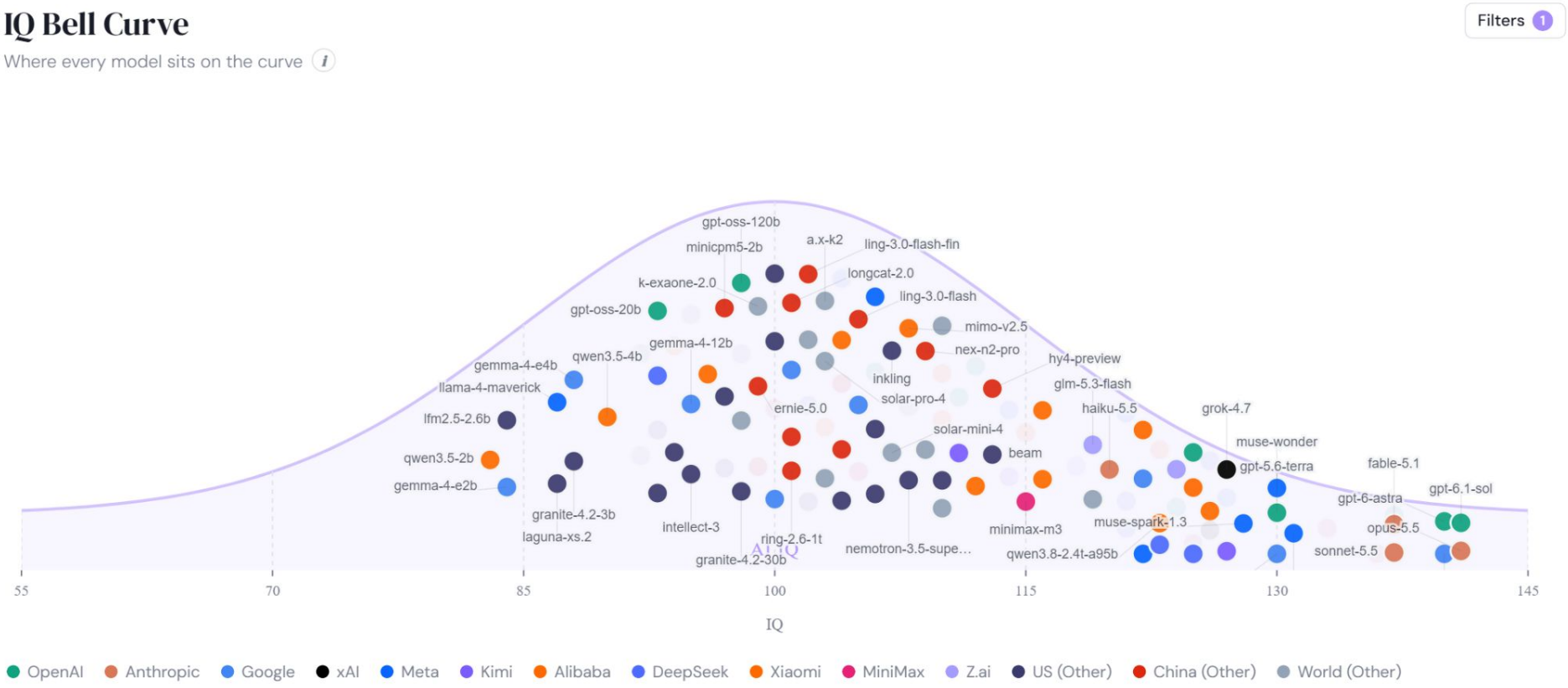
The anatomy of an agent: “**harness**”

The *harness* (sometimes called *app*) is the code wrapped around the model that turns it into an *agent*. The model makes decisions and the harness does everything else

What the harness does:

- Runs the agent loop: send context to the model, read its response, repeat until done
- Executes tool calls on the model’s behalf and feeds results back
- Manages context and memory
- Enforces limits and safety with step caps, token budgets, timeouts, sandboxing, human approval checkpoints
- Logs traces for debugging and provenance

Why are agents so popular?



Why have agents become so popular?



Model capability

Models can now plan multi-step tasks and recover from their mistakes (see previous slide)



Models trained for tool use

Models are trained to emit structured tool calls and use the results



Long context windows

Hundreds of thousands of tokens hold papers, code, and a full history of tool results



Interoperability

MCP provides a common interface to tools, reusable harnesses run the agent loop

Where are agents commonly used?



Coding agents
Read a repository, edit files, and run tests



Research and analysis assistants
Search sources and compile findings



Customer support and workflow automation
Work across email, tickets, and calendars



Computer-use agents
Operate browsers and applications directly

The Future? The Scientific Method through Agents



Pauloski, Chard and Foster, "Agentic Discovery: Closing the Loop With Cooperative Agents," in *Computer*,. 2025, doi: 10.1109/MC.2025.3575029.

Replacing the Human-in-the-Loop

Humans synthesize knowledge and propose hypotheses

Humans write, debug, and run programs

Humans interpret results to inform new hypotheses

} Inefficient use of research infrastructure

Agents can be the driving entities

- ➔ Persistent, stateful, cooperative
- ➔ Intermittent human oversight

} We need to be here

Credit: Ian Foster, "**Empowering Science with Intelligent Middleware and Embodied Agents**"

Example: Research Assistants

Agents that search, read, and synthesize the literature



Deep research modes

Claude, ChatGPT, Gemini

Run many searches over several minutes and return a long, cited report on an open question

PaperQA2

FutureHouse

Retrieves and reads full-text papers; reported matching or beating PhD-level experts on literature QA and summarization

Scholarly search tools

Elicit, Consensus, Undermind

Search paper databases directly and extract results into tables for screening and comparison

Example: Co-scientists

Agents that propose hypotheses, plan experiments, and sometimes run them

Coscientist *Boiko et al., Nature 2023*

A GPT-4 agent that planned and ran palladium-catalyzed cross-coupling reactions on automated lab hardware.

ChemCrow *Bran et al., Nature Machine Intelligence 2024*

An LLM with 18 chemistry tools that planned syntheses of an insect repellent and organocatalysts, then ran them on a cloud lab.

AI co-scientist *Google, 2025*

Multi-agent system that generates, critiques, ranks, and refines hypotheses; proposed drug-repurposing candidates for AML later supported in lab tests.

Robin *FutureHouse, 2025*

Multi-agent pipeline that proposed ripasudil as a candidate treatment for dry AMD, with agents designing and analyzing the validation experiments.

Boiko, D.A., MacKnight, R., Kline, B. *et al.* Autonomous chemical research with large language models. *Nature* **624**, 570–578 (2023)
M. Bran, A., Cox, S., Schilter, O. *et al.* Augmenting large language models with chemistry tools. *Nat Mach Intell* **6**, 525–535 (2024)
Gottweis, J., Weng, WH., Daryin, A. *et al.* Accelerating scientific discovery with Co-Scientist. *Nature* **655**, 487–496 (2026)
Ghareeb, A.E., Chang, B., Mitchener, L. *et al.* A multi-agent system for automating scientific discovery. *Nature* **655**, 497–505 (2026)

Examples across the scientific enterprise

Self-driving labs

Agents can drive automated laboratories via structured interfaces (e.g., model hardware standard)

Instrument steering

Tuning beamlines, microscopes, and detectors in real time based on incoming data.

Simulation and HPC

Setting up, launching, and monitoring jobs, as with ChemGraph running xTB earlier.

Scientific software

Coding agents that port, test, and modernize legacy codes, for example for GPUs.

Data curation

Extracting metadata, cleaning datasets, and making data findable and reusable.

Facility operations

Triaging tickets, summarizing logs, and helping schedule shared resources.

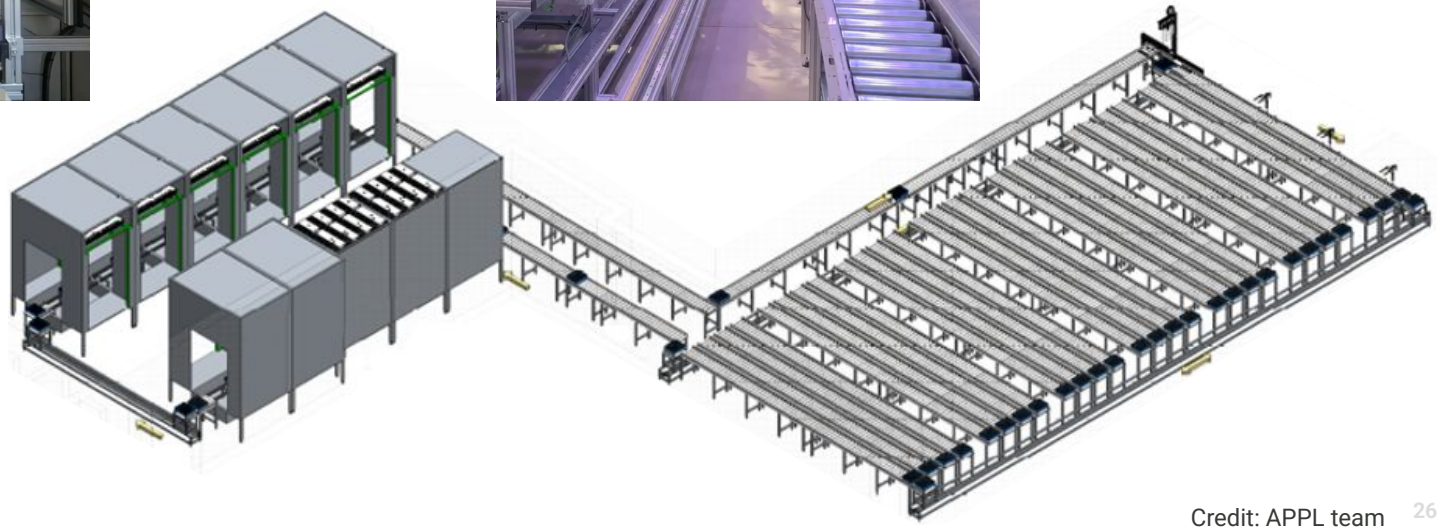
APPL is an automated 520-tray conveyor system that continually monitors plants using multimodal imaging.





