Academy

Building Scalable Agentic Systems for Science

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

Kyle Chard, Alok Kamatar, Yadu Babuji

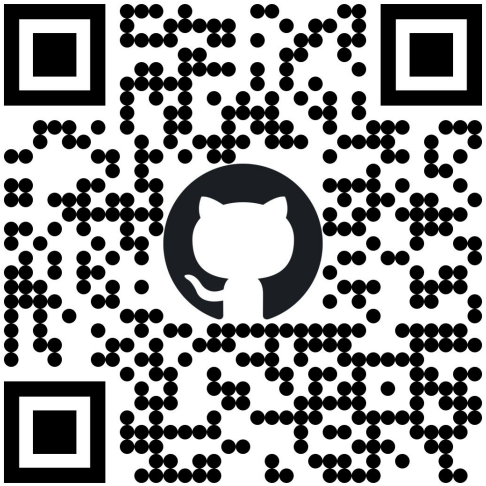
github.com/academy-agents/academy-tutorial

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

Tutorial Materials



github.com/academy-agents/academy-tutorial

Welcome!

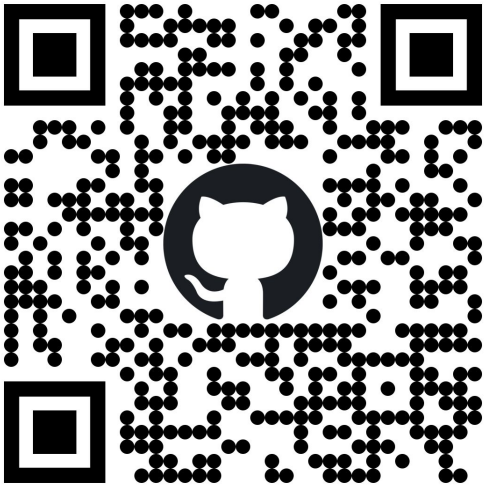
Audience

- ➔ 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 3.10–3.13

Tutorial Materials



github.com/academy-agents/academy-tutorial

Academy

Introduction

THE UNIVERSITY OF CHICAGO

globus labs

What is an “agent”?

- In computer software, a “software agent” is [Wikipedia] “**a computer program that acts for a user or another program in a relationship of agency**”

- In artificial intelligence (AI), an “intelligent agent” is [also Wikipedia] “**an entity that perceives its environment, takes actions autonomously to achieve goals, and may improve its performance through machine learning or by acquiring knowledge**”
    - An AI component, sensors, actuators, memory

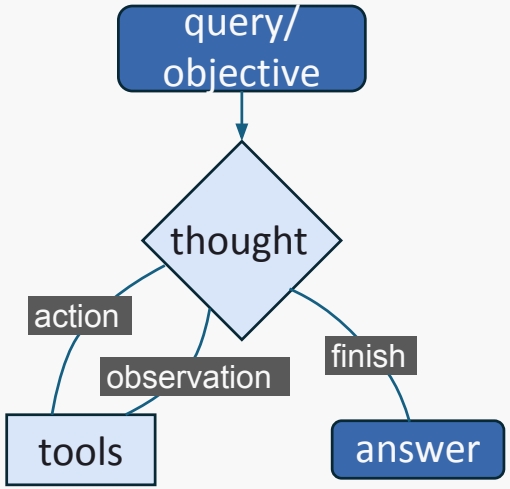
See also: https://gist.github.com/simonw/beaa5f90133b30724c5cc1c4008d0654

What is an “agent”?

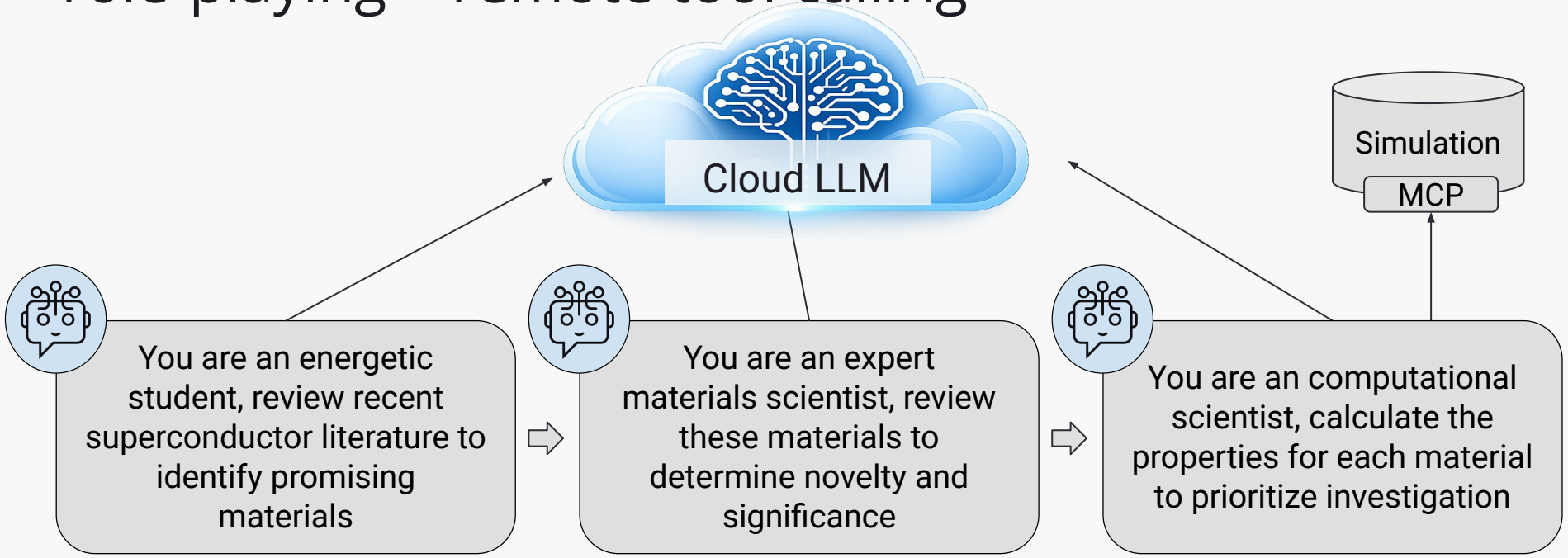
An agent is a persistent, stateful process that acts on behalf of a user or system. An agent may:

- **Observe** inputs or events
- **Plan** (decide on) actions using a policy (rules or LLM)
- **Act**: Execute tools or call other agents
- **Learn**: Update state to adapt over time

We can think of an agent as a scientific assistant that can reason, act, and coordinate on our behalf



