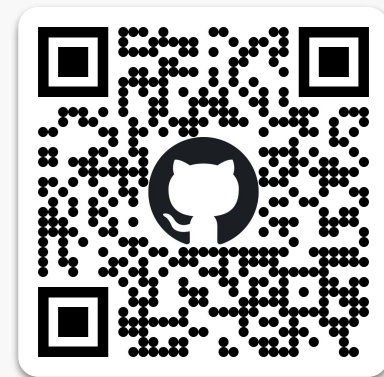




Building Scalable Agentic Systems for Science

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

Alok Kamatar, Kyle Chard

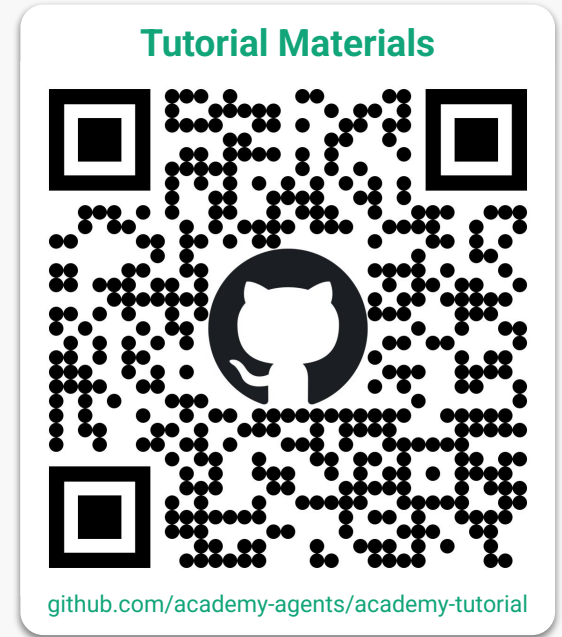


github.com/academy-agents/academy-tutorial

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



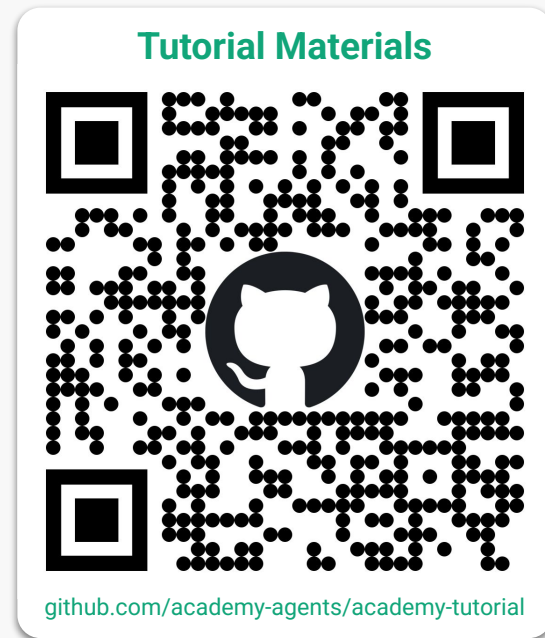
Welcome!

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



Agenda

| Time | Topic |
|---------------|----------------------------------|
| 8:30 - 9:00 | Introduction |
| 9:00 - 9:30 | Academy Overview |
| 9:15 - 10:00 | Hands-on: Getting Started |
| 10:00 - 10:30 | Break |
| 10:30 - 11:00 | Hands-on: Remote agents |
| 11:00 - 11:30 | Hands-on: Battleship |
| 11:30 - 11:45 | Demo: What's next |
| 11:45 - 12:00 | Wrap up |

Room J



Alok Kamatar
UChicago



Kyle Chard
UChicago & ANL



Introduction

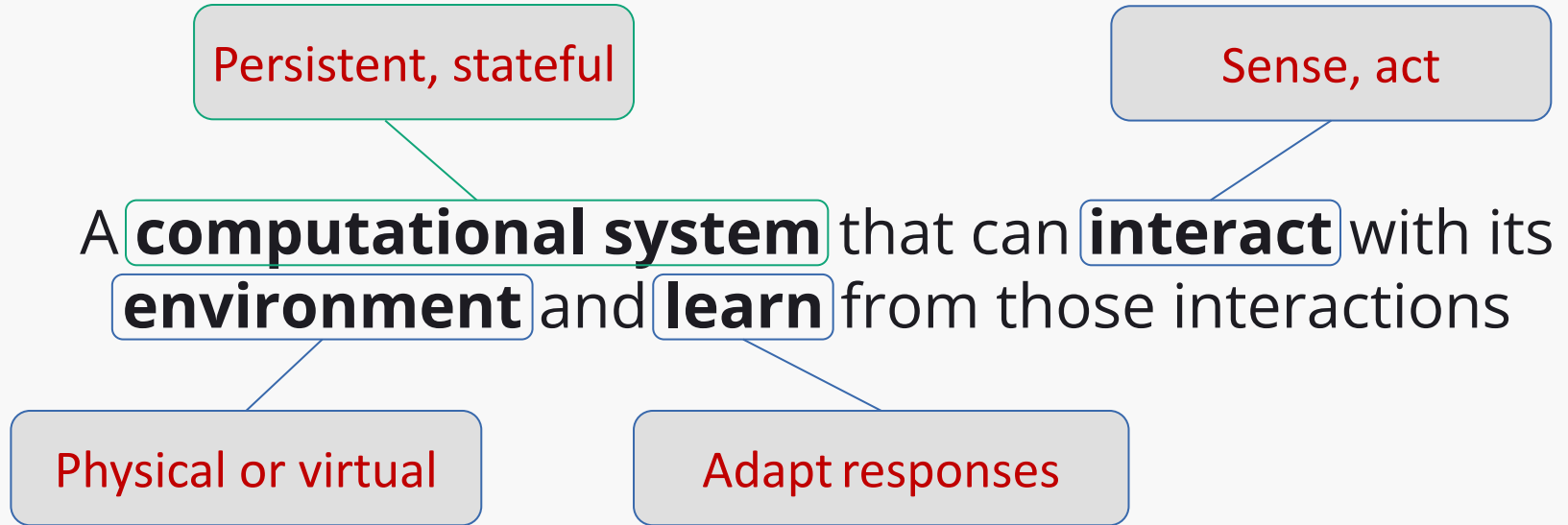
8:30 – 9:00 am

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>

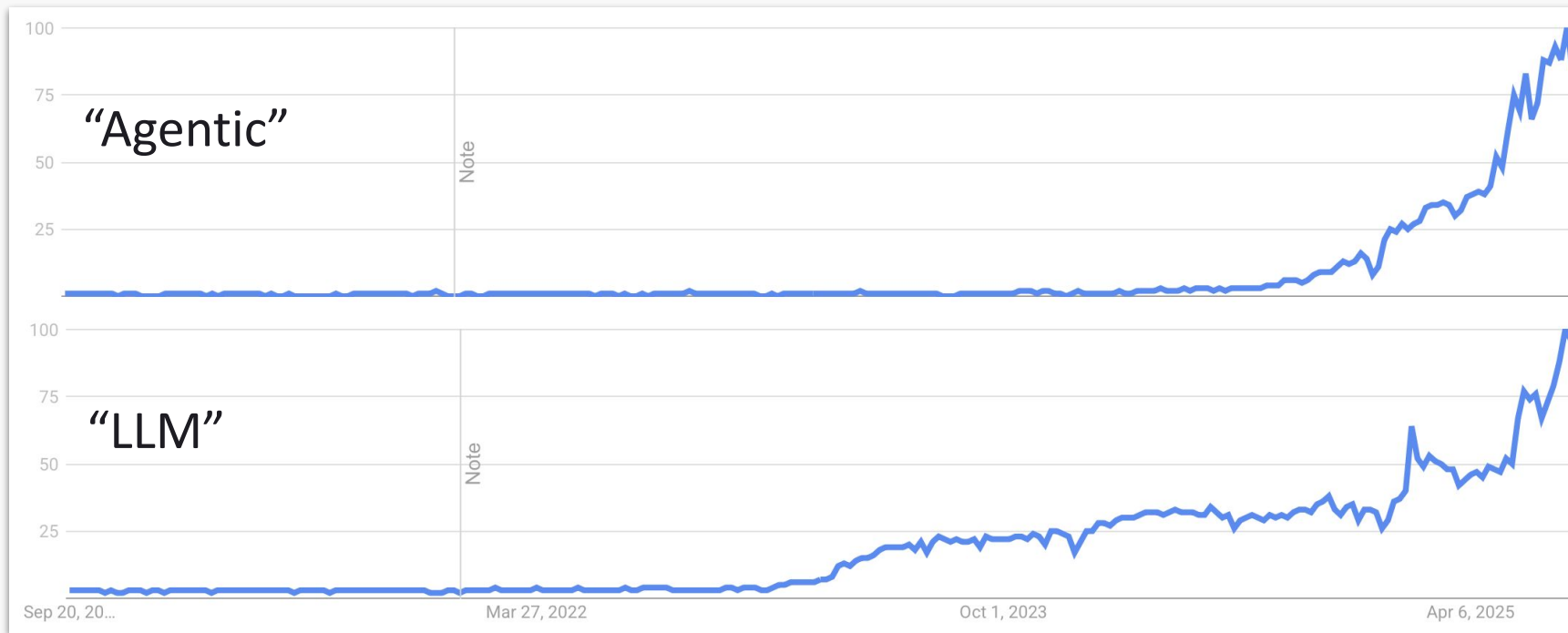
Related concept: An “embodied agent”