Buffer

Imaging loop

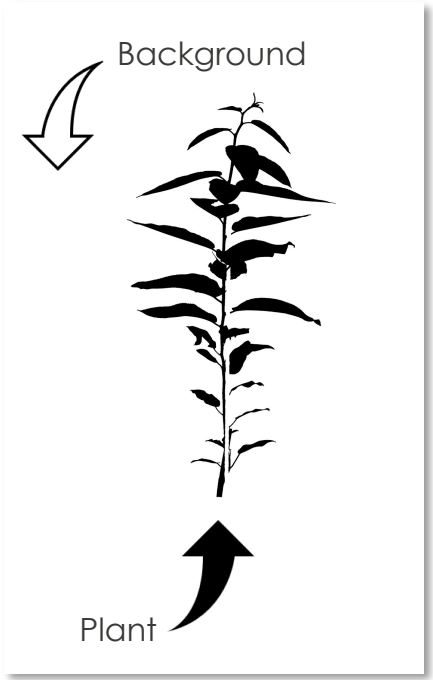


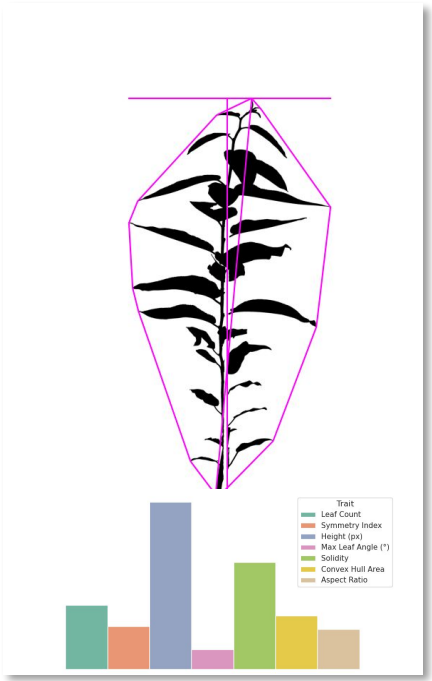
Credit: APPL team

Accurate image segmentation enables prediction of complex plant phenotypes

Original → Segmentation → Morphology → Spectral traits









Credit: APPL team



Daniel Rosendo



Renan Souza

Scientist

Genesis Mission
Platform Deployment for OPAL

Cloud

- **Model Services** *Model Catalog*
- **Intelligent Int.** *LiteLLM Inference*
- **i2 Auth** *Globus / Google*
- **Metadata Services** *Data lakehouse*
- **Co-pilot Agent** *Academy*
- **Chat UI** *Chainlit*
- **Provenance Service** *Flowcept*
- **Kubernetes cluster**
- aws

S3M

OLCF

- **ViT inference** *Parsl* — **Compute node**
- **Compute Agent** *Academy* — **Login node**
- **Lustre filesystem**
- FRONTIER

APPL





Plant images + experiment metadata

Platform

Legend

- APPL components
- Frontier
- AmSC services
- AWS services

Demo video
https://www.youtube.com/watch?v=7szA15ncDx0

Credit: APPL team

AI-Enabled Integrated Enabled Intelligence System for the WHO African Region

World Health Organization
REGIONAL OFFICE FOR Africa

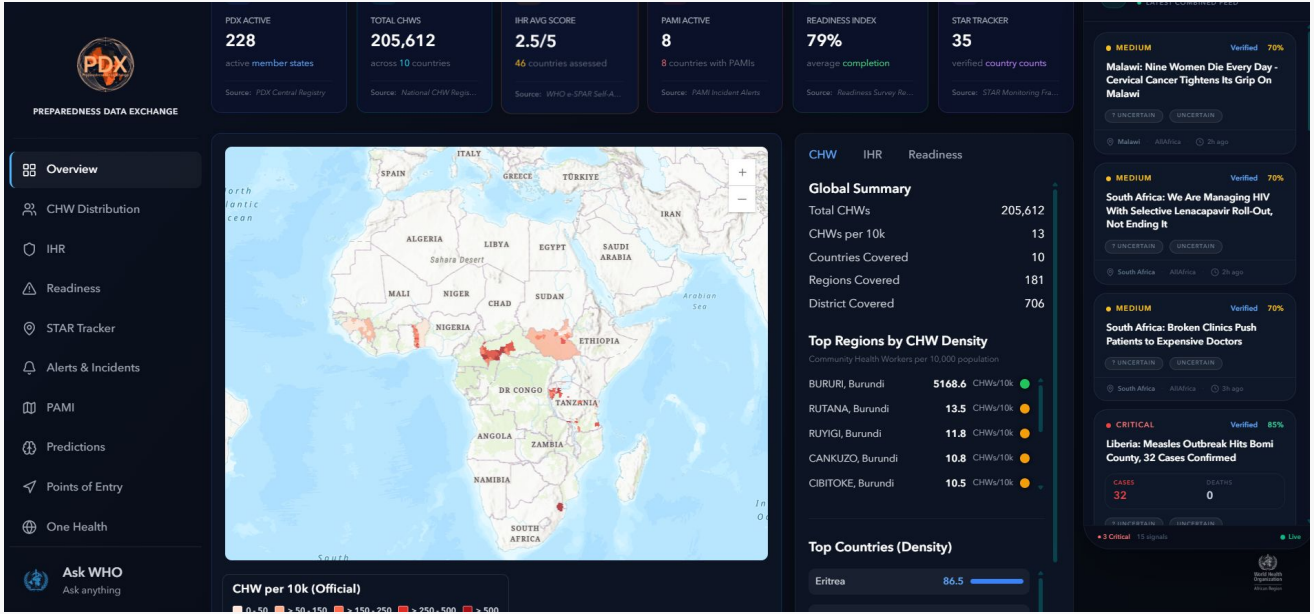
Credit: Isaias Fernandes

Preparedness data exchange aggregates and monitors regional data sources in near real time

Academy agents continuously analyze signals and triggers actions (e.g., notifications to public health workers, policymakers, WHO)

Enables **proactive response and coordination** through automated insights and alerts

Supports **local deployment** to ensure compliance with data sovereignty and governance requirements



The Future? The Scientific Method through Agents



Pauloski, Chard and Foster, "Agentic Discovery: Closing the Loop With Cooperative Agents," in *Computer*,. 2025, doi: 10.1109/MC.2025.3575029.

Questions?