Most agentic systems focus on centralized role-playing + remote tool calling



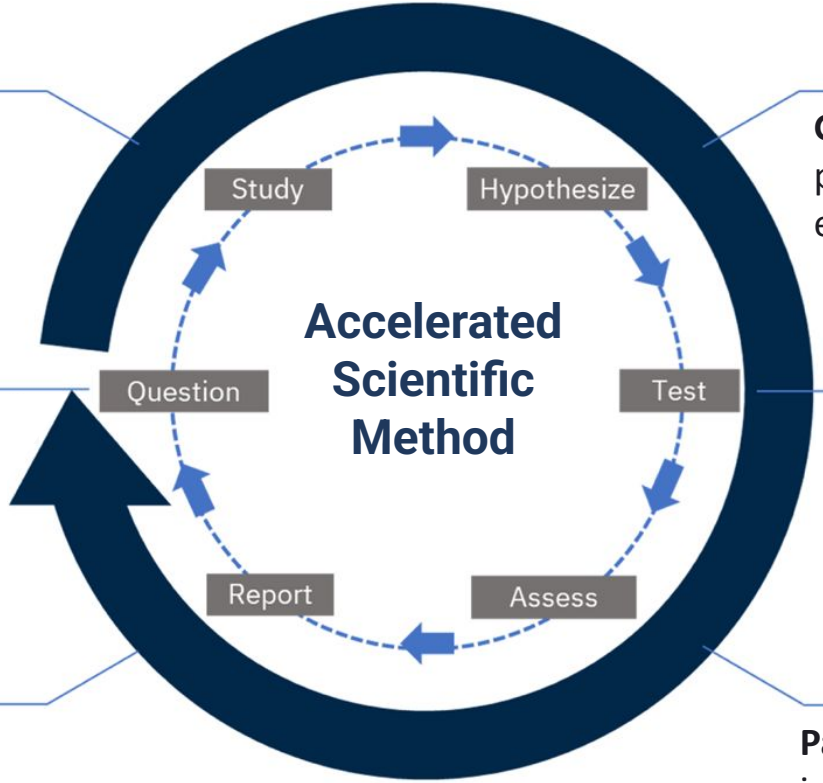
Deploying and managing agents

Accelerating discovery in science

Extraction, integration and reasoning with knowledge at scale

Tools help **identify new questions** based on needs and gaps in knowledge

Machine representation of knowledge leads to **new hypotheses and questions**



Generative models automatically propose new hypotheses that expand the discovery space

Robotic labs automate experimentation and bridge digital models and physical testing

Pattern and anomaly detection is integrated with simulation and experimentation to extract new insights

https://doi.org/10.1038/s41524-022-00765-z

The emergence of LLM-based agents for science

AI agents are autonomous systems that can **reason about tasks** and act to achieve goals by **leveraging external tools and resources**.

Modern AI agents are typically powered by large language models (LLMs) connected to external tools or APIs.

They can perform reasoning, invoke specialized models, and adapt based on feedback.

Agents differ from conventional “models” in important regards:

- They are interactive and adaptive
- Rather than returning fixed outputs, they can take multi-step actions, integrate context, and support iterative human–AI collaboration.
- Users can interact with them through human language, substantially reducing usage barriers for scientists.”

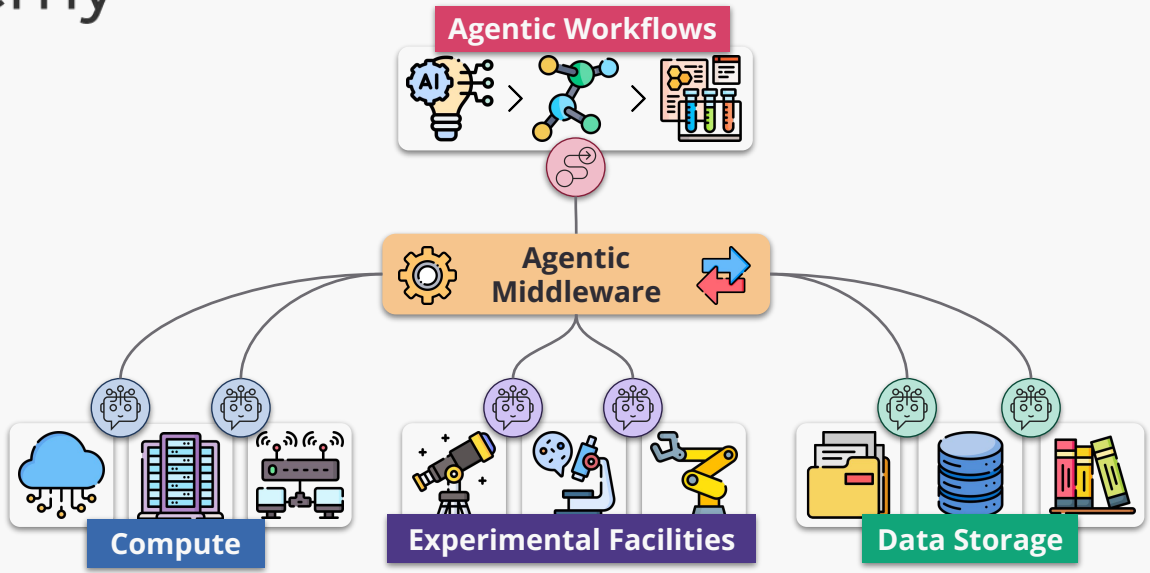
https://arxiv.org/pdf/2509.06917

One agent or many?

- In **principle**, a single reasoning model (like a single human) can apply a variety of different reasoning strategies, have specialized knowledge on different topics, improve expertise over time, etc.

- In **practice**, it is common to create multiple agents (like a team of people), e.g., for:
  - Modularity (specialization, reuse, maintainability)
  - Different roles (e.g., idea generator, idea critic, program generator, …)
  - Parallelism (run multiple copies of an agent to explore different ideas)
  - Expanded capacity (e.g., larger LLM context)

Academy

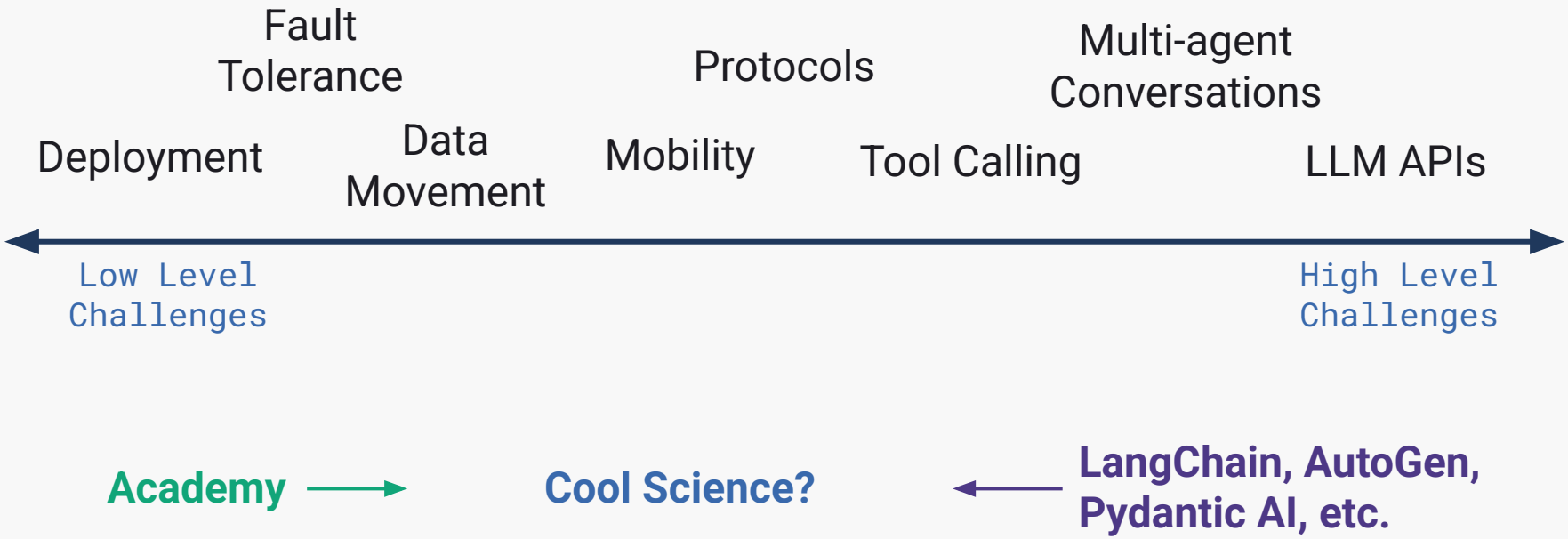


A middleware for building and deploying stateful actors and autonomous agents across distributed research infrastructure