The “sense-plan-act-learn” loop

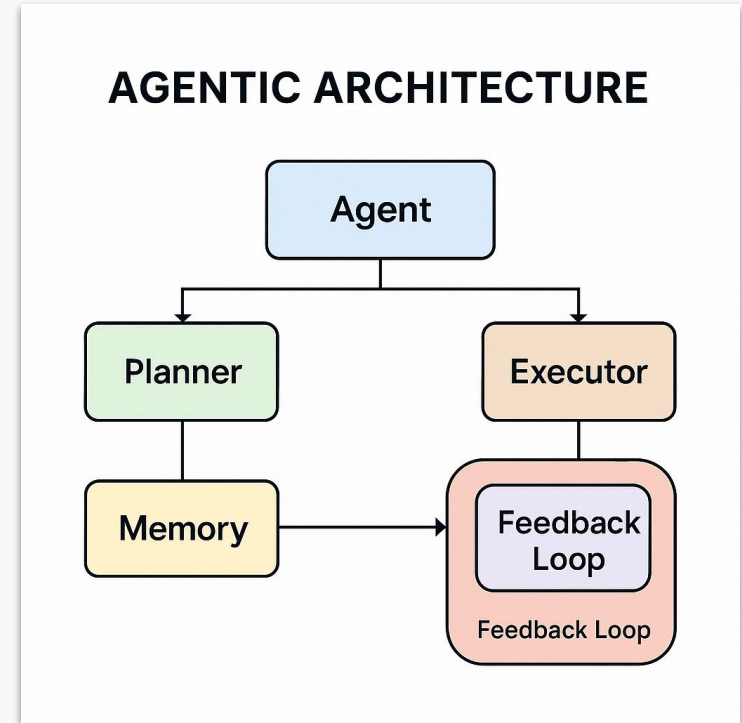
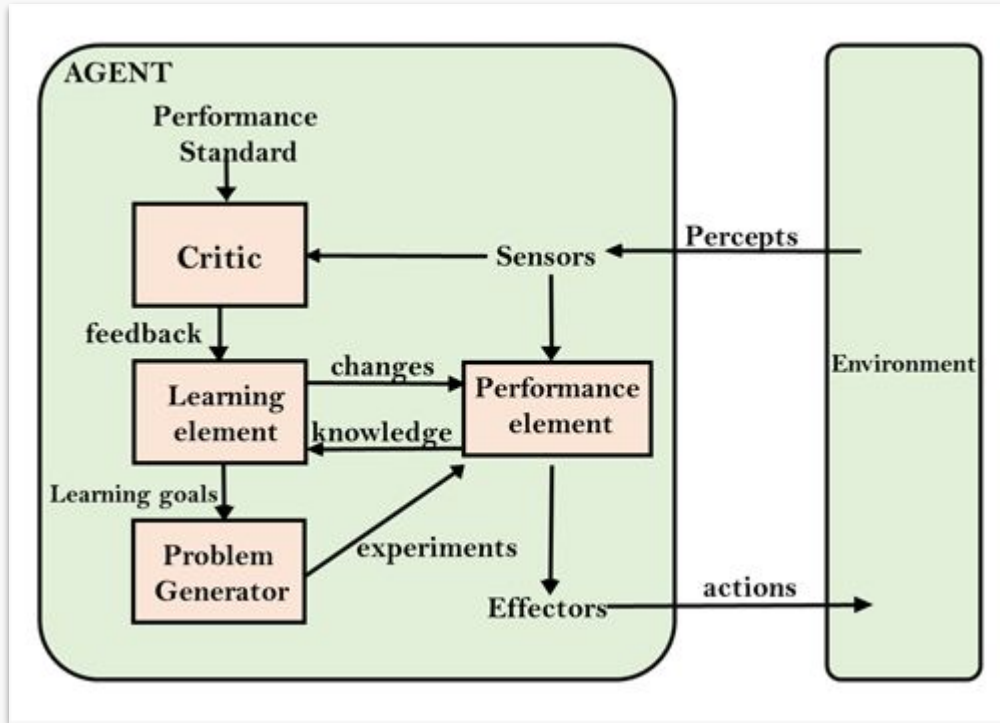
- Initialize state and goals
- Repeat until termination:
 - **Sense:** Gather observations from environment
 - **Plan:** Evaluate goals and state, plan/select next action
 - **Act:** Execute chosen action on environment
 - **Learn:** Update internal state, memory, or model based on outcomes

Current excitement around agents is due to LLMs

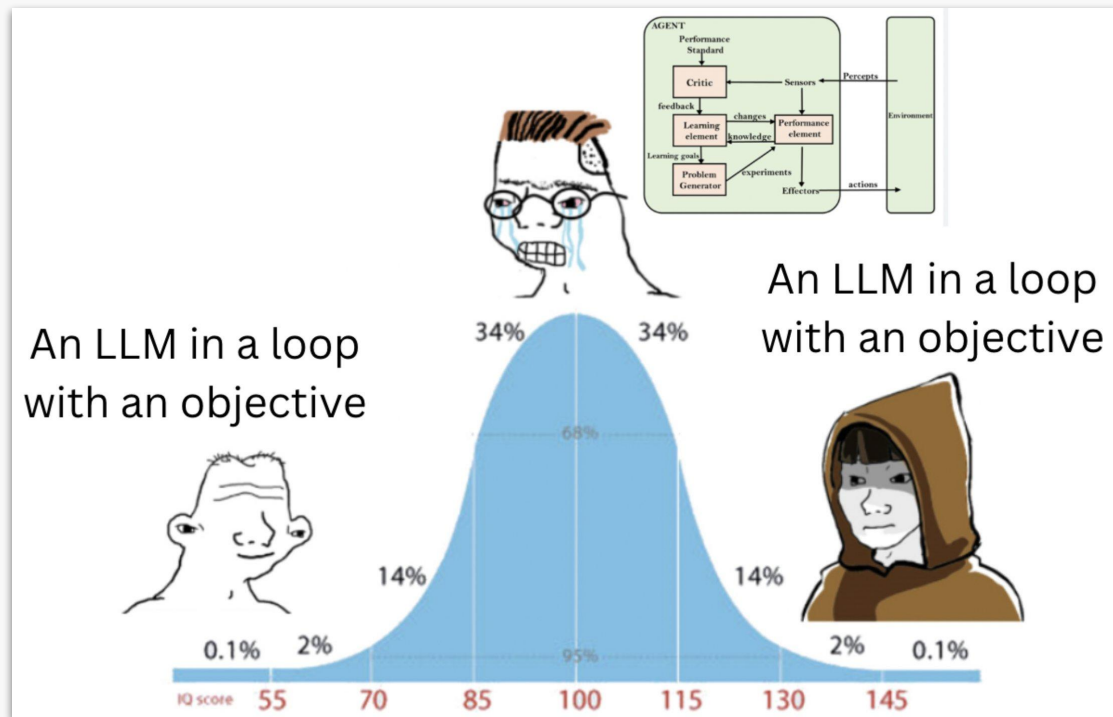


<https://trends.google.com>

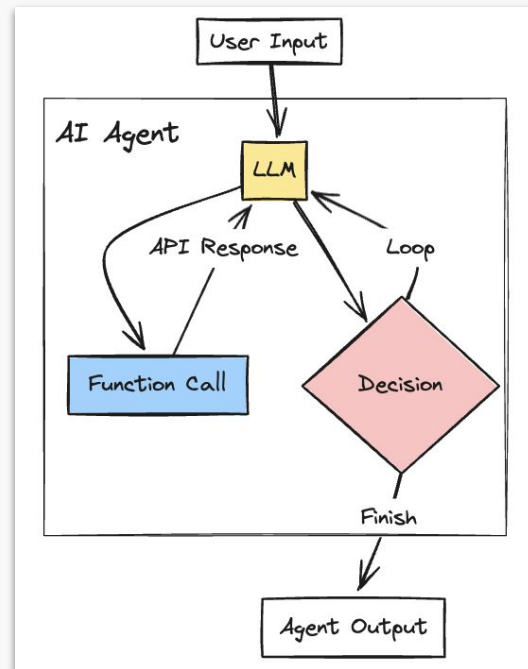
Many related ideas regarding agentic architecture