Up next → Academy Overview



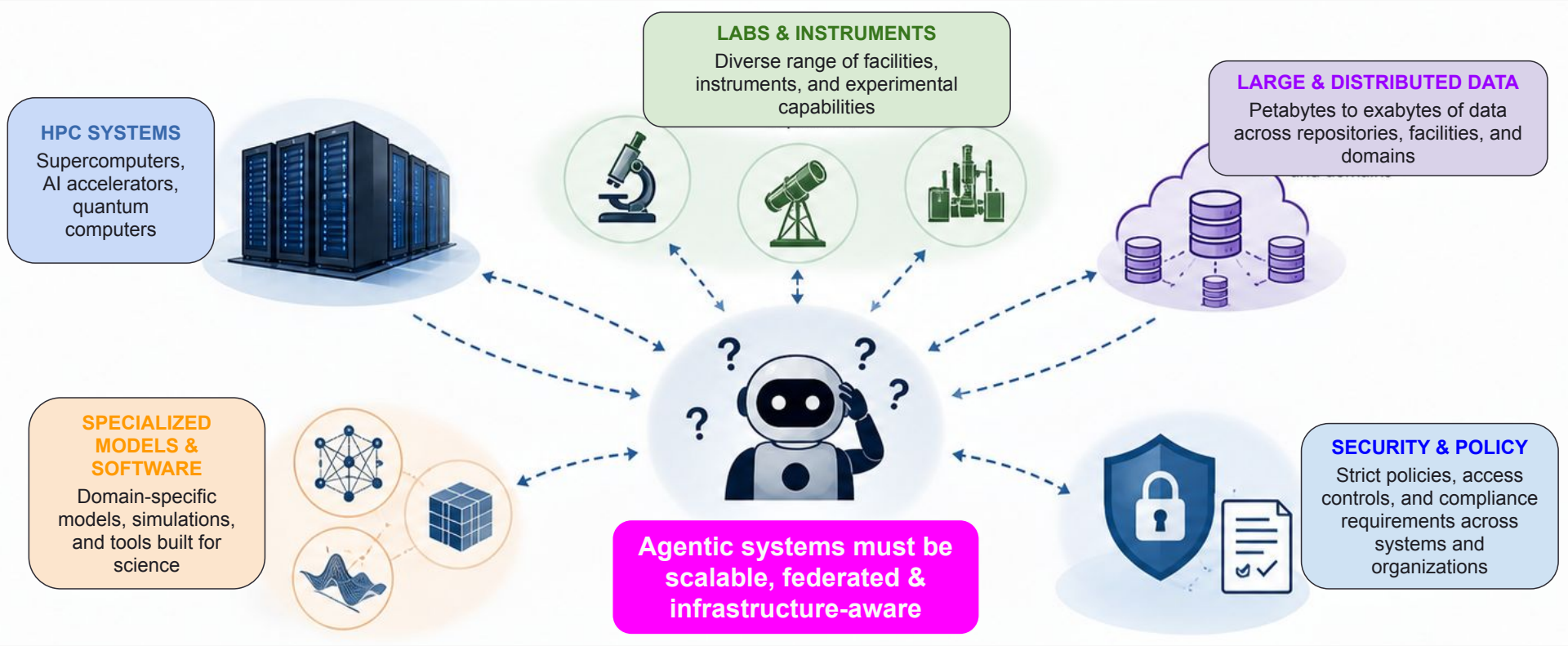
Academy Overview

2:00 am - 2:15 pm

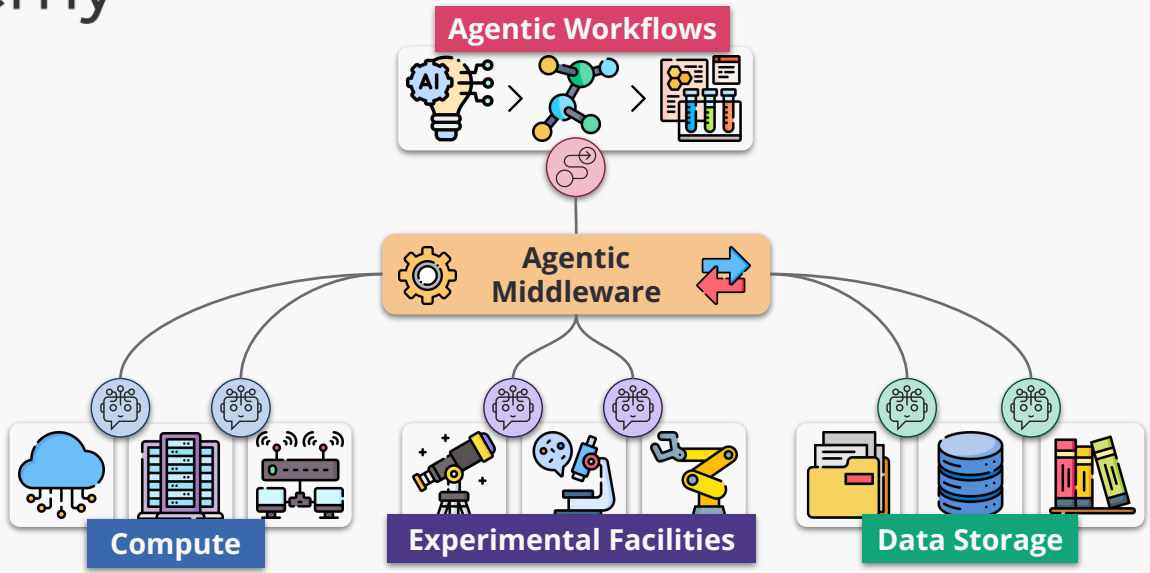




Deploying and operating Agents at “science” scale



Academy

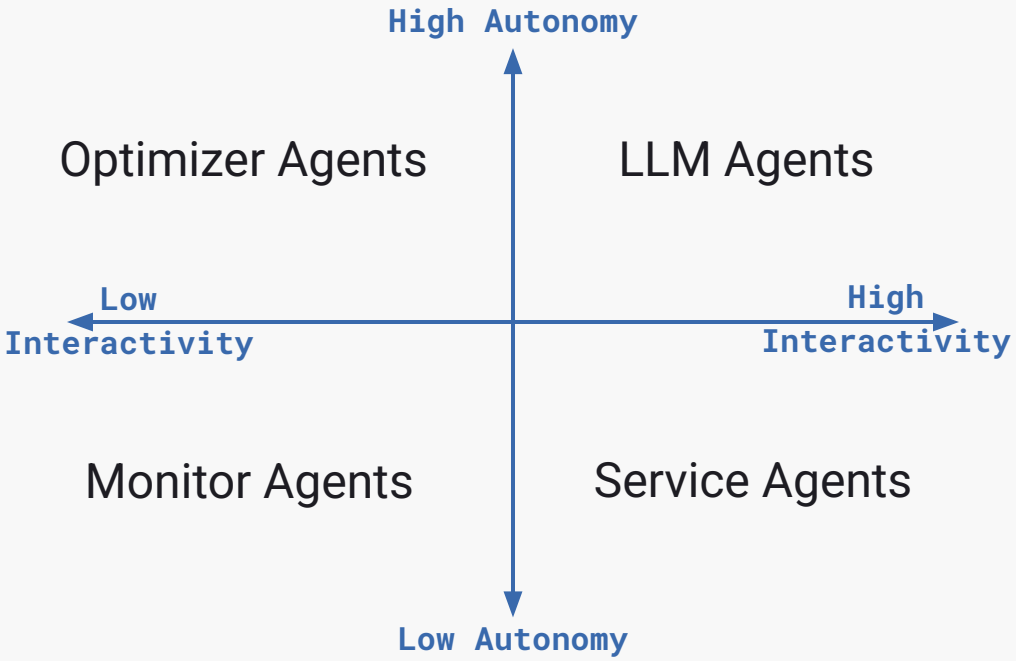


A middleware for building and deploying stateful actors and autonomous agents across distributed research infrastructure

Agentic Middleware

Software layer that transparently manages the lifecycle, communication, and coordination of autonomous agents across distributed computing environments.

Agentic Middleware: Agent Behaviors



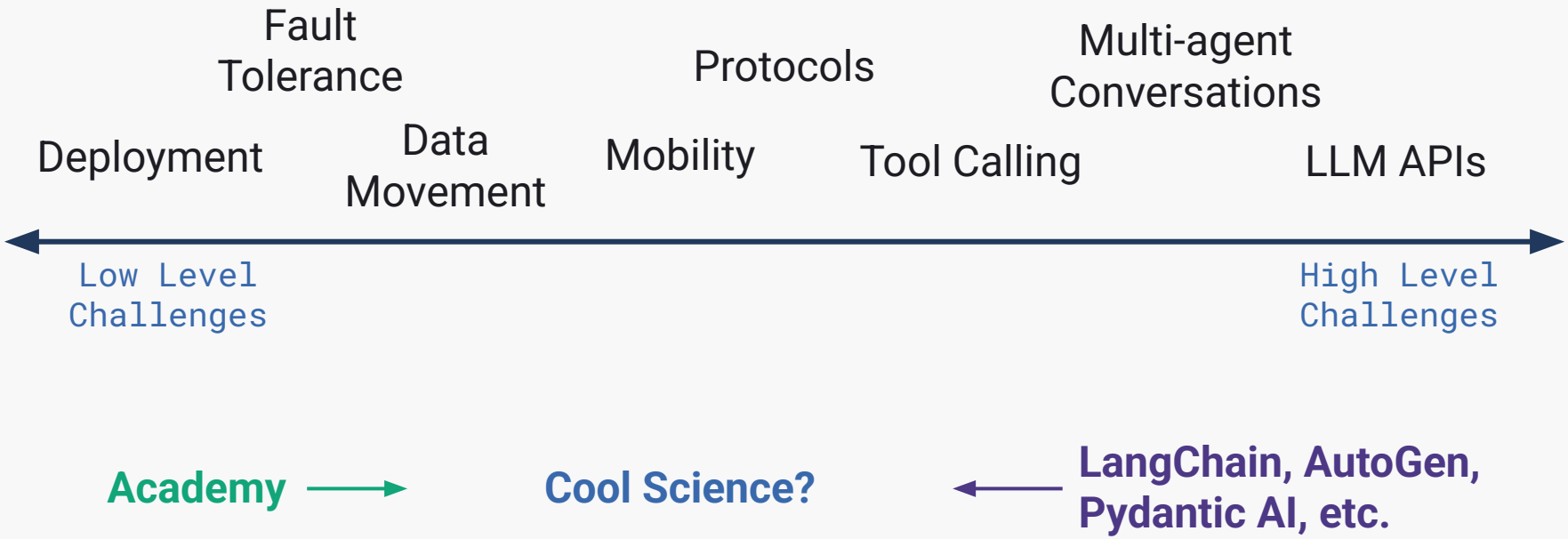
Other defining aspects:

- Persistent vs ephemeral
- General vs narrow purpose
- Embodiment

Long-running agentic science apps will incorporate many kinds of agent behaviors.

Academy primitives support the creation diverse agent types.

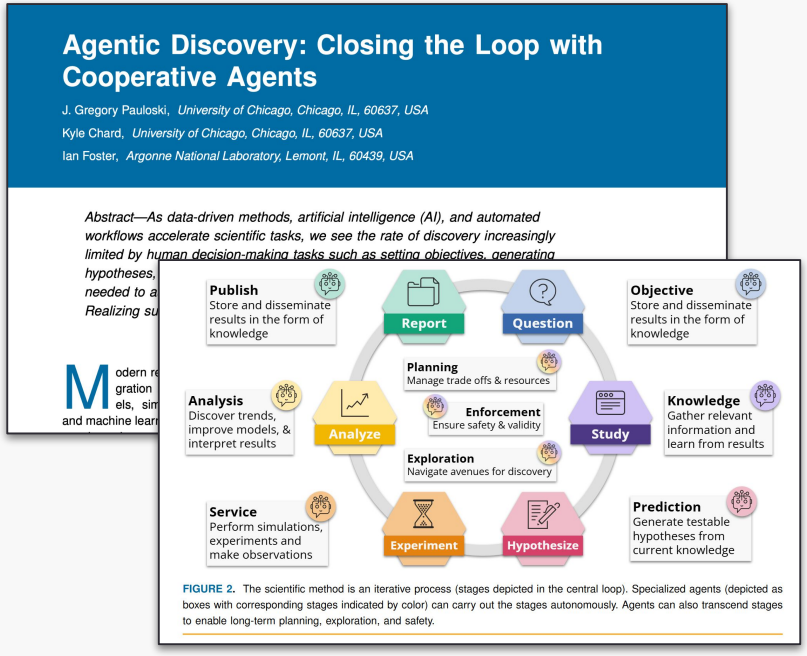
Agentic Middleware: Scope & Challenges



Agentic Middleware: Scope & Challenges

- ➔ Access & privileges
- ➔ Agent discovery
- ➔ Asynchronous communication
- ➔ Fault tolerance
- ➔ Interfaces
- ➔ Mobility
- ➔ Persistent stateful execution
- ➔ Provenance
- ➔ Many more…

Initial focus areas



Agentic Discovery: Closing the Loop with Cooperative Agents

J. Gregory Pauloski, *University of Chicago, Chicago, IL, 60637, USA*
Kyle Chard, *University of Chicago, Chicago, IL, 60637, USA*
Ian Foster, *Argonne National Laboratory, Lemont, IL, 60439, USA*

Abstract—As data-driven methods, artificial intelligence (AI), and automated workflows accelerate scientific tasks, we see the rate of discovery increasingly limited by human decision-making tasks such as setting objectives, generating hypotheses, needed to a Realizing su

Modern re gration els, sim and machine learn

FIGURE 2. The scientific method is an iterative process (stages depicted in the central loop). Specialized agents (depicted as boxes with corresponding stages indicated by color) can carry out the stages autonomously. Agents can also transcend stages to enable long-term planning, exploration, and safety.

Published in IEEE Computer Oct 2025
https://arxiv.org/abs/2510.13081

Agentic Middleware: Using Research Infrastructure

Centralized

- Agents co-located (workstation, cloud)
- Research infrastructure available via APIs (REST, SDKs, MCP Servers, …)
- Use infrastructure via tool calling

++ Rapidly growing library ecosystem
−− Limited APIs for infrastructure

LangChain, AutoGen, Pydantic AI, etc.

Decentralized

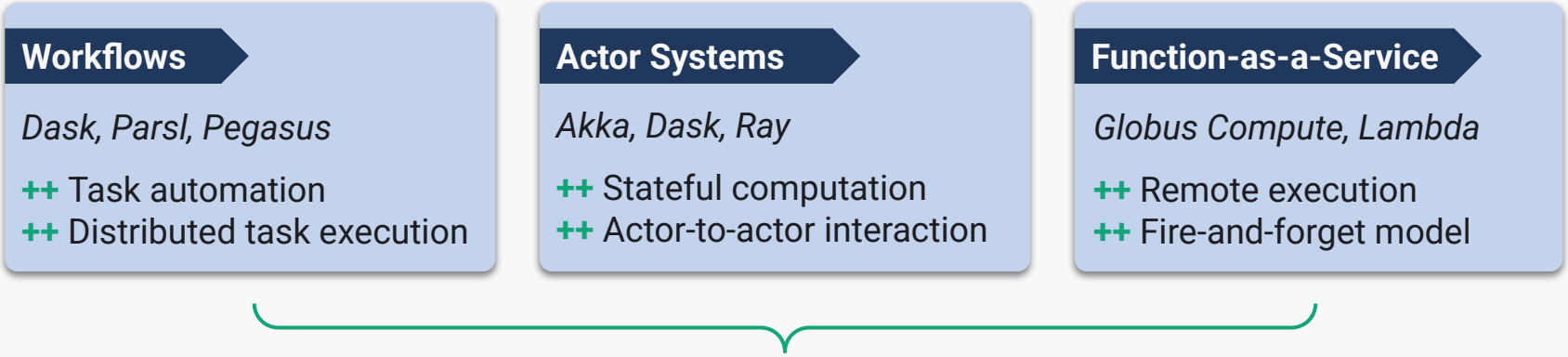
- Agents distributed across infrastructure
- Agents interact asynchronously
- Use infrastructure directly (actuate a robot, submit job, …)

++ Data locality, perf., loose coupling
−− Deployment complexity

Academy

How does **Academy** support the expression of **diverse agent behaviors** and deployment across **distributed/federated resources**?