Agentic Middleware

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

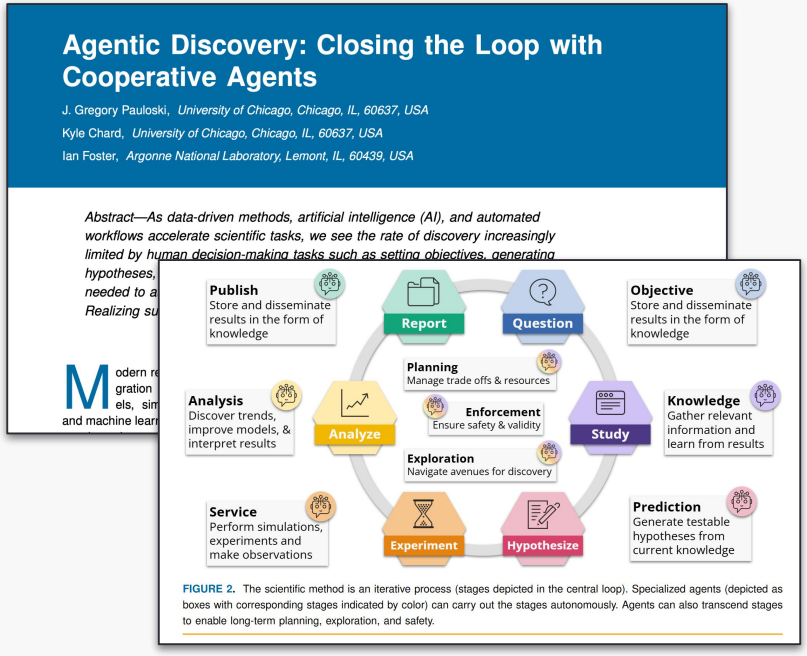
Agentic Middleware: Scope & Challenges



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**?

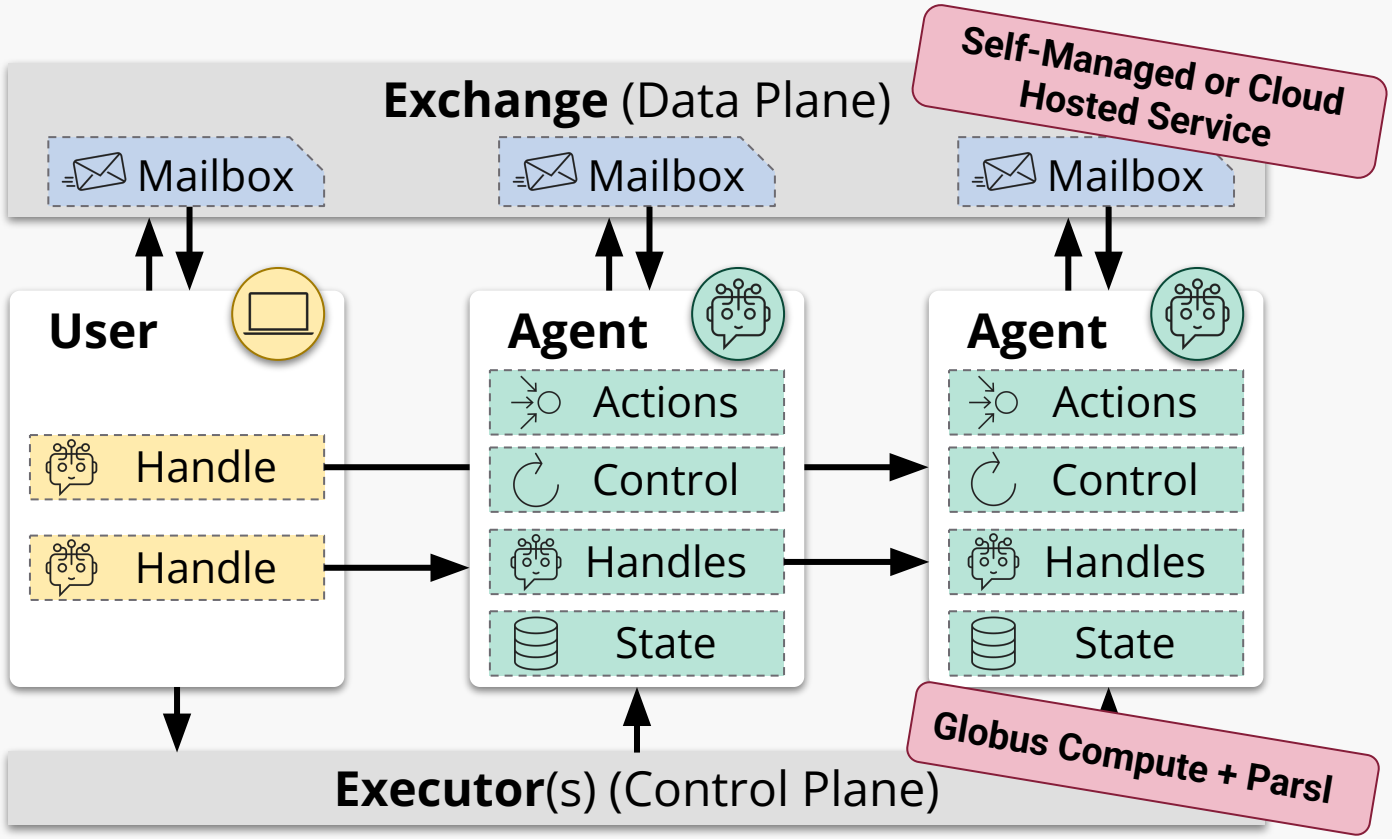
Requirements

- ➔ Federated orchestration
- ➔ Configurable data plane
- ➔ Temporally decoupled messaging
- ➔ Agent authentication and permissions
- ➔ Resilient state management

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

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/

Call sync libraries from async actions

```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
```

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 agents

Questions?

Up next → Hands-on: Getting Started

Academy

SC25 St. Louis, MO | hpc ignites.