A simple perspective



https://x.com/josh_bickett/status/1725556267014595032



@SullyOmarr

“CACTUS: Chemistry Agent Connecting Tool Usage to Science”

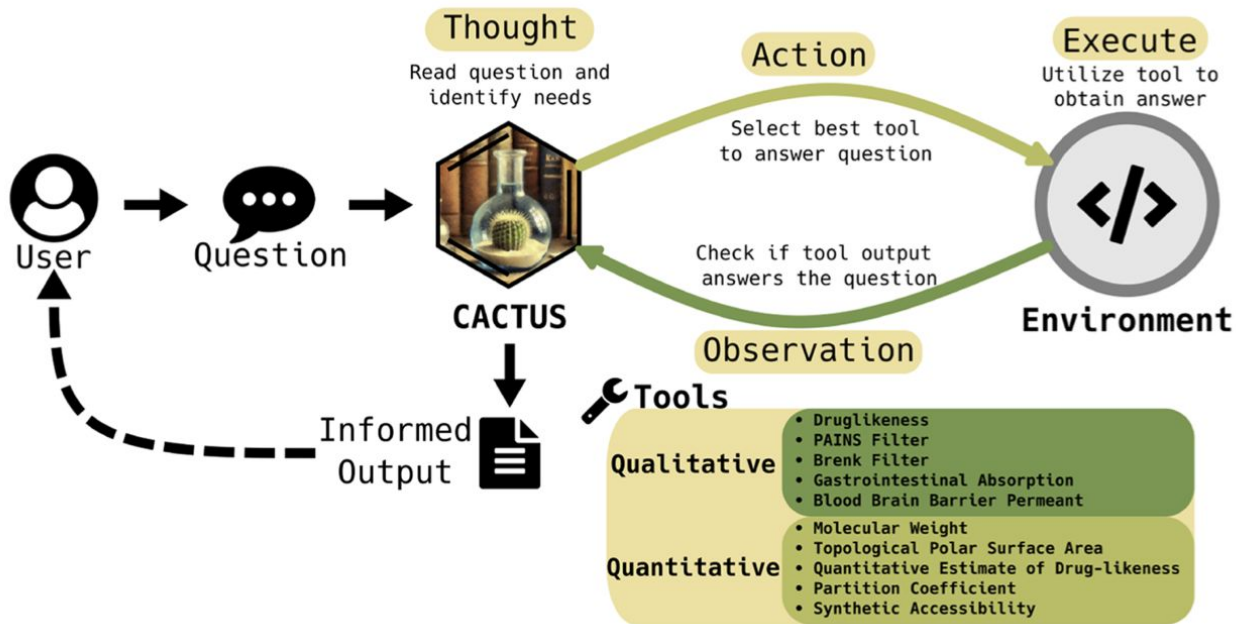


Figure 1. General workflow of the CACTUS agent that details how the LLM interprets input to arrive at the correct tool to use to obtain an answer. Starting from the user input, CACTUS follows a standard “Chain-of-thought” reasoning method with a Planning, Action, Execution, and Observation phase to obtain an informed output.

<https://pubs.acs.org/doi/pdf/10.1021/acsomega.4c08408>

Overview

- What is an “agent”?
- **LLMs, foundation models, reasoning models**
- Agents and scientific discovery



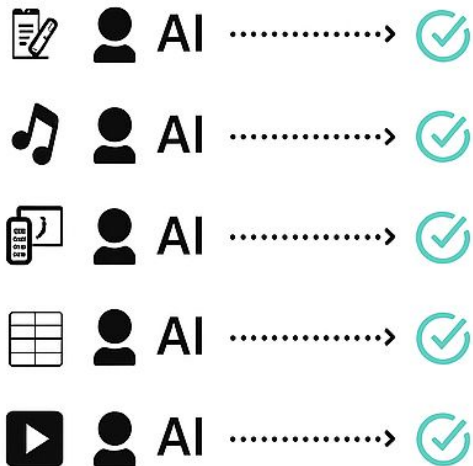
Traditional ML



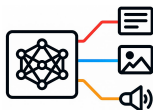
- Individual siloed models
- Require task-specific training
- Lots of human supervised training



Traditional ML

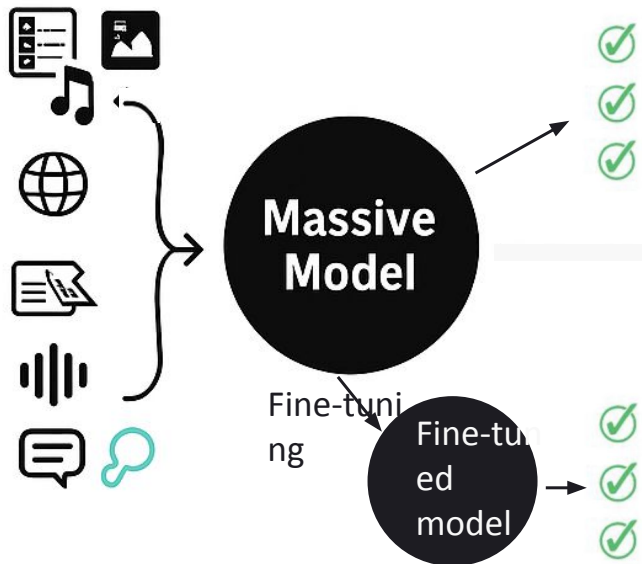


- Individual siloed models
- Require task-specific training
- Lots of human supervised training



Foundation models

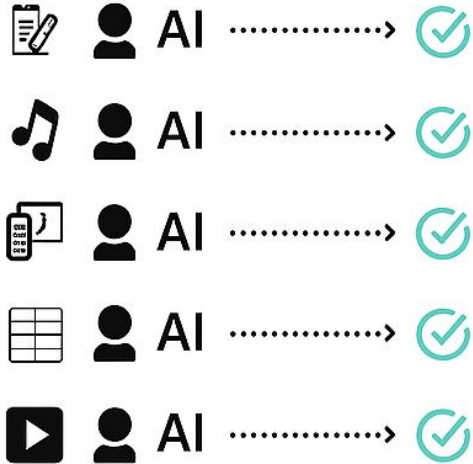
(E.g., Large Language Models: LLMs)



- Massive multi-modal model
- Adapted with minimal training
- Pre-trained unsupervised learning