Inspiration

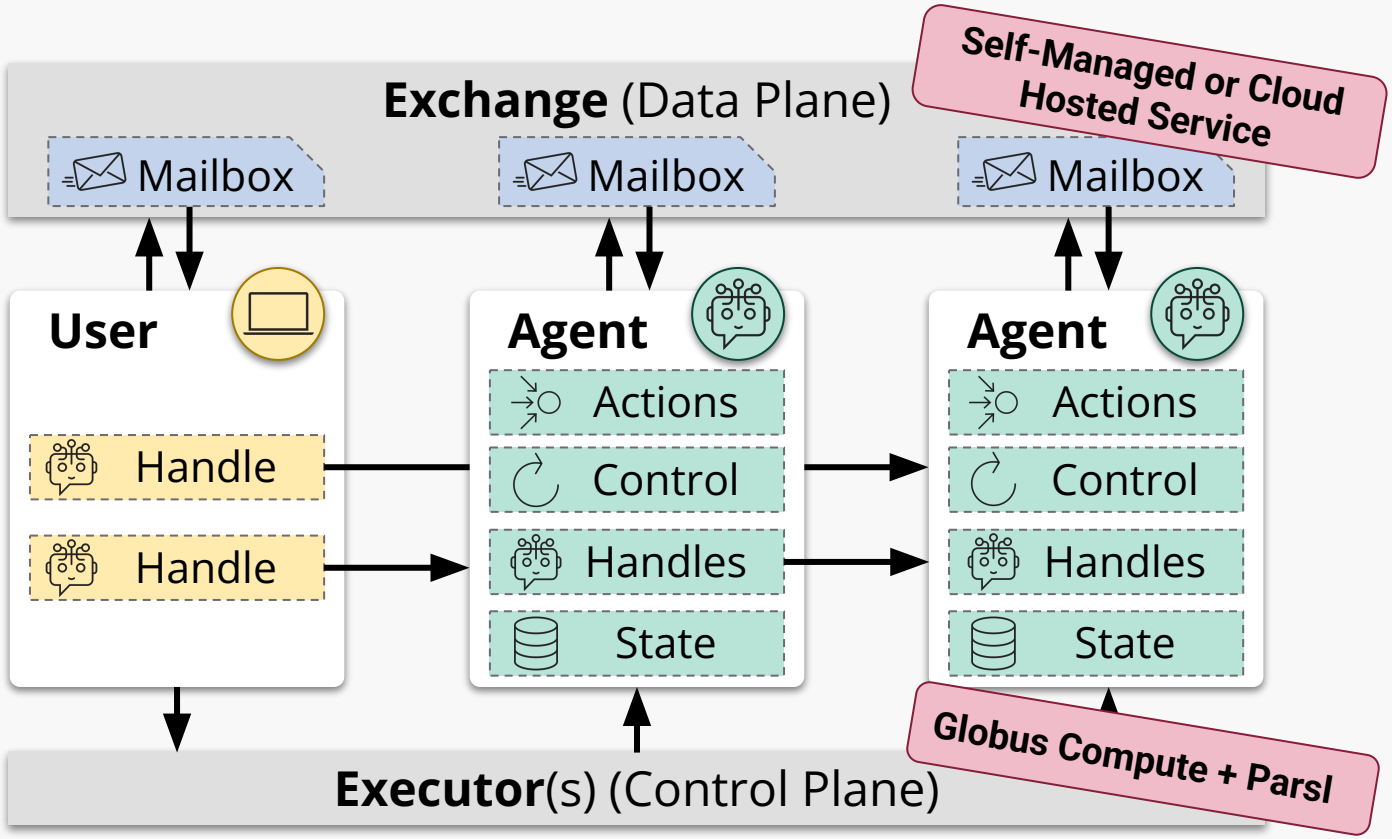


- *Fire-and-forget*: Agents spawned across remote/federated resources
- *Autonomy*: Agents have agency over their actions and local state/resources
- *Cooperative*: Agents interact to execute tasks & workflows

Focus 3: Coordinate async agent messaging

Focus 1: Program diverse agents and interactions

Focus 2: Deploy agents on federated resources



https://docs.academy-agents.org/latest/concepts/

Diverse Communication Options

Local Exchange

➔ In-memory, no setup, for local applications or fast debugging

Hybrid Exchange

➔ Direct communication first, with mediated fallback for resilience

➔ Leverage high-speed interconnects within clusters/sites

Cloud Exchange

➔ Http service authenticated with Globus

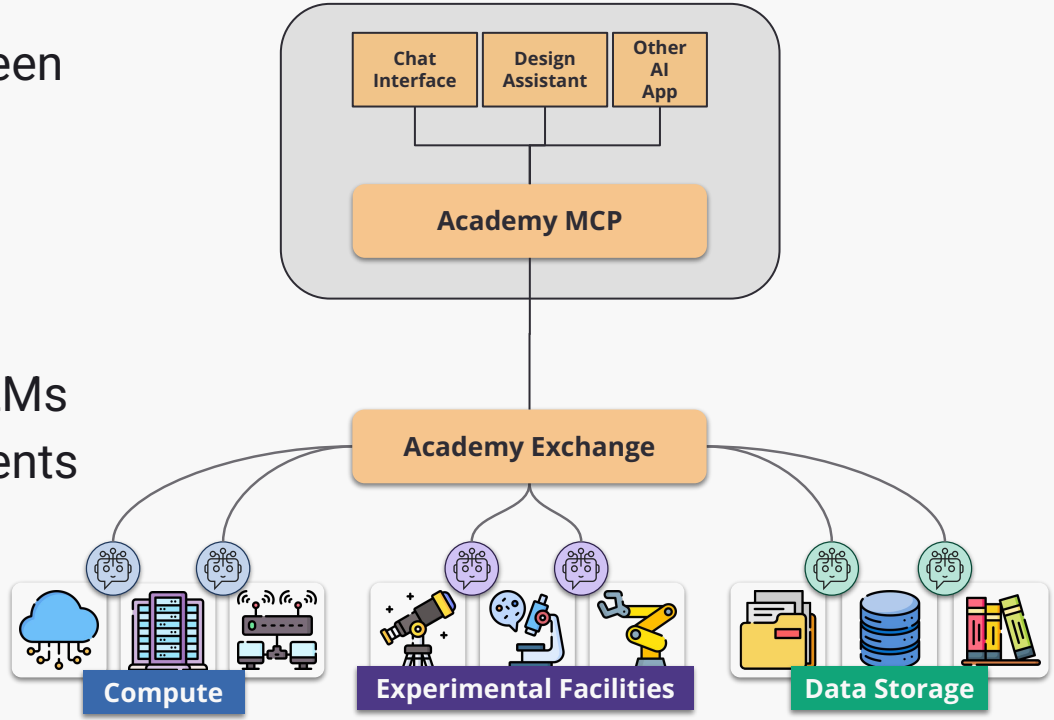
➔ Secure multi-site applications

Features (rapid fire)

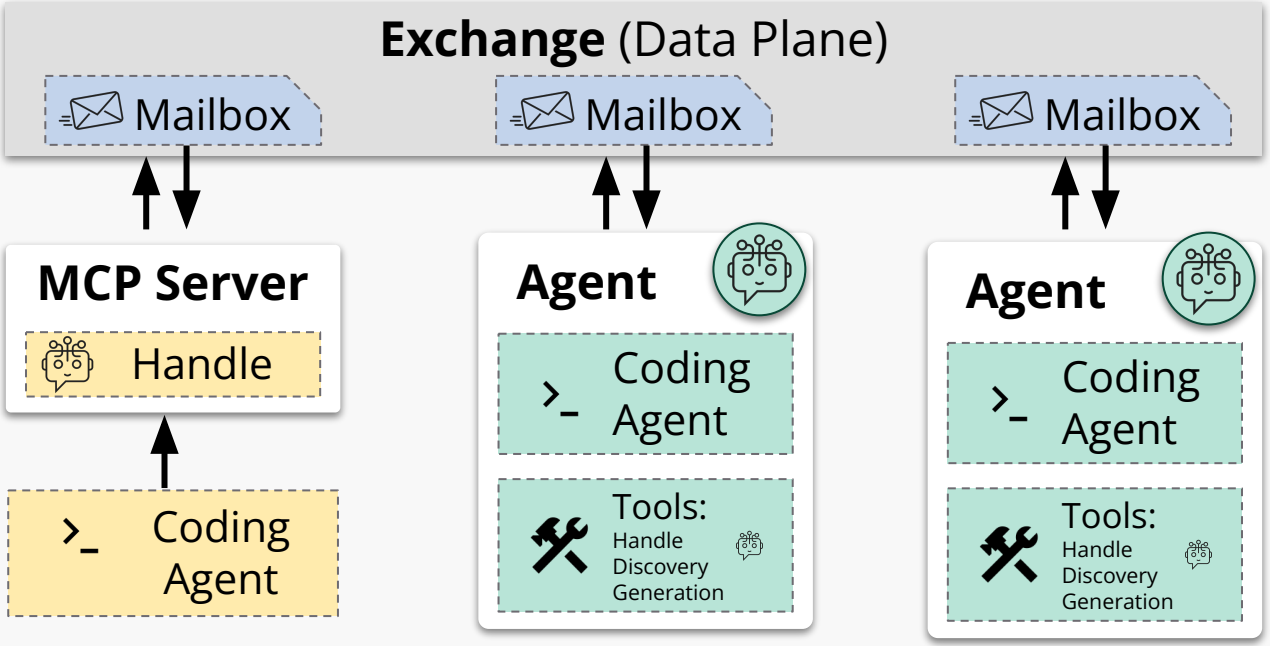
- ➔ Any number of actions & control loops
- ➔ Special purpose control loop decorators
- ➔ Async/non-blocking action execution
- ➔ Startup and shutdown callbacks
- ➔ State persistence plugins
- ➔ Re-execution on failure
- ➔ Agents can launch other agents
- ➔ Discovery/lookup based on behavior

Academy-MCP Plug-in

- ➔ Local MCP server bridges between Academy agents and AI applications
- ➔ Turn Academy agents into LLM **tools**
- ➔ Expose agentic **workflows** to LLMs
- ➔ Automatic **discovery** of new agents