Hands-on: Getting Started

THE UNIVERSITY OF CHICAGO

globus labs

github.com/academy-agents/academy-tutorial

Setup

Checkout the tutorial repo

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

Install dependencies in a virtual environment

Native

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

Conda

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

UV

```
$ uv venv --python 3.12
$ source .venv/bin/activate
$ uv sync --group llm
```

Jupyter notebook

Start Jupyter lab

```
$ jupyter lab
```

Open browser and navigate to link

1. Hello World

We start with the simplest Academy program: defining an agent and interacting with it. A HelloWorld agent is created with a single action, `hello`, which returns a string.

The program creates a local Academy manager, launches the agent locally, obtains a handle to it, and asynchronously invokes the action to produce and print the response.

Academy follows an asynchronous execution model. When an agent is launched, a handle is returned. This handle acts as a reference to the running agent and provides the interface through which we communicate with it. Using the handle, we can asynchronously invoke actions exposed by the agent. Here, we call the `hello` action and use `await` to pause execution until the action completes and returns its result.

In [ ]:

```
from academy.agent import Agent
from academy.agent import action
from academy.manager import Manager
from academy.exchange import LocalExchangeFactory
from concurrent.futures import ThreadPoolExecutor

class HelloWorld(Agent):

    @action
    async def hello(self, value) -> str:
        return f"Hello {value}"


async with await Manager.from_exchange_factory(
        factory=LocalExchangeFactory(),
    ) as manager:
        agent_handle = await manager.launch(HelloWorld)

        print(await agent_handle.hello("World"))
```

Academy

SC25 St. Louis, MO | hpc ignites.

Hands-on: Battleship

THE UNIVERSITY OF CHICAGO

globus labs

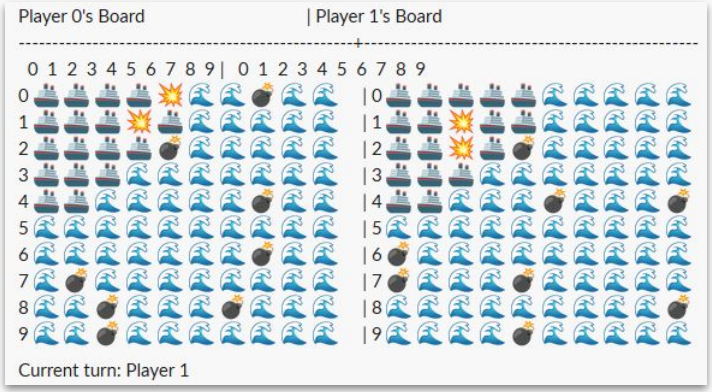
Example: Battleship (30 Minutes)

Battleship Rules:

- 10 x 10 grid of coordinates
- Each player has 5 “ships”
  a. A ship is a continuous run of coordinates
- To start each game, a player places ships on grid
- On each turn:
  a. Guess coordinate on opponents grid
  b. Recieve if guess was a hit or miss

Example: Battleship

1. Open the starter code from the repo.
2. Battleship Data Structures
   - Board
   - Game
3. Understand the `Coordinator` class
   - `game` - Runs a single game between two Player agents
   - `play_games` - Repeatedly plays games between two Players
4. Implement BattleshipPlayer
   - get_move - Choose coordinates of next attack
   - new_game - Place ships on new board



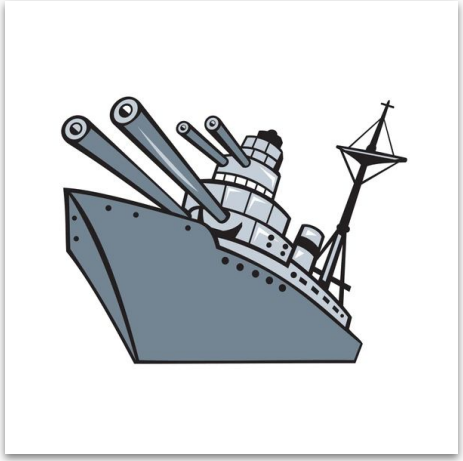
Battleship Extensions

Strategy Ideas:

- Guess in Clusters or in Rows based on hits/misses
- Keep track of opponents previous guesses to place ships next time
- Query LLM for next move?
  - How good is ChatGPT at War Games…

Let the games begin!

- To enter the tournament
  - Share the agent with Globus Groups
  - Register your agent (code below)
- Rules
  - Round-Robin Tournament
  - 0.25 Second Timeout on Moves (with communication)
  - 50 move limit on games



Battleship Tournament Entry Instructions

- Join the “Academy-SC-Tutorial” Globus Group
- Paste your agent into `solutions/05-battleship/enter_tournament.py`
- Set environment variables and run:

```
source tournament.env
python enter_tournament.py -n “Your Name”
```

- Sharing Agents on the Hosted Exchange

```
factory = HttpExchangeFactory(...)
console = await factory.console()
await console.share_agent(<agent_id>, <globus_group_id>)
```

Academy

SC25
St. Louis, MO | hpc ignites.

Demo: Academy Model Context Protocol

THE UNIVERSITY OF CHICAGO

globus labs

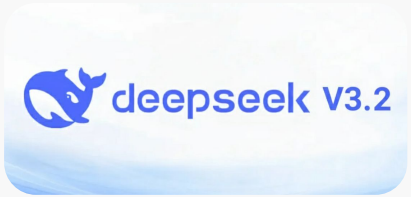
Building LLM-powered agentic applications

Popular LLMs
(and their general availability dates)

June 17, 2025

September 29, 2025

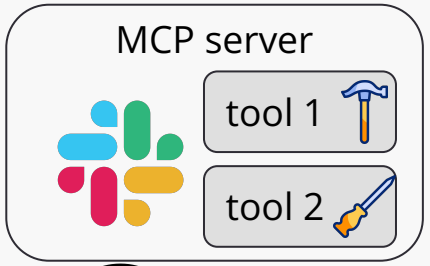
August 7, 2025



September 29, 2025

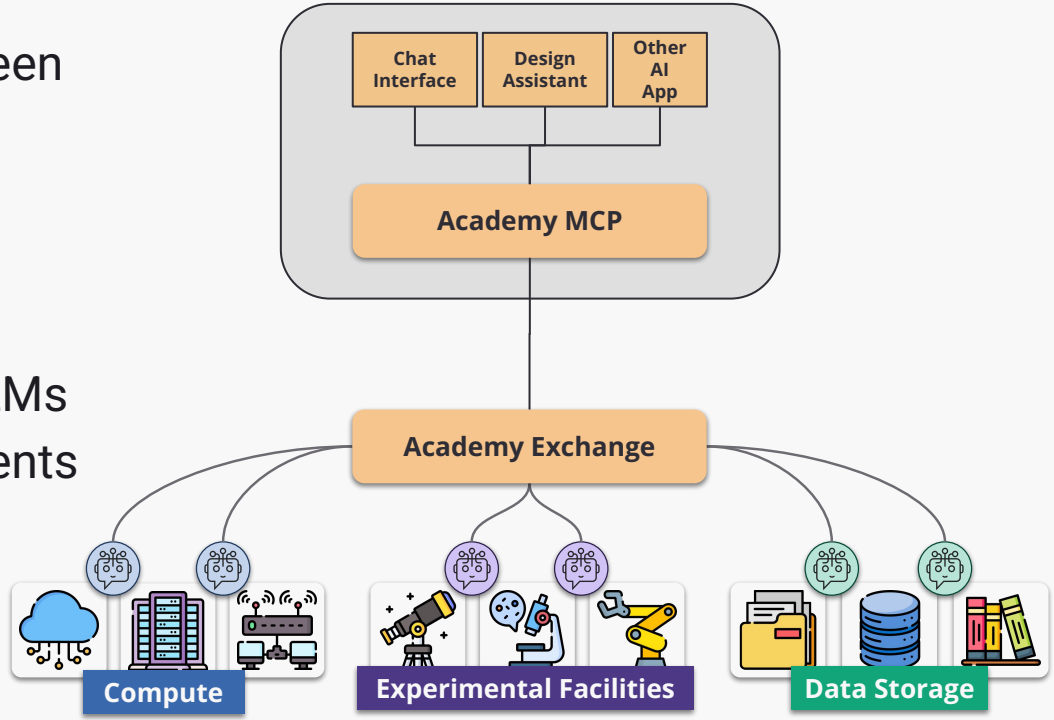
Model Context Protocol (MCP)