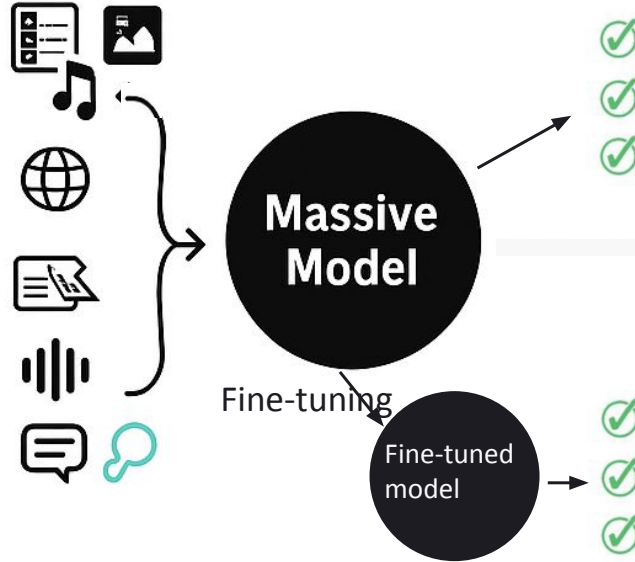


Traditional ML

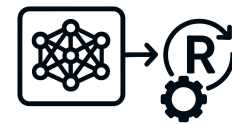


Foundation models

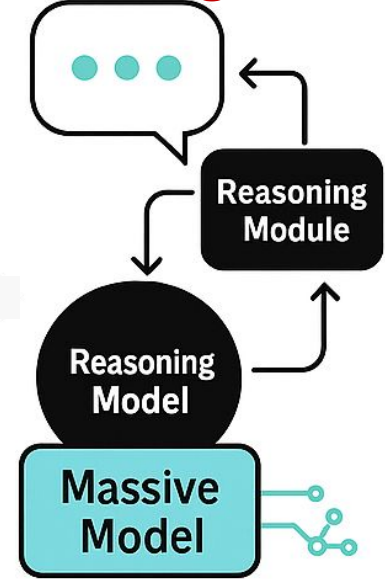
(E.g., Large Language Models: LLMs)



- Massive multi-modal model
- Adapted with minimal training
- Pre-trained unsupervised learning



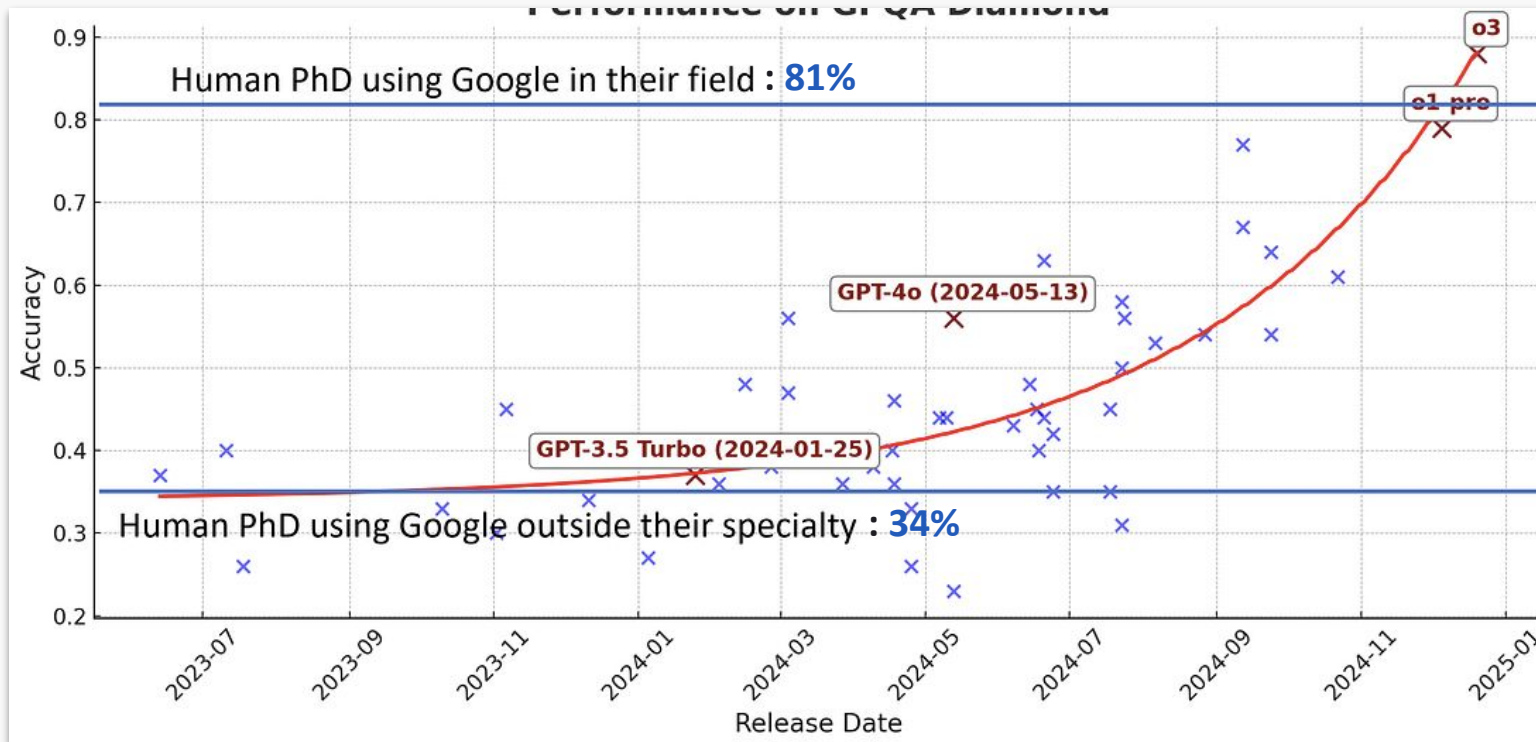
Reasoning models



- Deliberative multi-step logic
- Self-consistency checks
- Slower but more accurate inference

- Individual siloed models
- Require task-specific training
- Lots of human supervised training

Graduate-Level Google-Proof Q&A test (GPQA), Diamond problems



<https://arxiv.org/pdf/2311.12022>

<https://epoch.ai/data/ai-benchmarking-dashboard>

Overview

- What is an “agent”?
- LLMs, foundation models, reasoning models
- **Agents and scientific discovery**

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

Agents can be the driving entities

→ Persistent, stateful, cooperative

→ Intermittent human oversight

Inefficient use of
research infrastructure

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

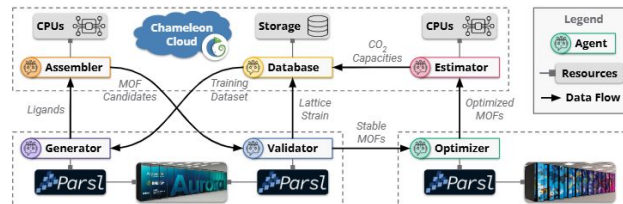
I'll help you find the SMILES string for the molecule 1-(2-methylphenyl)-3-(3-methylpyridin-2-yl)urea. To do this, I'll use the molecule_name_to_smiles function. However, in this case, the input is a systematic chemical name rather than a common molecule name, so the function might not work directly.

[...]

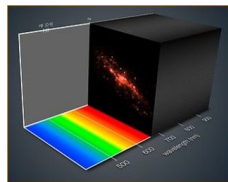
Would you like me to help you find alternative ways to represent this molecule?

Human

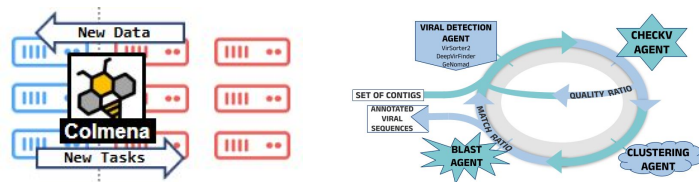
Multi-Site Applications



Integrating Instruments/Experiments



Steering Workflows

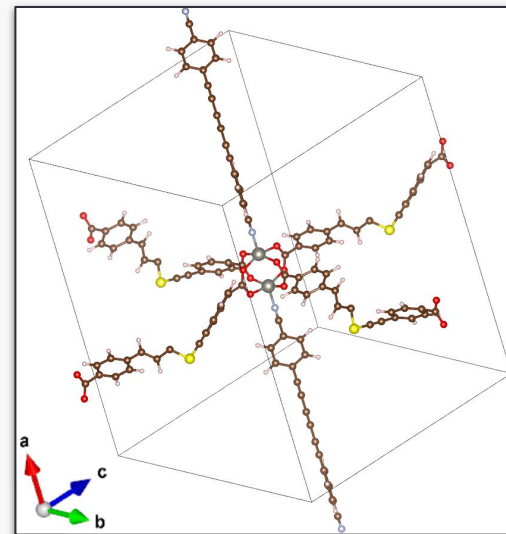


Use Case: MOF Discovery

Metal Organic Frameworks (MOF)