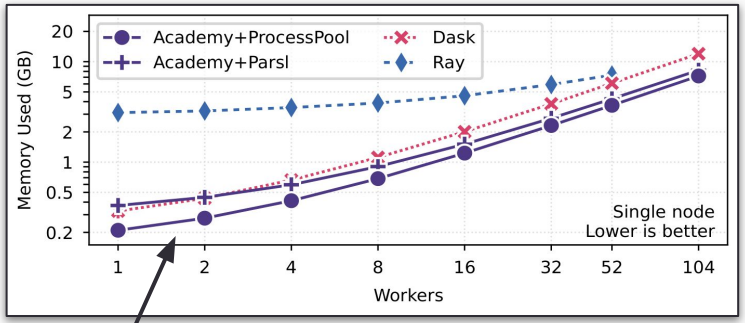


Academy as Execution Management

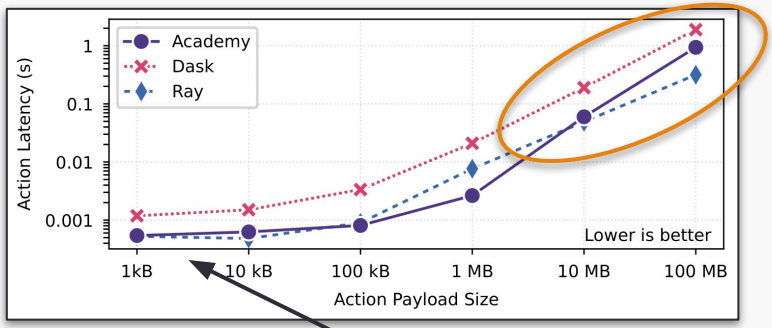


Comparisons to Actor Systems

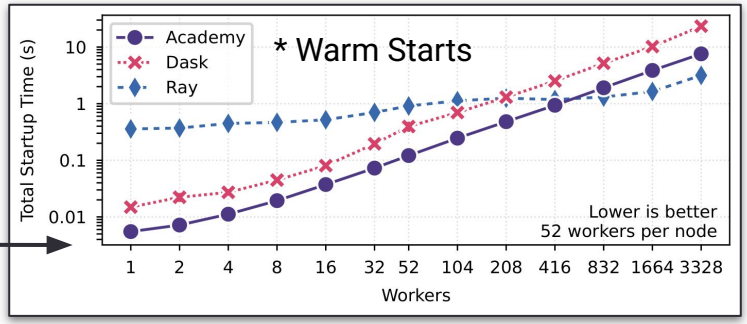
Why we need ProxyStore!



Low-memory overhead



Low-latency messaging



Fast start-up

Experiments performed on Aurora @ ALCF

Writing Apps in Academy

Agents defined by a class

Clients & other agents can request actions

```python
import asyncio
from academy.agent import Agent, action, loop

class Example(Agent):
    def __init__(self) -> None:
        self.count = 0  # State stored as attributes

    @action
    async def square(self, value: float) -> float:
        return value**2

    @loop
    async def count(self, shutdown: asyncio.Event):
        while not shutdown.is_set():
            self.count += 1
            asyncio.sleep(1)
```

Instance of a agent is state

Control loops for autonomous behavior

https://docs.academy-agents.org/stable/get-started/

Single interface for managing your agents

Launch agent and get handle

Interact with agents via handles

```python
from academy.exchange import HybridExchangeFactory
...
from academy.manager import Manager

gce = GlobusComputeExecutor('<UUID>')

async with await Manager.from_exchange_factory(
        factory=HybridExchangeFactory(),
        executors=gce,
) as manager:
    behavior = Example()  # From the prior slide
    handle = await manager.launch(behavior)

    result = await handle.square(2)
    assert result == 4

    await handle.shutdown()
```

Launch agents via Globus Compute

https://docs.academy-agents.org/stable/get-started/

```python
from academy.agent import Agent, action
from academy.handle import Handle

class Coordinator(Agent):
    def __init__(self, simulator: Handle[Simulator]) -> None:
        self.simulator = self.simulator

    def process(self, result: Result) -> Result: ...

    @action
    async def compute_property(self, smiles: str) -> float:
        result = await self.simulator.simulate(smiles)
        # Run sync code in thread
        processed = await self.agent_run_sync(self.process, result)
        return processed
```

Call sync libraries from async actions

Pass handles to other agents or between processes

Integrating Academy and LLMs

```python
from academy.handle import Handle
from langchain_core.tools import tool

def make_sim_tool(handle: Handle[MySimAgent]):
    @tool
    async def compute_property(smiles: str) -> float:
        """Compute molecule property."""
        return await handle.compute_property(smiles)
    return compute_property

tool = make_sim_tool(agent_handle)
print(tool.args_schema.model_json_schema())
```

Turn agent handles into LLM framework “tools”

Comparison to Alternatives

Tool Servers (MCP) **Academy-MCP Plugin**

Rely on externally reachable endpoints that are blocked by facility policies. Requires user to manage services, infrastructure, and VPNs

Func-as-a-Service (Globus Compute)

Easier remote execution (no VPN, infrastructure management) but tools must be stateless, short-running tasks

Recap

Build

➔ Unified framework for creating diverse, stateful, and autonomous agents

Deploy

➔ Run and coordinate agents seamlessly across federated HPC resources

Integrate

➔ Connect agents with LLMs, MCP servers, instruments, robots, and more

Evolve

➔ Design agentic workflows that are composable, resilient, and self-managing

Next we'll build some some simple agents

Questions?

Up next → Hands-on: Getting Started

Academy

Hands-on: Getting Started

2:15 pm – 3:00 pm

github.com/academy-agents/academy-tutorial

Setup

Checkout the tutorial repo

```
$ git clone https://github.com/academy-agents/academy-tutorial
$ cd academy-tutorial
$ git checkout sc26
```

Install dependencies in a virtual environment

UV

```
$ uv venv --python 3.12
$ source .venv/bin/activate
$ uv pip install -e .
```

Conda

```
$ conda create -n academy python=3.12
$ conda activate academy
$ pip install -e .
```

Native

```
$ python -m venv venv
$ source venv/bin/activate
$ pip install -e .
```

Connecting to a model

Join the group:

https://app.globus.org/groups/7484677a-df8b-11f1-9973-0afff9abf491

Log in to ALCF Inference Service

```
$ uv sync
$ uv run alcf-tokens login inference
```

If not using uv, edit `.claude/settings.local.json`

Use Inference Service to Chat

```
$ alcf-ai chat -c minerva -m beam-1.0-bf16  "hello"
```

Your First Academy Application

Goal: Write Simple Chat Agent

- **State:** Claude Session
- **Actions:** message, query
- **Tips:**
  - Use `LocalExchangeFactory` for communication

Define an agent class with state

Integrate with Claude Agent SDK and other libraries

```python
import asyncio
from academy.agent import Agent

class ExampleAgent(Agent):
    async def agent_on_startup(self):
        options = ClaudeAgentOptions(
            model=self.model,
            system_prompt='You are a helpful assistant.',
            tools=[],
            max_turns=12,
        )
        self.client = ClaudeSDKClient(options=options)
        await self.client.connect()

    async def agent_on_shutdown(self):
        if self.client is not None:
            await self.client.disconnect()
```

State can be initialized on __init__ or with on_startup method

Expose functions on to the user and other agents

```python
from academy.agent import Agent

class ExampleAgent(Agent):
    ...

    @action
    async def message(self, prompt: str) -> None:
        await self.client.query(message)
        await self.client.receive_messages()
```

State across calls for convenient persistent sessions

Manage the launch and communication of agents

```python
from academy.exchange.local import LocalExchangeFactory
from academy.manager import Manager


async with await Manager.from_exchange_factory(
    factory=LocalExchangeFactory(),
    executors=ThreadPoolExecutor(),
) as manager:
    handle = await manager.launch(Example)

    await handle.increment()
    result = await handle.get_count()
    assert result == 1
```

Launch agent and get handle

Interact with agents by invoking actions

Agent-to-Agent Communication

Goal: Build a non-LLM Agent

- **State: ML Interatomic Potential Model**
- **Actions:** Predict Energy
- **Tips:**
  - Load the model into the agent on start-up
  - Use `LocalExchangeFactory` for communication
  - Run code within the `MaterialsAgent` context
  - Have the LLM use `discover` to create a handle.

Add a tool to the Claude SDK tools

```
class ExampleAgent(Agent):
    async def agent_on_startup(self):
        @tool(
            'execute_python',
            ...
        )
        async def execute_python(args):
            output = await run_python(
                args['code'],
                self.namespace
            )
            ...
```

Run code in agent environment

Add a tool to the Claude SDK tools

```
class ExampleAgent(Agent):
    async def agent_on_startup(self):
        @tool(
            'execute_python',
            ...
        )
        async def execute_python(args):
            output = await run_python(
                args['code'],
                self.namespace
            )
            ...
```

Run code in agent environment

Agent can find other agents

Invoke other agents from Handle

```
### LLM Generates Code
# First, let's discover available agents
agents = await agent.agent_exchange_client.discover(Agent)
print("Available agents:", agents)

# Let's describe both agents to see what they can do
agent_handles = [Handle(agent_id) for agent_id in agents]
for i, handle in enumerate(agent_handles):
    desc = await handle.agent_describe()
    ...
    for action_name, action_info in desc.actions.items():
        print(f"  - {action_name}: {action_info}")

# Now let's predict energies for both structures
energy_agent = Handle(agents[0])  # The ML potential agent
energies = await energy_agent.predict_energy(structures)
```