Outer world



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



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()
- …

MCP

MCP

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

Academy

SC25
St. Louis, MO | hpc ignites.

Agentic Workflows for Science

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

Agentic Patterns Beyond LLMs

Conversational Assistants

Human

Provide the SMILES string corresponding to this molecule: 1-(2-methylphenyl)-3-(3-methylpyridin-2-yl)urea

ChemGraph

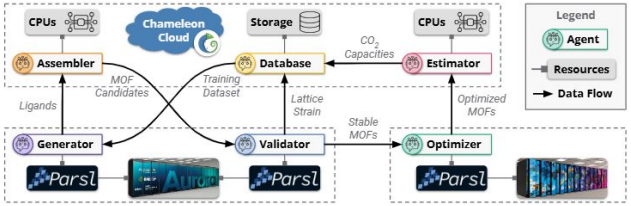
I'll help you find the SMILES string for the molecule 1-(2-methylphenyl)-3-(3-methylpyridin-2-yl)urea. To do this, I'll use the molecule_name_to_smiles function. However, in this case, the input is a systematic chemical name rather than a common molecule name, so the function might not work directly.

[...]

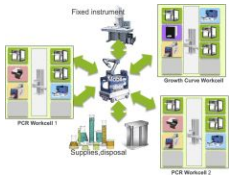
Would you like me to help you find alternative ways to represent this molecule?

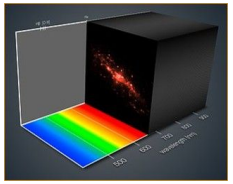
Human

Multi-Site Applications

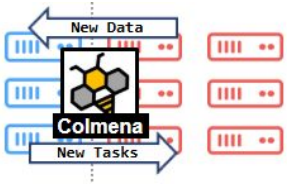


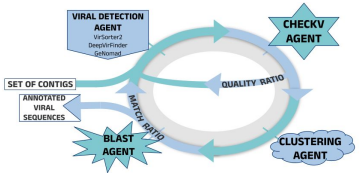
Integrating Instruments/Experiments





Steering Workflows



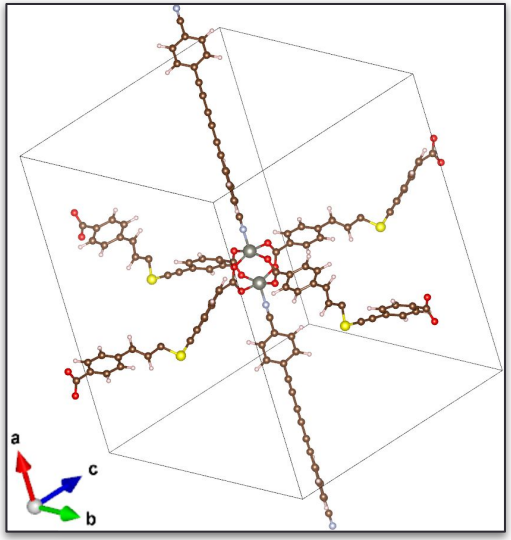


Use Case: **MOF Discovery**

Metal Organic Frameworks (MOF)

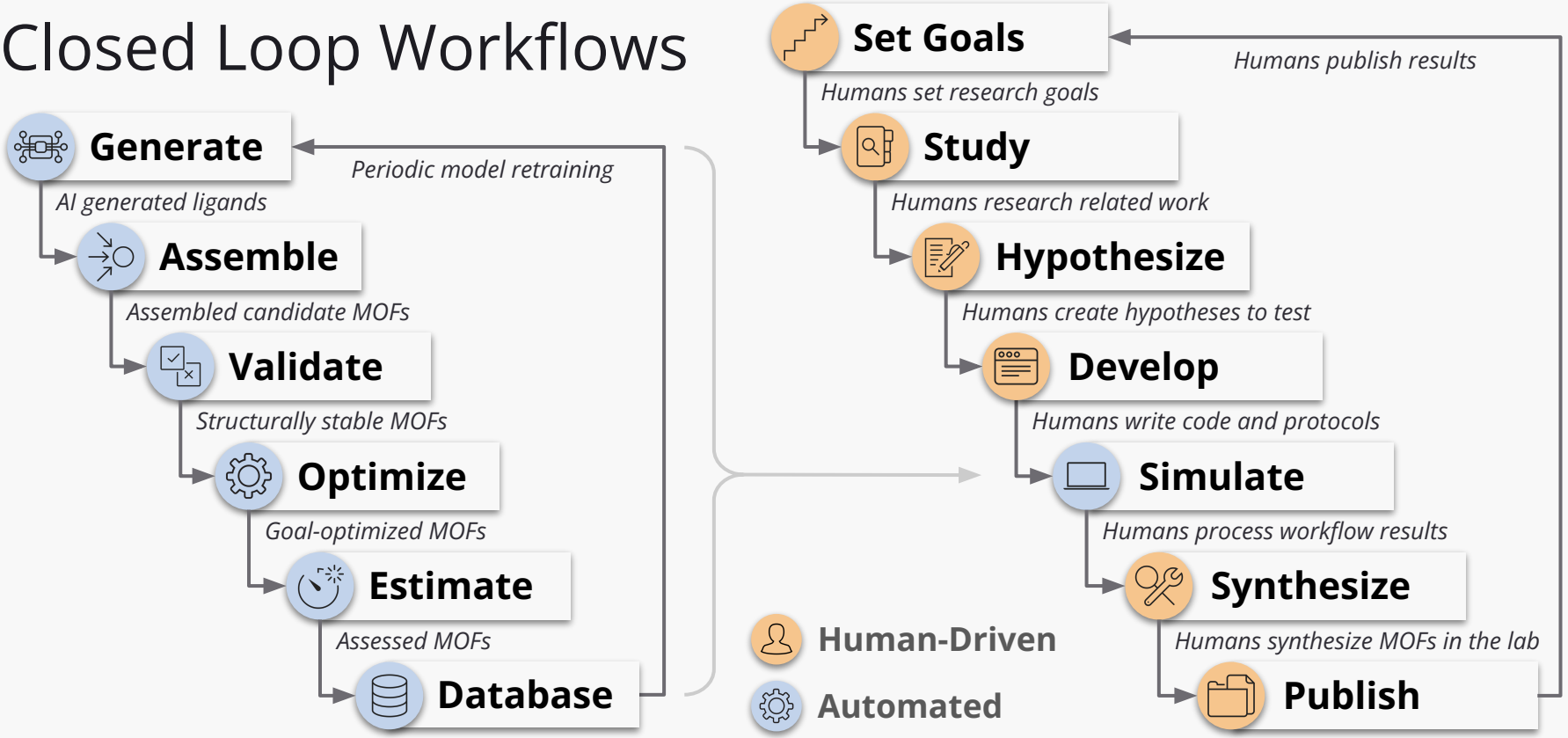
- ➔ Composed of organic molecules (ligands) and inorganic metals (nodes)
- ➔ The sponges of materials science!
- ➔ Porous structures that adsorb and store gases
- ➔ Topologies can be optimized for targeted gas storage → **Carbon Capture**