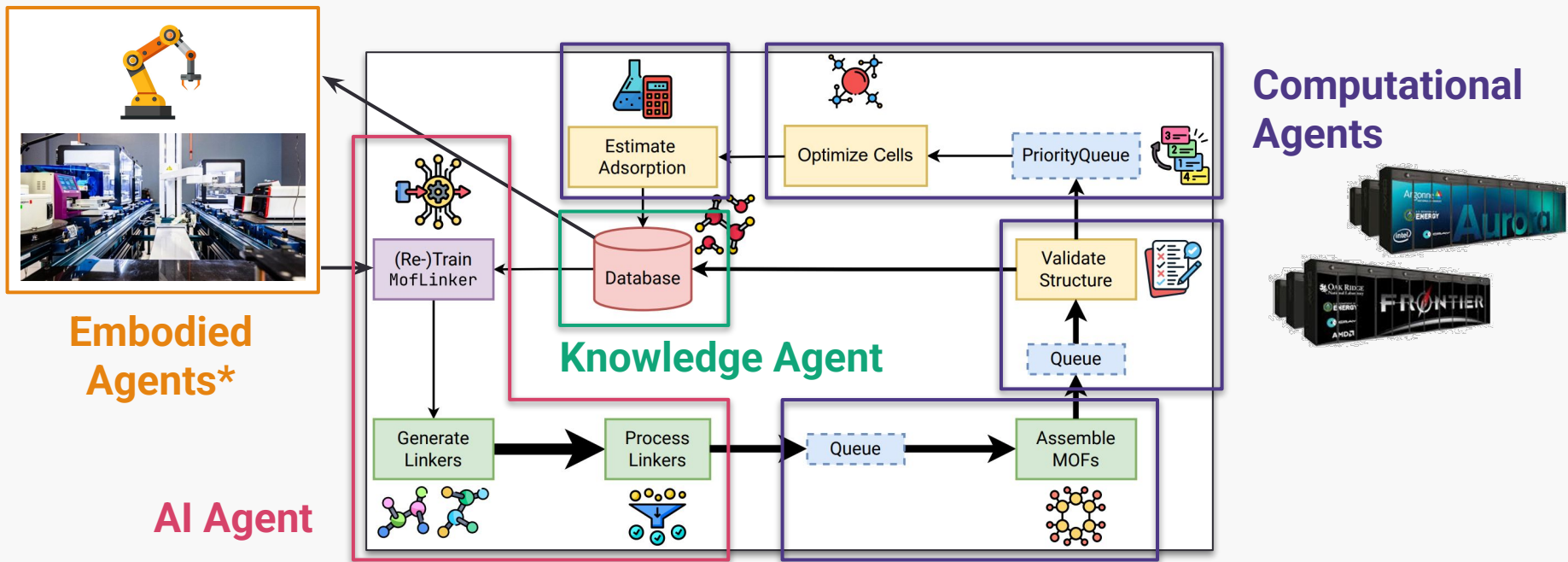
- Composed of organic molecules (ligands) and inorganic metals (nodes)
- The sponges of materials science!
- Porous structures that adsorb and store gases
- Topologies can be optimized for targeted gas storage → **Carbon Capture**

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



Intractable search space of ligand, node, & geometry combinations

MOFA: Online learning + GenAI + Simulation



Embodied Agents*

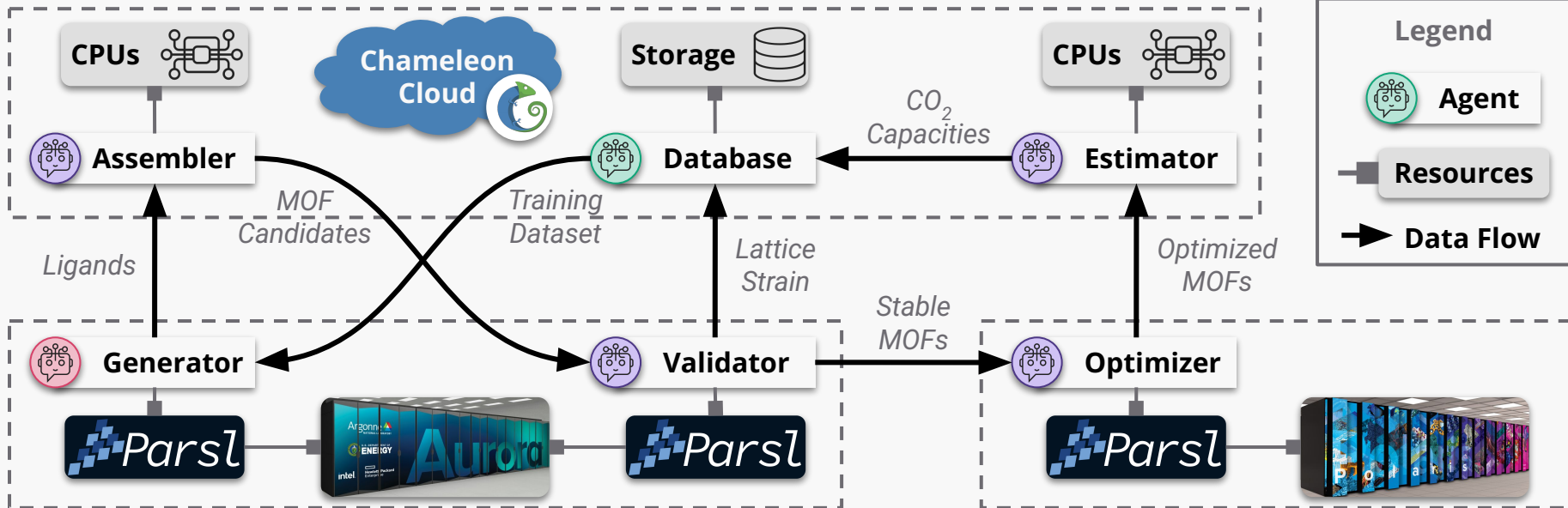
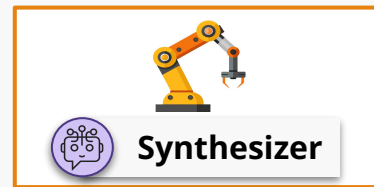
AI Agent

Computational Agents



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

MOFA through Autonomous Agents

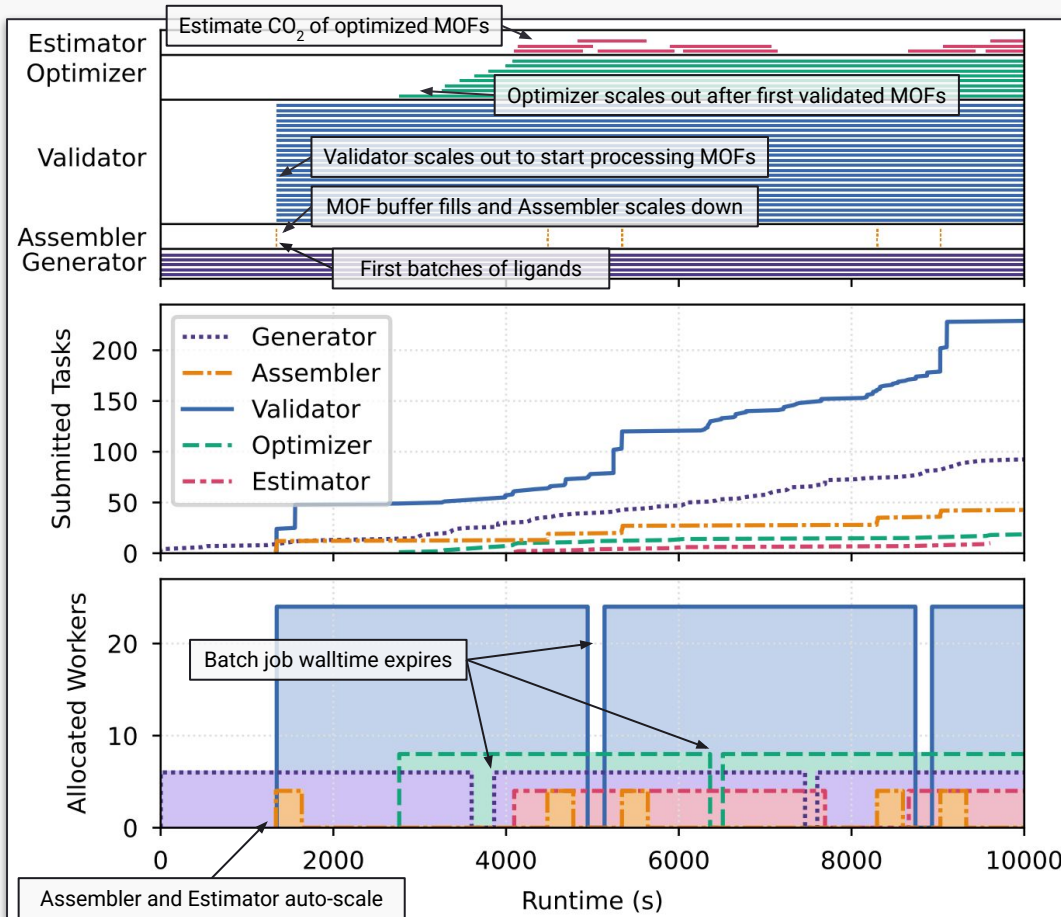


Agents executed remotely via Globus Compute

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)