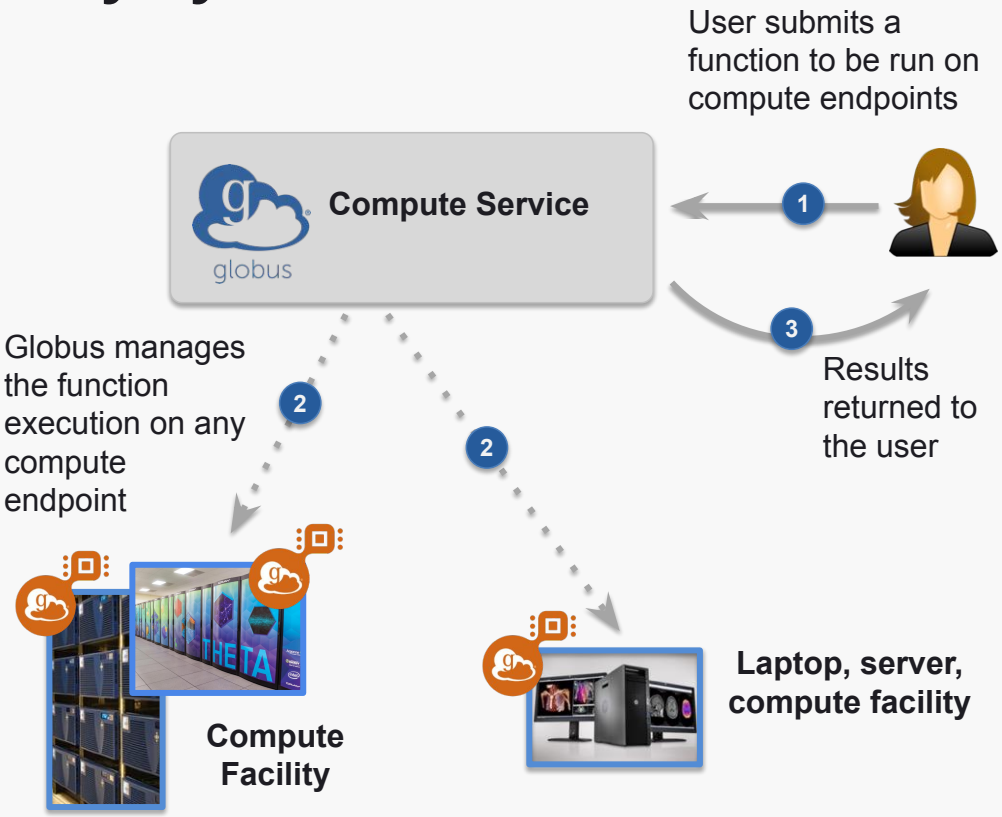
Agent docs exposed over exchange

Distributed Execution

Goal: Launch MLIP Agent Remotely

Managed compute …on any system

- **Support use of Python for functions**
- **Fire and forget function execution**
- **Federated authentication, and local access control**
- **Uniform interface to various compute resources**



Globus Compute

FaaS Service + “bring-your-own compute” model

1. Deploy “endpoint” on Cluster or Laptops
2. Define python function for remote execution
3. Submit function to cloud service

```
pip install globus-compute-endpoint
globus-compute-endpoint configure
globus-compute-endpoint start <ENDPOINT_NAME>
export ACADEMY_TUTORIAL_ENDPOINT=<ENDPOINT_ID>
```

Try it!

```
from globus_compute_sdk import Executor

def hello_world():
    return "Hello World!"

tutorial_endpoint_id = '360baa9b-3f74-41d6-8ce1-3c8e64a1231b'
with Executor(endpoint_id=tutorial_endpoint_id) as fxe:
    future = fxe.submit(hello_world)
    print(future.result())
```

Try online: https://jupyter.demo.globus.org/

Distributing Agents

Goal: Run MLIP Agent in the Cloud

- Use `HttpExchangeFactory` for communication

```
HttpExchangeFactory(
    url='https://exchange.academy-agents.org',
    auth_method='globus'
)
```

- Use `GlobusComputeExecutor` for launching

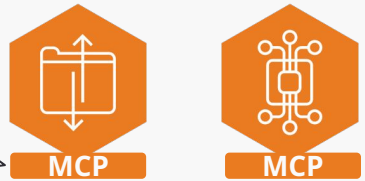
```
GlobusComputeExecutor('<YOUR_ENDPOINT_ID>')
```

Other features: Globus MCP servers

- Enable LLMs and agentic systems to manage data and computation across remote resources
  - Various transfer and compute capabilities supported with open Globus MCPs
  - Authentication via Globus client credentials



Tools:
- submit_transfer_task()
- get_task_events()
- …



Tools:
- list_my_endpoints()
- register_python_function()
- register_shell_command()
- submit_task()
- get_task_status()

Questions?

Up next → Hands-on: Battleship

Academy

Break

3:00 pm – 3:30 pm

Academy

Patterns and Examples of Agentic Applications

3:30 pm – 3:45 pm

THE UNIVERSITY OF CHICAGO

globus labs

Replacing the Human-in-the-Loop

Humans synthesize knowledge and propose hypotheses

Humans write, debug, and run programs

Humans interpret results to inform new hypotheses

} Inefficient use of research infrastructure

Agents can be the driving entities

- ➔ Persistent, stateful, cooperative
- ➔ Intermittent human oversight

} We need to be here

Credit: Ian Foster, "**Empowering Science with Intelligent Middleware and Embodied Agents**"

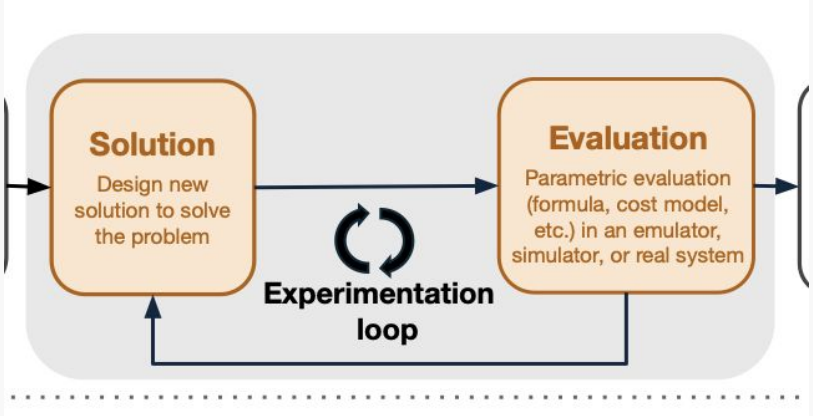
Agentic Workflows for Science

- ✓ Automate closed-loop processes
- ✓ Natural expression of scientific resources (compute, instruments, repositories)
- ✓ Operate autonomously but still cooperatively
- ✓ Execute multi-stage computational science processes
- ✓ Reduce mundane tasks & responsibilities of scientists

The whole is greater than the sum of its parts.
- Aristotle

What Makes a Good Problem?

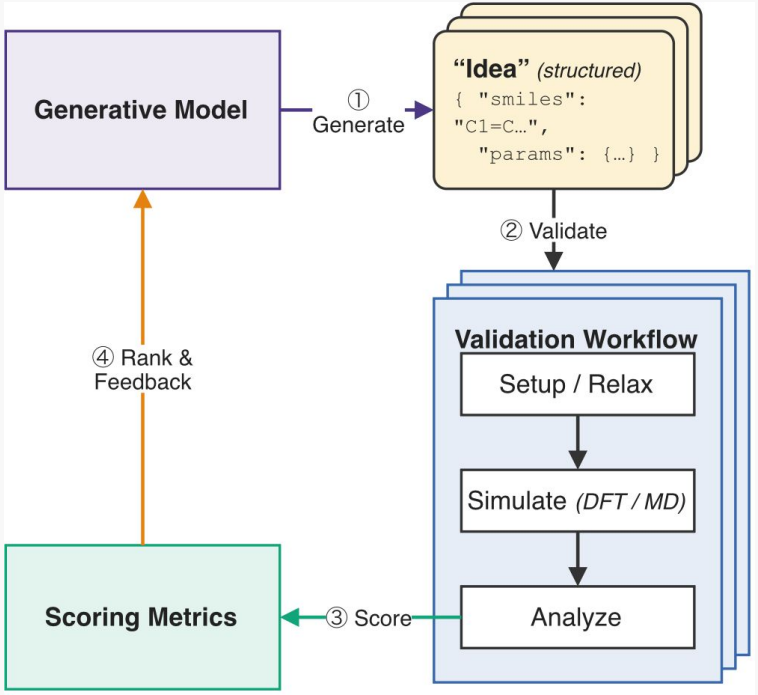
- *“ …verification accuracy is key to the success of an AI-driven approach…”* - Barbarians at the Gate
- Agents make “proposing” ideas very fast
  - Candidate molecules, protein binders, or materials
  - Algorithms/Heuristics for scheduling
  - Proofs
- Validation and Feedback are then the challenge
- Patterns differ in how/where proposal happens and how validation happens



Cheng et al., Barbarians at the Gate, https://arxiv.org/abs/2510.06189.

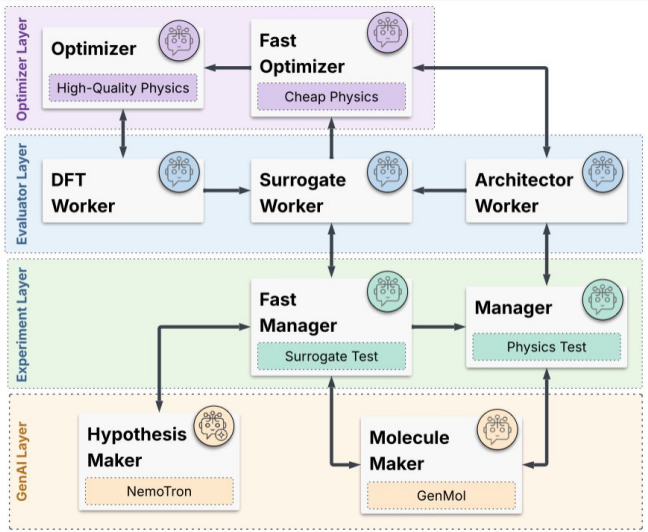
Pattern 1: Structured Validation Workflow

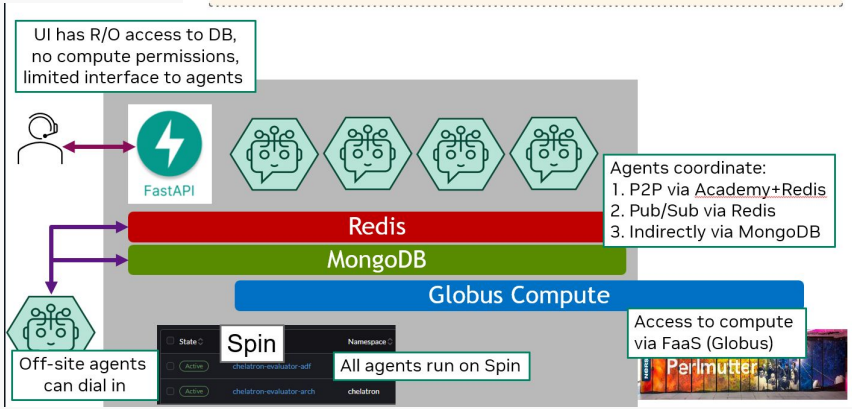
- Generative Model sits as component in the workflow
- “Idea” - structured data that can be used deterministically
- Scoring metrics and workflow are pre-defined
  - Domain expertise and experience informs construction
  - Traditional HPC tools ensure system utilization, scalability
- Throughput and feedback determined by workflow
- Low autonomy, high reliability



Chelator Design for Cancer Treatment

- LLM generates hypotheses of “good” chelators + simulation tools test candidates
- Experiment agents find traits suitable for molecules
- *Manager* (Nemotron with RAG) create hypotheses
- *GenMol* generates matching molecules
- Surrogate model (*Architector*) screens molecules
- *Optimizers* pick promising candidates from surrogate predictions and evaluate them with increasingly expensive physics codes
- Results are used to retrain the surrogate