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



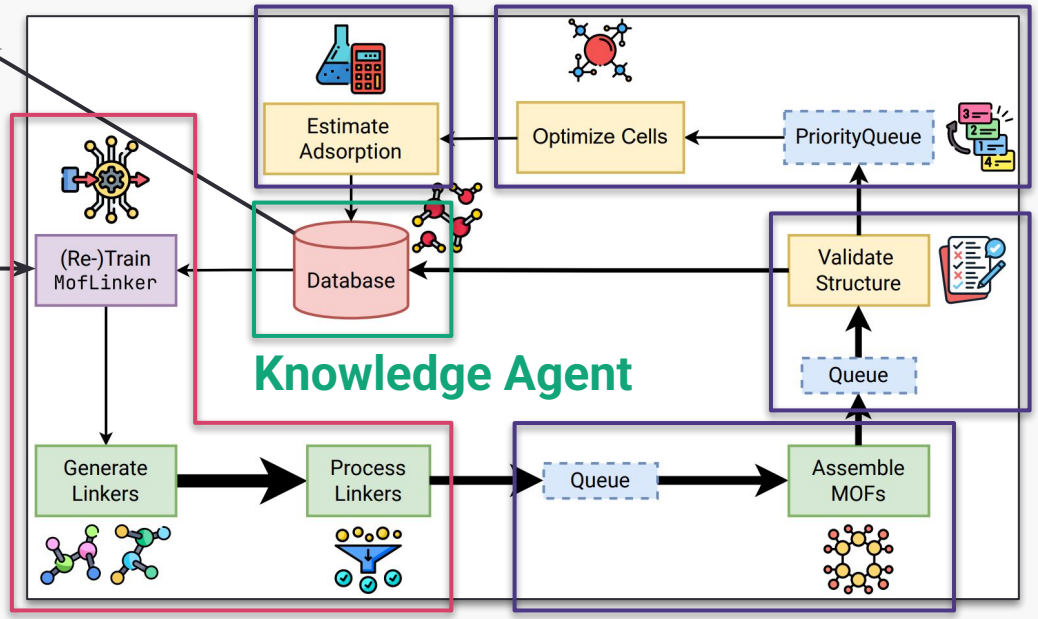
Intractable search space of ligand, node, & geometry combinations