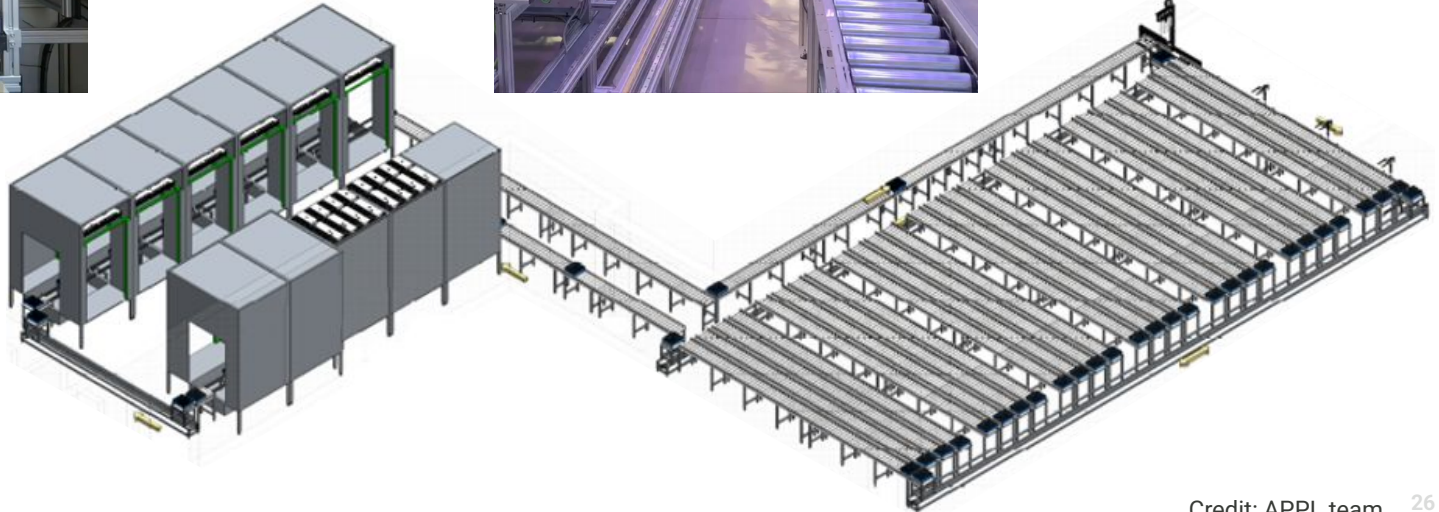
APPL is an automated 520-tray conveyor system that continually monitors plants using multimodal imaging.



Buffer

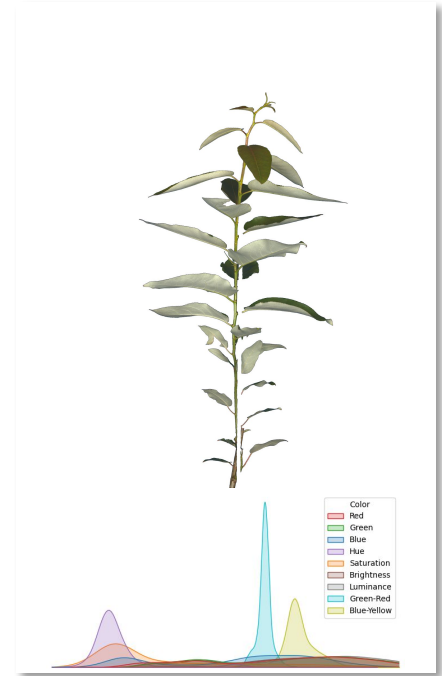
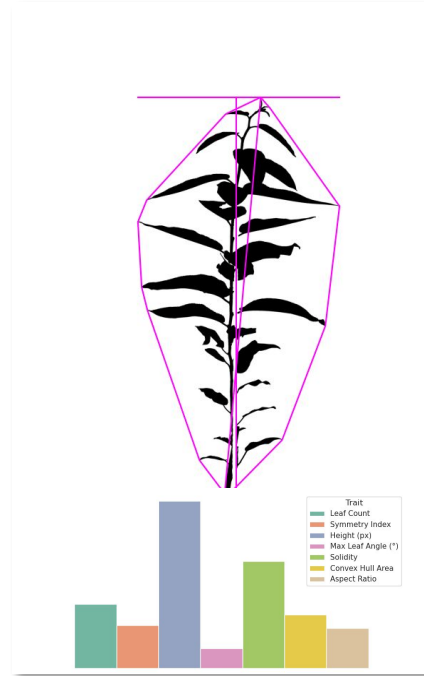
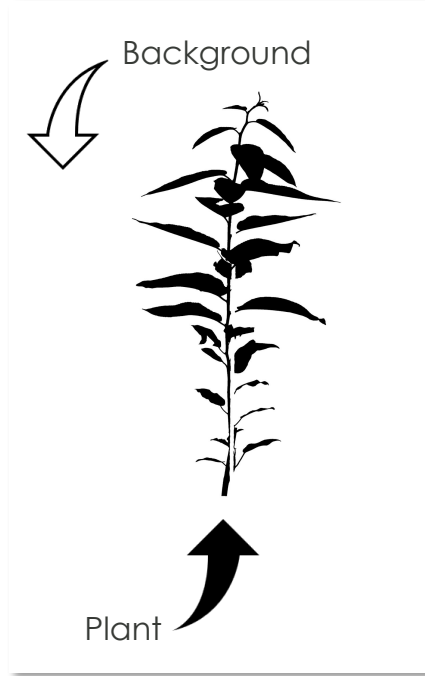