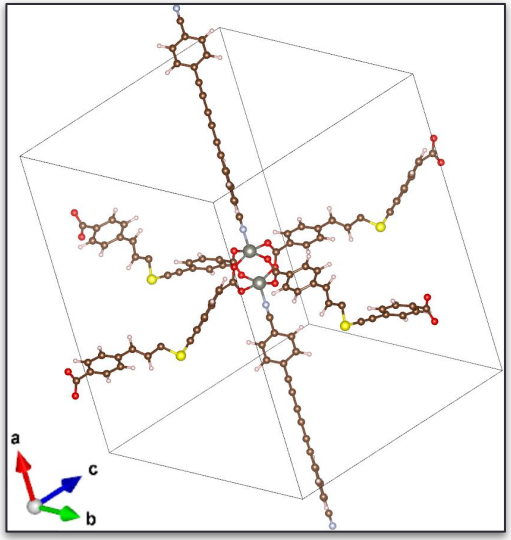


Use Case: **MOF Discovery**

Metal Organic Frameworks (MOF)

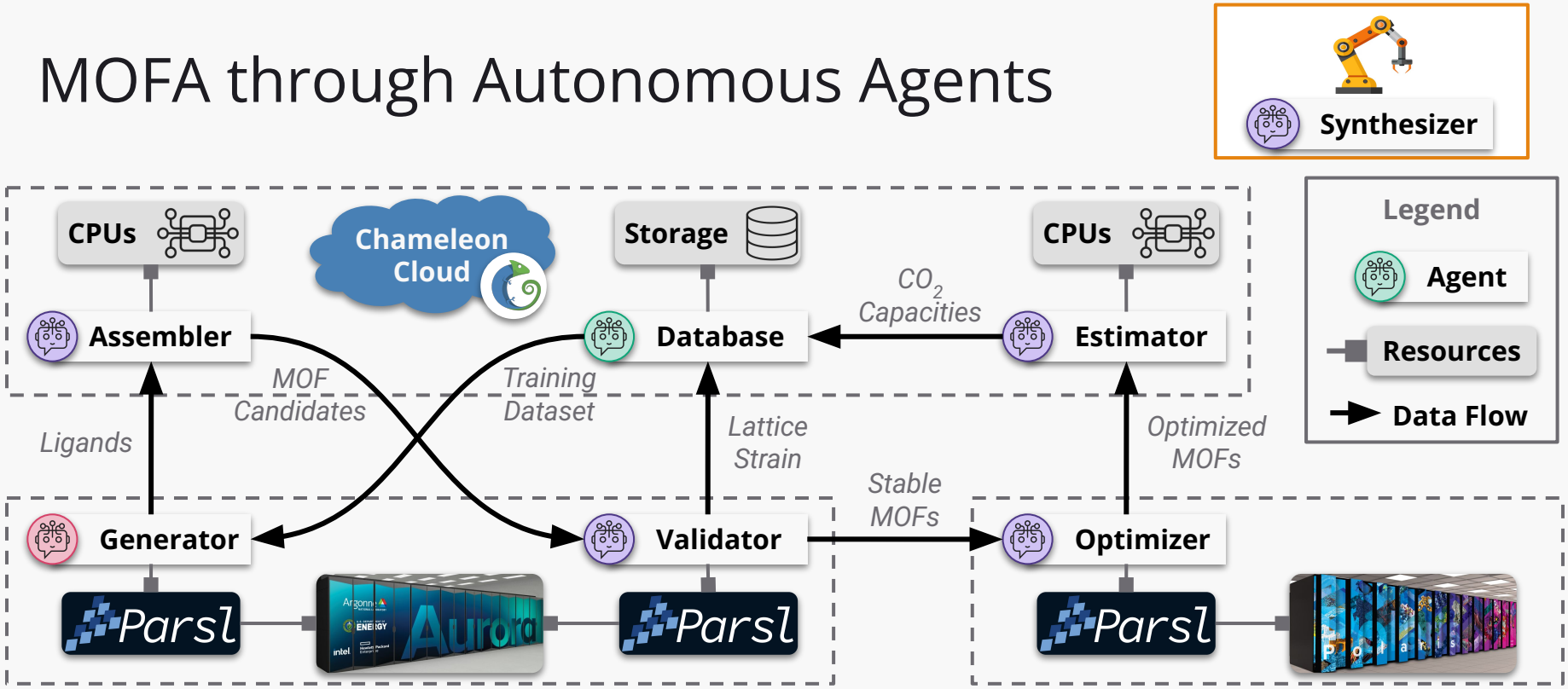
- ➔ Composed of organic molecules (ligands) and inorganic metals (nodes)
- ➔ The sponges of materials science!
- ➔ Porous structures that adsorb and store gases
- ➔ Topologies can be optimized for targeted gas storage → **Carbon Capture**

How to efficiently discover MOFs with desirable properties for target applications?



Intractable search space of ligand, node, & geometry combinations

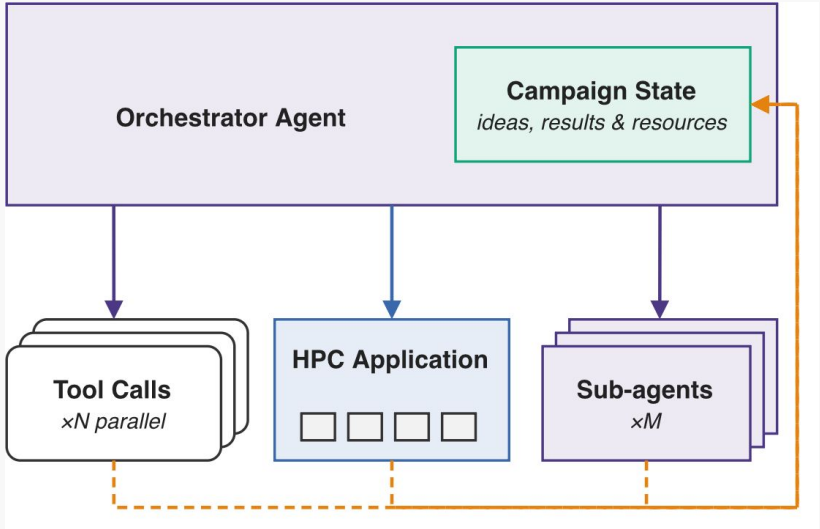
MOFA through Autonomous Agents



Agents executed remotely via Globus Compute

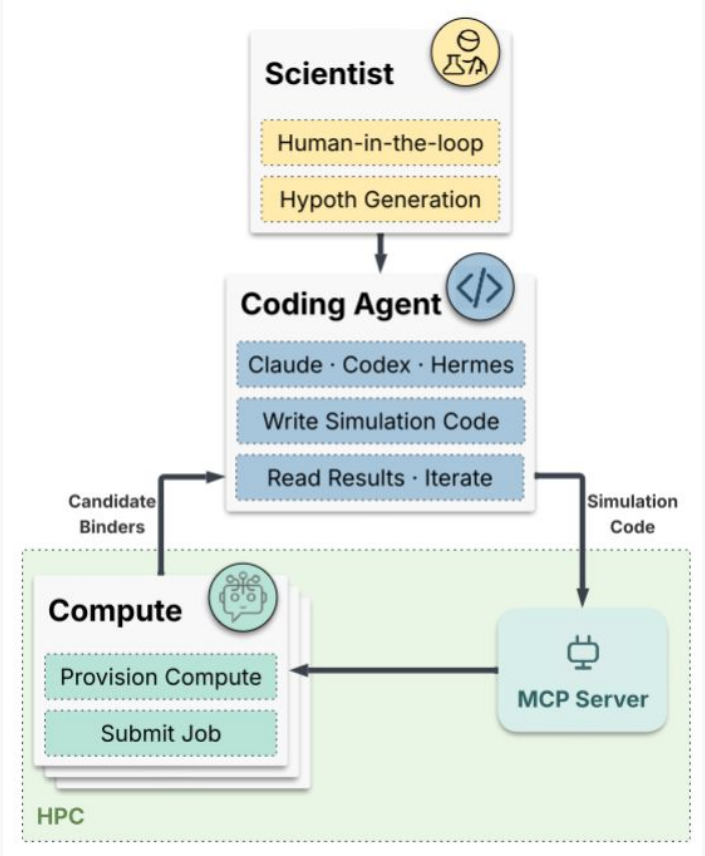
Pattern 2: Centralized Orchestration

- Centralized agent drives whole campaign
- Autonomy to plan campaign, provision resources, and divide work
- Full view of all information
- Can scale by:
  - Launching many tool calls
  - Writing HPC applications
  - Launching sub-agents



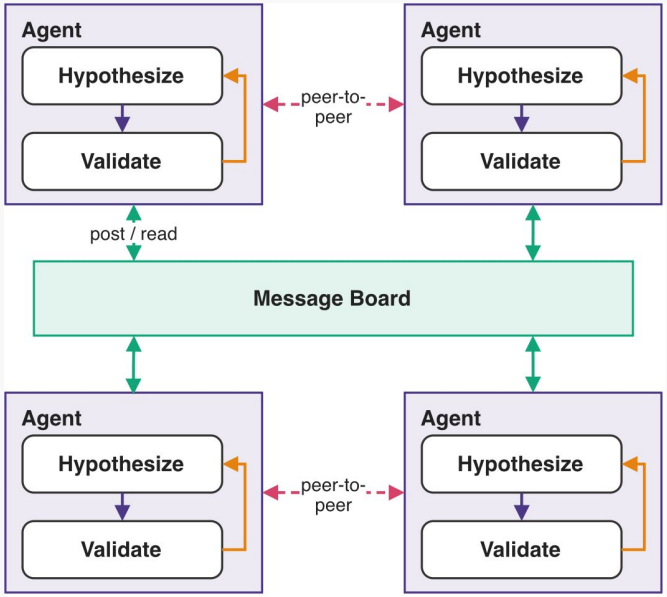
Protein-Binder Design

- Disordered proteins lack a stable binding site, defeating structure-based drug design
- A single agent in a commercial harness (Hermes or Claude Code) writes simulation code directly instead of calling predefined tools
- An Academy agent runs inside the compute job, keeps it alive, and submits the generated code to HPC resources
- Manages resources effectively across nodes and 20 GPUs; previous version ran to 2000 nodes