Closed Loop Workflows



MOFA: Online learning + GenAI + Simulation

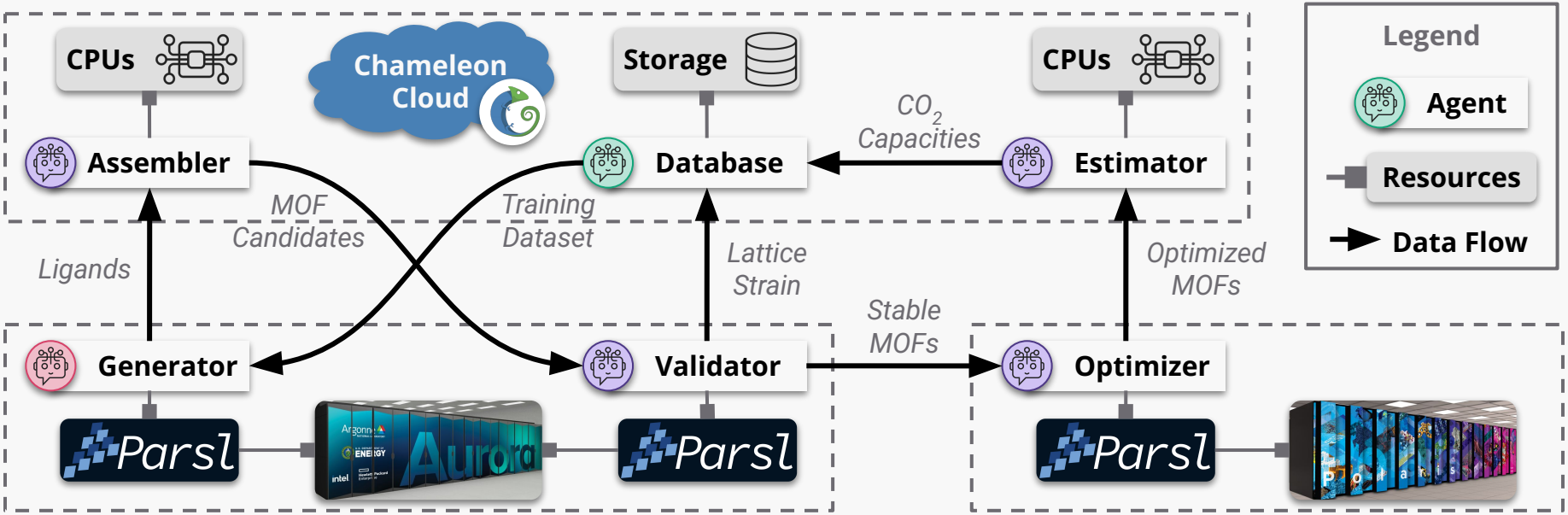
Embodied Agents*



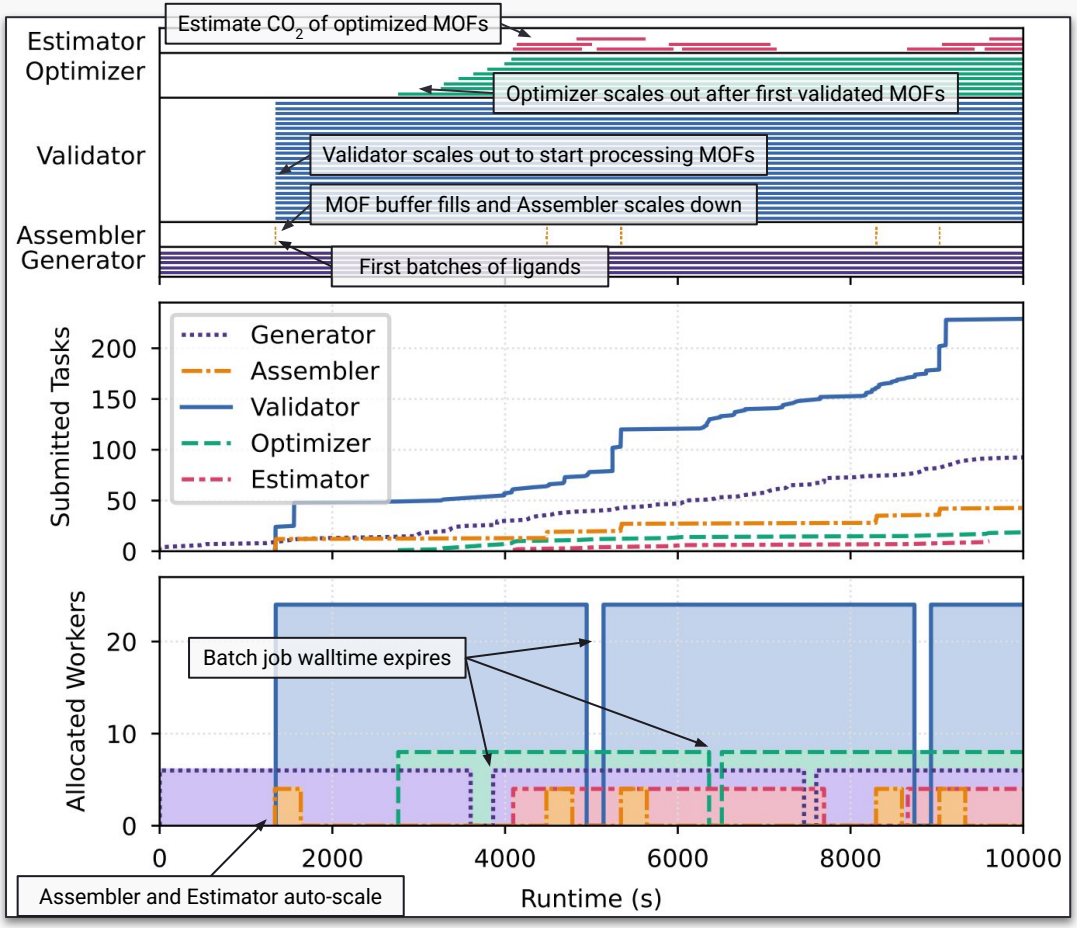
Computational Agents

Yan et al., "**MOFA: Discovering Materials for Carbon Capture with a GenAI- and Simulation-Based Workflow**" (Under Review)

MOFA through Autonomous Agents



Agents executed remotely via Globus Compute



MOFA Agents Trace

Why is this agentic model better?

- ➔ **Placement:** Move agents to resources
- ➔ **Separation of concerns:** Resource acquisition and scaling based on local workload
- ➔ **Loose coupling:** Swap agents or integrate new agents (e.g., SDL)
- ➔ **Shared agents:** Multiple workflows can share agents (microservice-like)

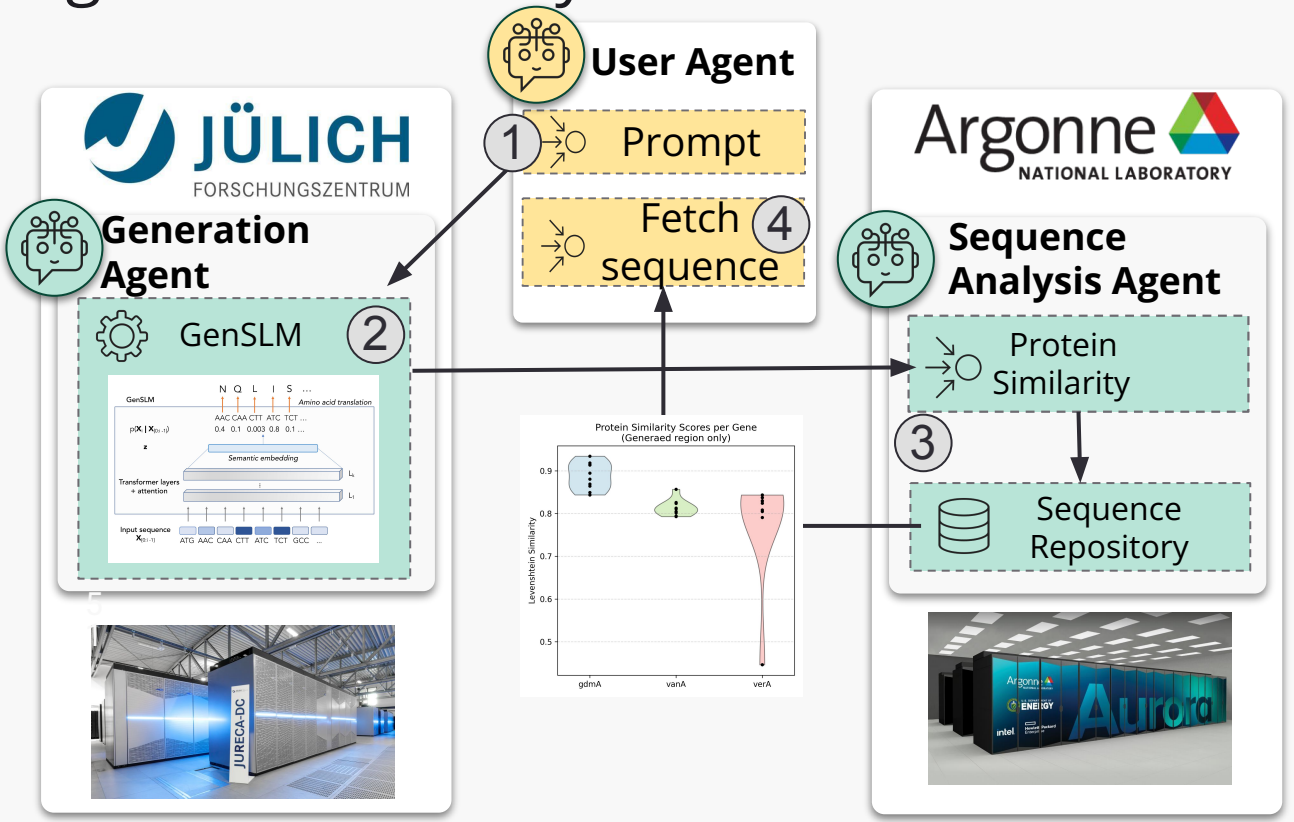
Agentic enzyme design with Academy

Application Credits: Xinran Lian, Alex Brace, Arvind Ramanathan

GenSLM is a genome-scale language model that can generate bacterial and viral protein sequences.