Imaging loop



Accurate image segmentation enables prediction of complex plant phenotypes

Original → Segmentation → Morphology → Spectral traits



Scientist



Daniel Rosendo



Renan Souza



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

OLCF

S3M

ViT inference
Parsl

Compute Agent
Academy

Compute node

Login node

Lustre filesystem



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



REGIONAL OFFICE FOR Africa

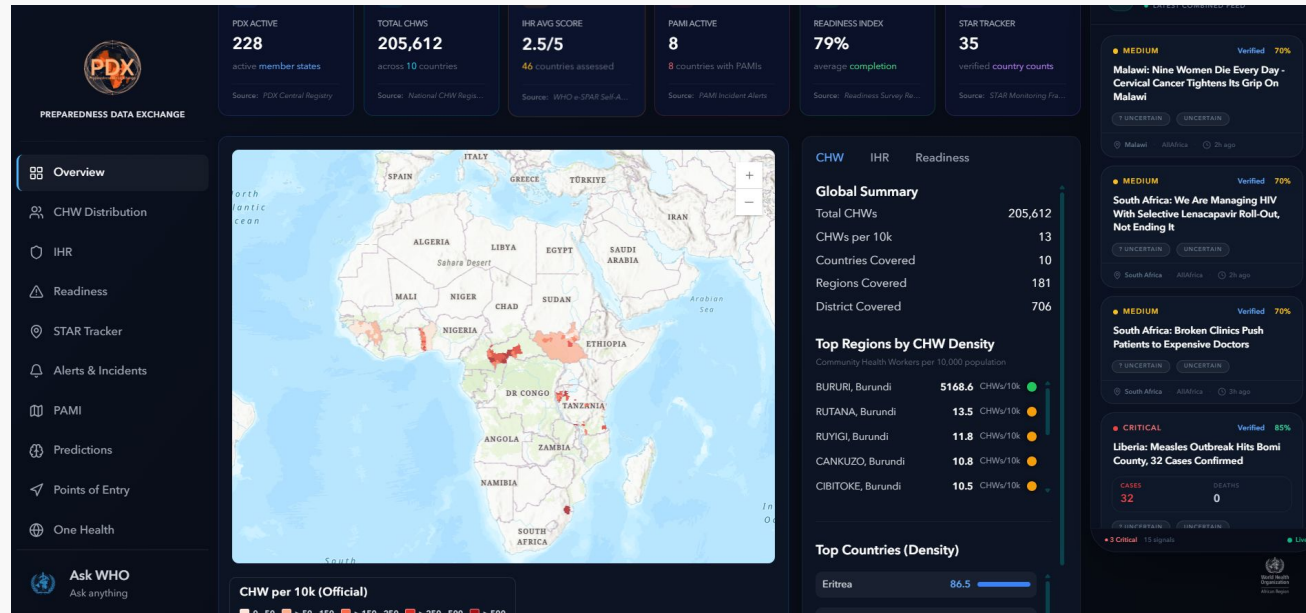
Credit: Isaias Fernandes

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)

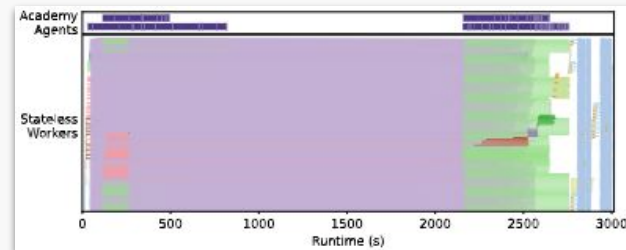
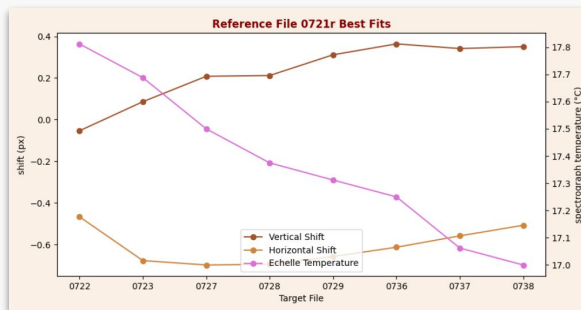
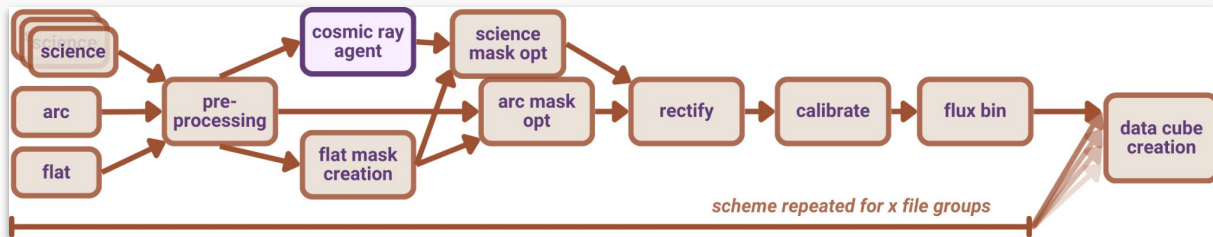
Enables **proactive response and coordination** through automated insights and alerts

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



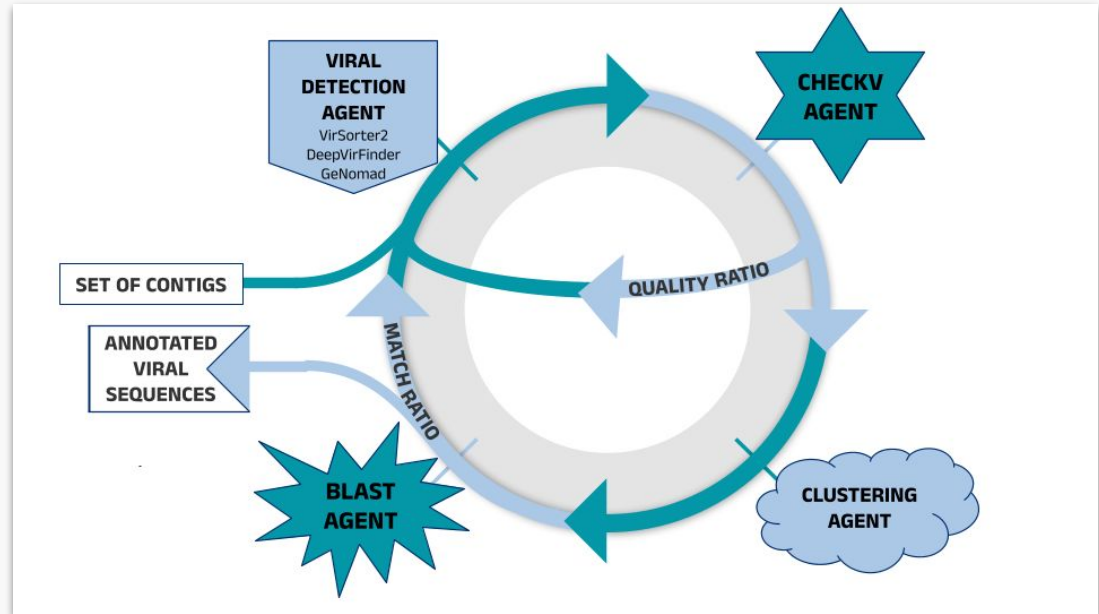
Use Case: Integral Field Unit Spectroscopy

- 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



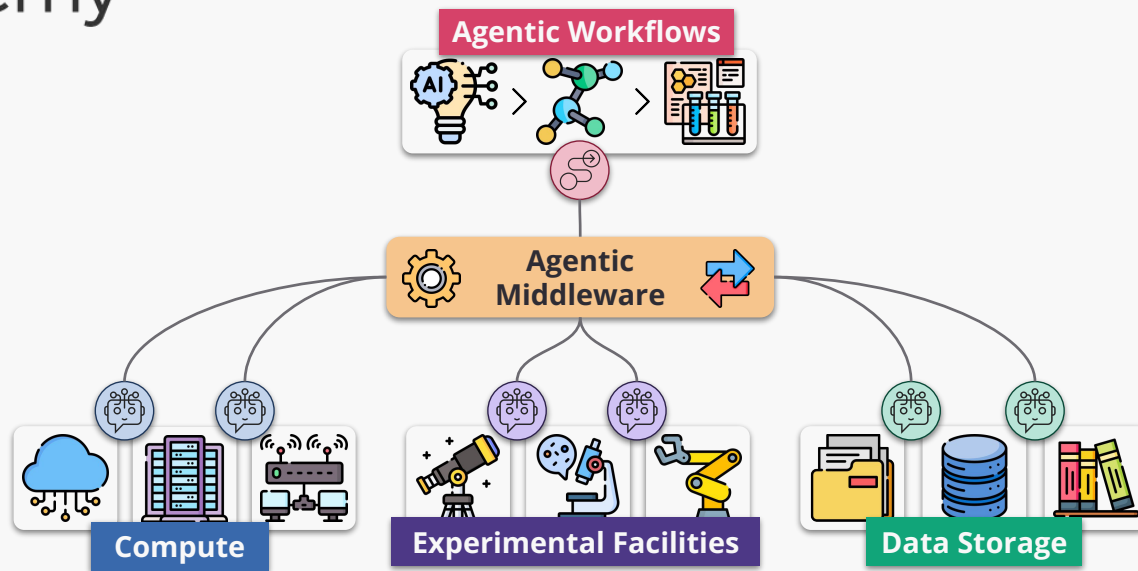
Questions?