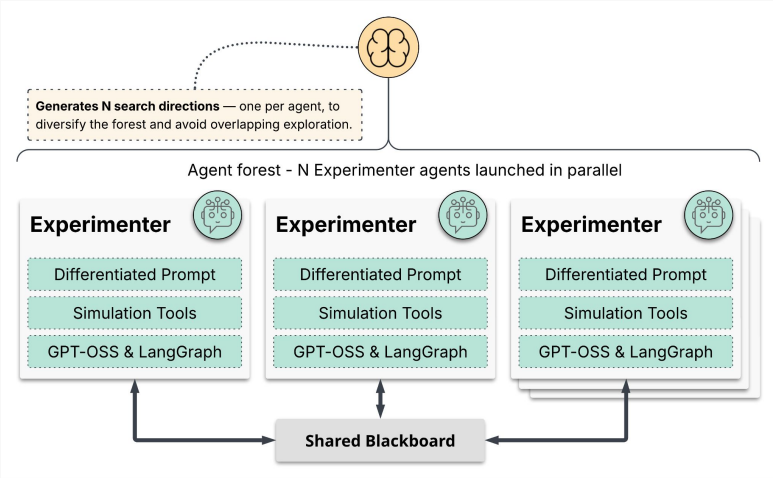
Pattern 3: Decentralized Coordination

- Parallel agents explore ideas independently
- Discovery and validation are LLM driven
- Adaptive exploration methods
- Peer-to-peer or message board for agent to agent communication
- Examples: Navier-Stokes, Hugging Face hack



Electrolyte Design for Organic Batteries

- Designing battery electrolytes requires balancing competing objectives (conductivity, stability, safety)
- Traditionally manual experiments, high-throughput screening, or Bayesian learning workflows
- Agent forest
  - Agents use open-source reasoning models to propose molecules + simulation tools to calculate properties
  - Each experimenter agent is given a different prompt regarding the search direction
  - Agents run independently but collaborate via blackboard



Academy

Hands-on: Multi-Agent Patterns

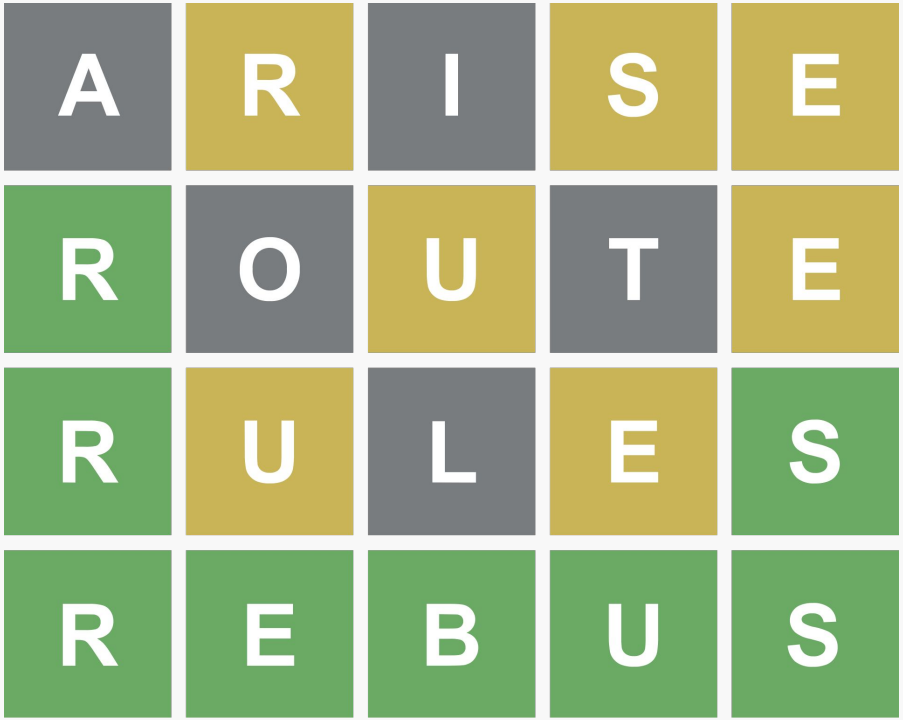
3:45 pm – 4:15 pm

THE UNIVERSITY OF CHICAGO

globus labs

Wordle Game Overview

- Hidden five letter word
- Iterative guessing to determine the target word
- Each letter is scored:
  - Green - correct letter in the correct spot
  - Yellow - letter within the word in the incorrect spot
  - Grey - Letter not in word



Parallel Agents

Goal: Launch Parallel Agents to Play Wordle

- **Tasks:** Implement the `play_game` action, and the parallel search
- **Tips**
  - Use the agent state to give the LLM the current game
  - Use `asyncio.wait(..., return_when=asyncio.FIRST_COMPLETED)`
  - Cancel tasks and `hdl.shutdown`

Blackboard Communication

Goal: Inject insights from peers into context to inform guesses

- **Task:** Implement an action to receive information from peer agents
- **Tips:**
  - Store information in the `peer_inights` dictionary.
  - Decide to store the latest information only, or to store

Centralized Orchestration

Goal: Search for best Wordle strategy, then play game

- **Task:** Search for the best strategy using multiple agents across games
- **Tips:**
  - Searching the strategies should be independent of playing the game
  - The tools for the LLM are already provided, you just need to query the LLM
  - *If there is extra time*: Try giving either the Coordinator or the Worker agents a coding tool instead of the simulation tool

Academy

Hands-On: Monitoring Agents

4:30 pm – 4:45 pm

Dashboard

How do you monitor and interact with remote agents?

- Monitoring and observability through `academy-dashboard`

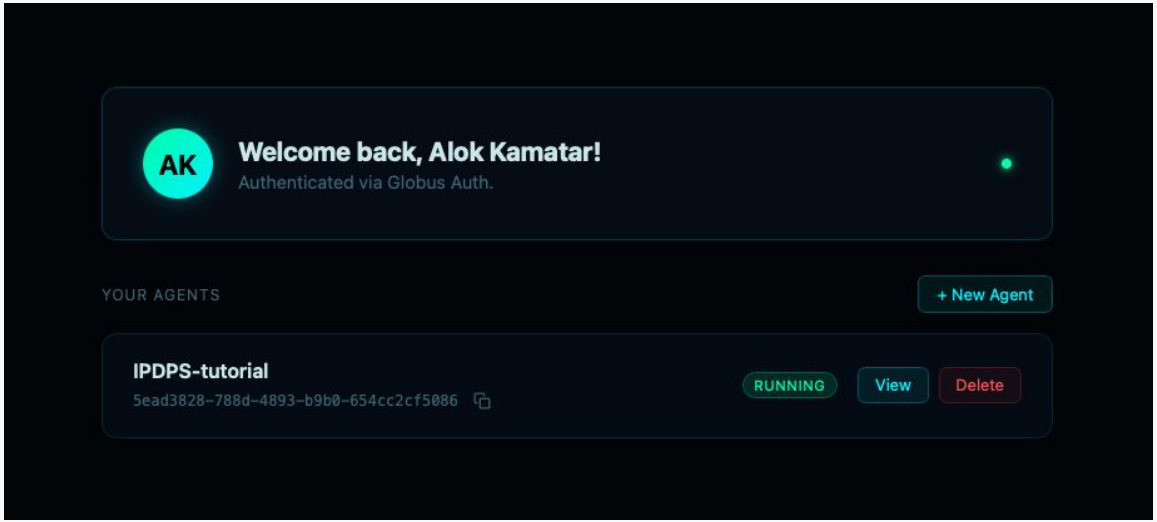
```
$ uv sync --extra dashboard
```

- Locally hosted or service based solution

https://dashboard.academy-agents.org

Demo: Dashboard

Linked to the hosted exchange or local exchange



Observability structured as `Agents`

Monitored Agent

```
$ uv sync --extra dashboard
```

MonitoredAgent sends logs back to UserAgent

Can query UserAgent before performing a task

```python
import asyncio
from academy_dashboard import MonitoredAgent

class Example(MonitoredAgent):
    def __init__(self, user_agent) -> None:
        super().__init__(user_agent_handle=user_agent)

    @action
    async def unsafe_action(self) -> None:
        await self.user_agent.prompt_user(
            ...
        )
```

Academy

Wrap up

4:45 pm – 5:00 pm

Discussion

- How are you using (or planning to use) agents in your work?
- What makes scientific and cyberinfrastructure contexts uniquely challenging or interesting for agentic systems?
- What are the key capabilities agents need to effectively support scientific workflows (e.g., reasoning, coordination, adaptation)?
- What challenges have you faced (or anticipate) in deploying, managing, or trusting agents across distributed systems?
- How can we promote interoperability and shared frameworks for agents across institutions and domains?
- What new opportunities could agentic systems enable for scientific discovery or collaboration?

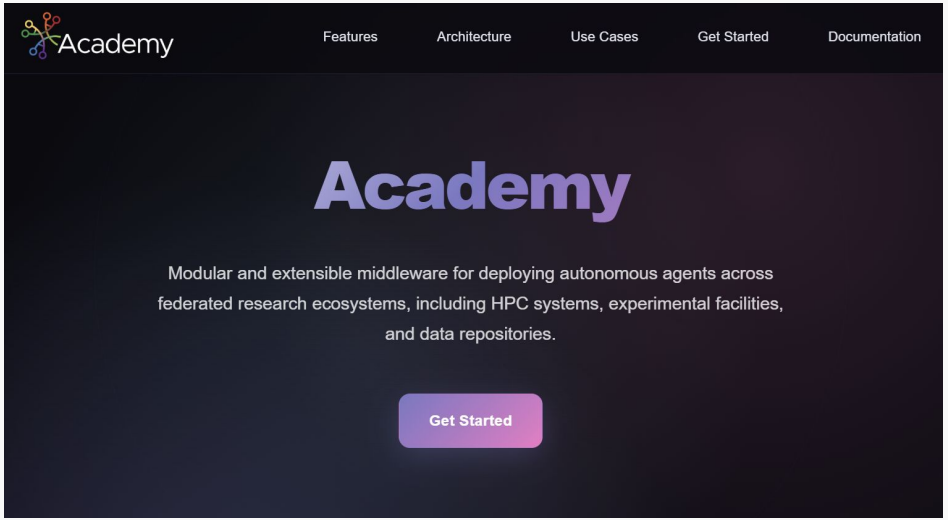
Get Engaged

Try it out / star Academy on github / open issues / contribute to the open source project

Read the paper

Join the community / let us know how you are using Academy

Join the open community meeting Wednesday 1pm CT



Questions?

Academy

Thanks for attending!

Meet the Academy team


Greg **Pauloski**


Alok **Kamatar**


Yadu **Babuji**


Ryan **Chard**


Mansi **Sakarvadia**


Ben **Clifford**


Kyle **Chard**


Ian **Foster**

Learn More

- ➔ Website: academy-agents.org
- ➔ Docs: docs.academy-agents.org
- ➔ Paper: arxiv.org/abs/2505.05428v2

Reach Out

- “Community” linked in website footer
- github.com/academy-agents
- chard@uchicago.edu

Get Started

$ pip install academy-py

academy-agents.org

THE UNIVERSITY OF CHICAGO

globus labs