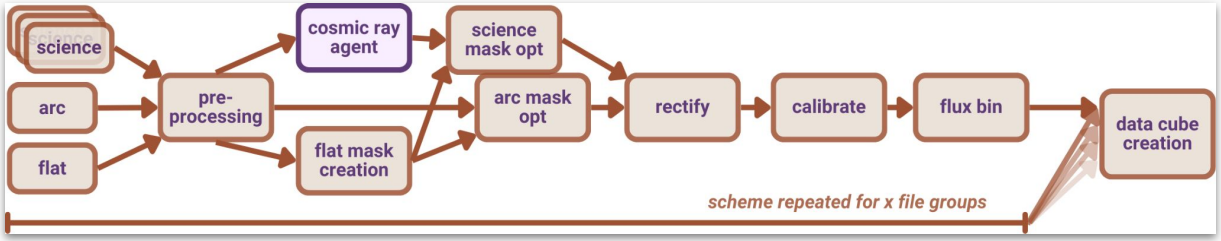
We implement three agents:

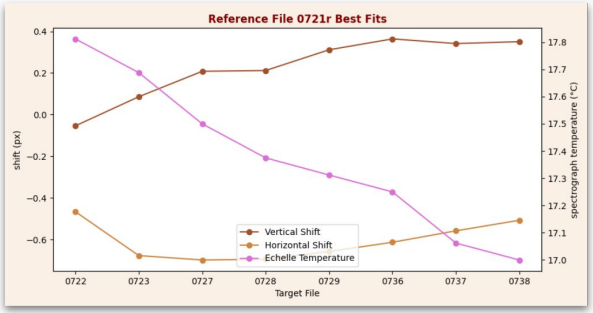
1. **User agent** triggers new analyses
2. **Generation agent**, which runs GenSLM models, trained on proprietary data, to generate sequences
3. **Sequence analysis agent,** which hosts methods for evaluating protein similarity, and stores promising sequences
4. **User agent** (again) monitors sequence repository for promising candidate sequences

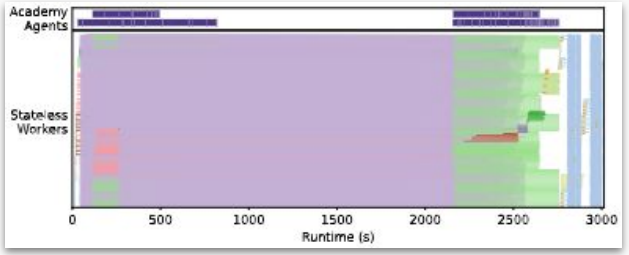


Use Case: **Integral Field Unit Spectroscopy**

- ➔ Highly-sensitive to instrument calibration
- ➔ Optical parameters are **continuous in time**
- ➔ Resolution and speed can be improved using **stateful processing**

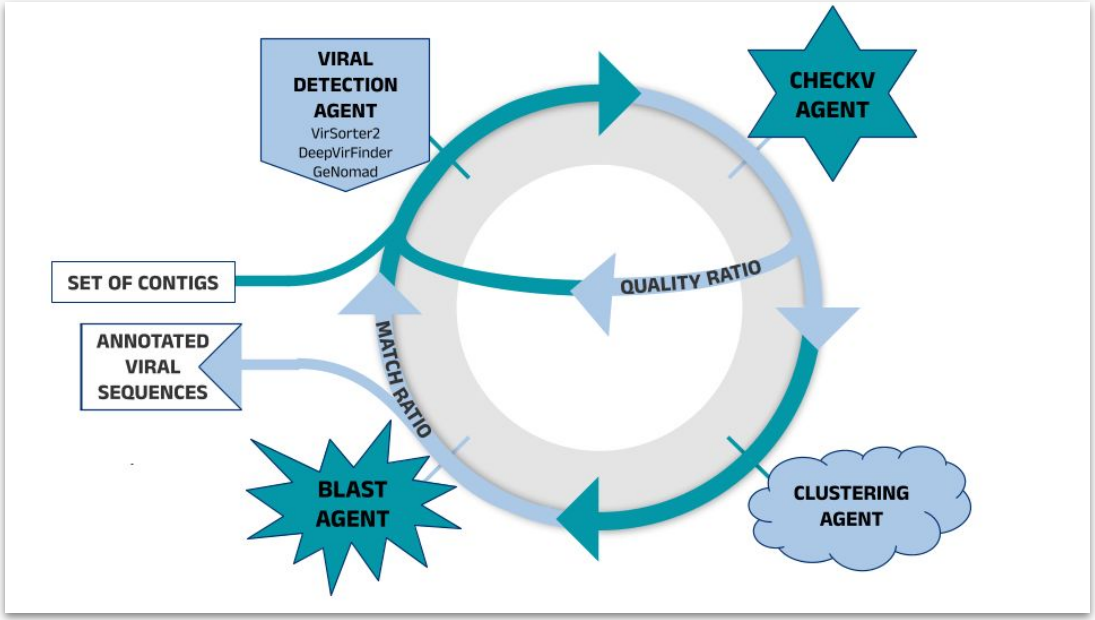






Use Case: **Viral Microgenomics**

- ➔ Multiple tools exist to complete the same tasks (i.e. viral detection)
- ➔ Workflow construction is manual and heuristic
- ➔ Can we learn from downstream metrics to automate workflow design?



The Future? The Scientific Method through Agents

Academy

SC25 St. Louis, MO | hpc ignites.

Wrap up

THE UNIVERSITY OF CHICAGO

globus labs

What's Next for Academy?

- Community and Governance
  - Build an engaged, cross-disciplinary community of researchers and developers
  - Establish a sustainable governance and contribution model
- Scientific Engagement
  - Collaborate with domain science teams to evaluate efficacy, identify challenges, and uncover new opportunities for agents in practice
- Ecosystem Integration
  - Connect with the broader agent, AI, and scientific cyberinfrastructure ecosystem
- Technical Directions
  - Develop a secure, scalable, and robust managed exchange for the agent community
  - Enable integrated tool-calling and remote application execution
  - Strengthen provenance, auditability, and reproducibility across agentic workflows

Join the community / open meeting Wednesdays 1pm CT

Discussion

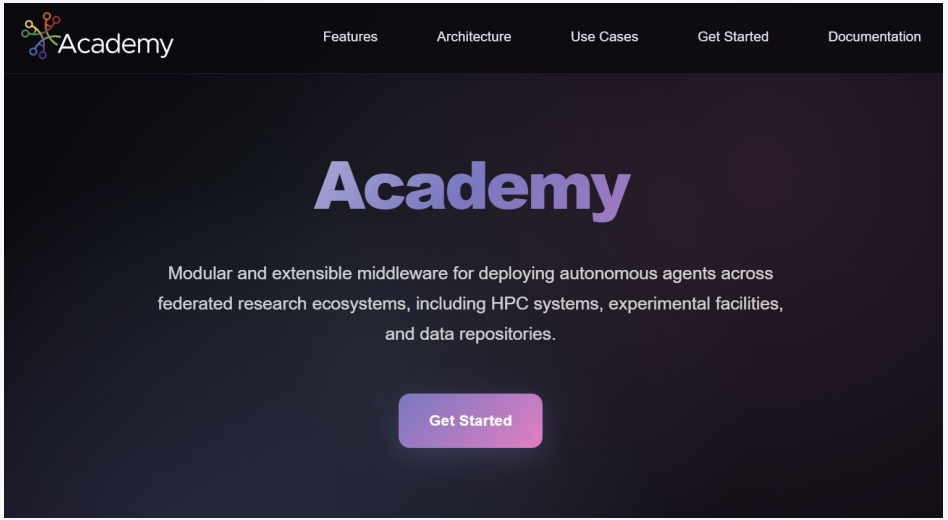
- How are you currently using (or planning to use) agents in your work or research infrastructure?
- What makes scientific and cyberinfrastructure contexts uniquely challenging or interesting for agentic systems?
- What are the key capabilities agents need to effectively support scientific workflows (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



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