Up next → Academy Overview



Academy Overview

9:00 – 9:30 am

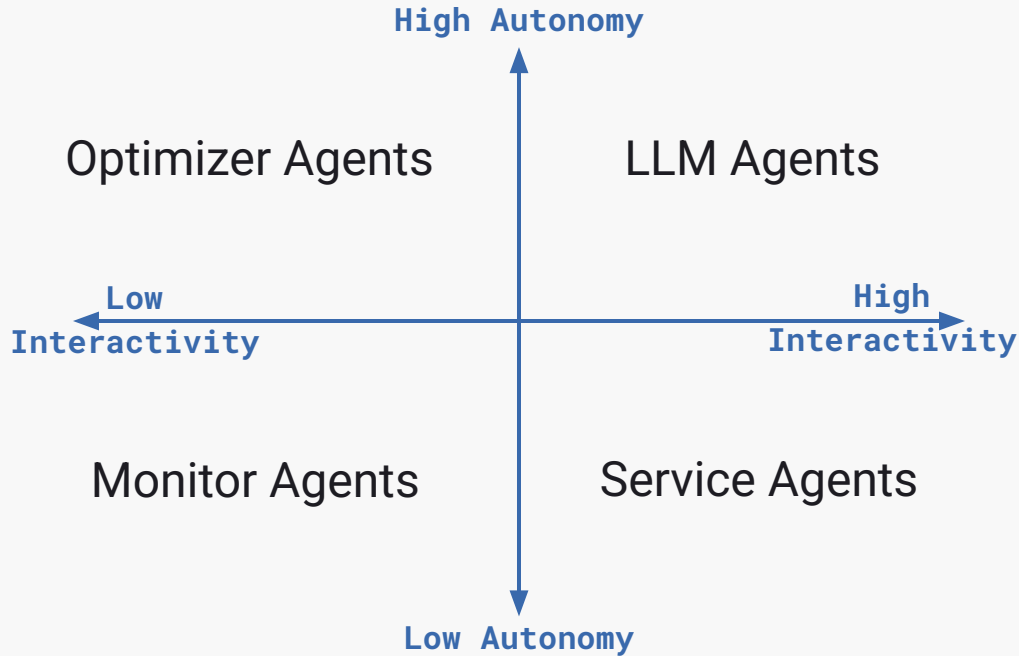


A middleware for building and deploying stateful actors and autonomous agents across distributed research infrastructure

Agentic Middleware

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

Agentic Middleware: Agent Behaviors



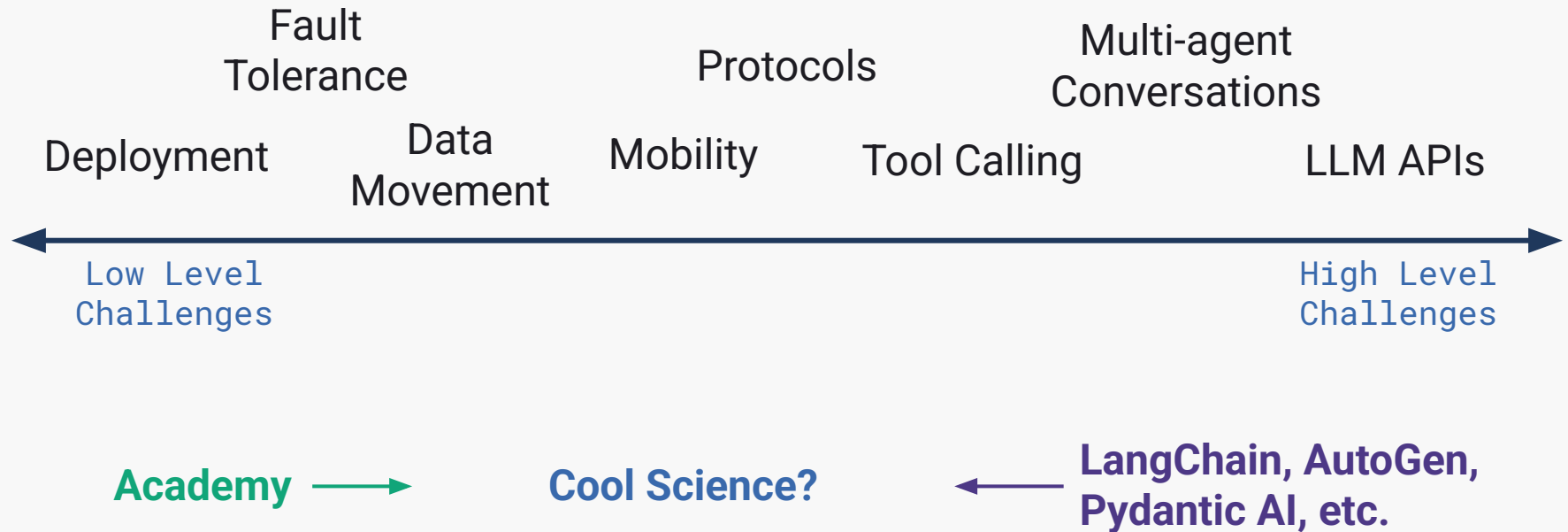
Other defining aspects:

- Persistent vs ephemeral
- General vs narrow purpose
- Embodiment

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

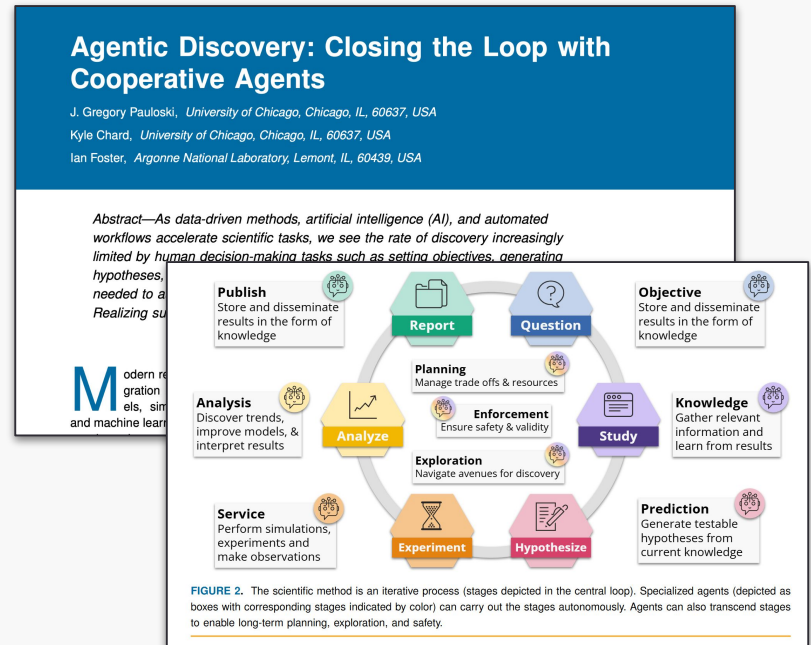
Academy primitives support the creation diverse agent types.

Agentic Middleware: Scope & Challenges



Agentic Middleware: Scope & Challenges

- Access & privileges
- Agent discovery
- Asynchronous communication
- Fault tolerance
- Interfaces ← *Initial focus areas*
- Mobility
- Persistent stateful execution
- Provenance
- Many more...



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

Inspiration

Workflows

Dask, Parsl, Pegasus

- ++ Task automation
- ++ Distributed task execution

Actor Systems

Akka, Dask, Ray

- ++ Stateful computation
- ++ Actor-to-actor interaction

Function-as-a-Service

Globus Compute, Lambda

- ++ Remote execution
- ++ Fire-and-forget model

Academy

- *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

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

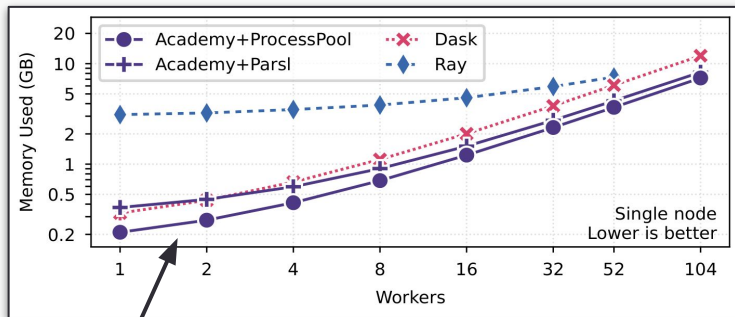
- Http service authenticated with Globus
- Secure multi-site applications

Features (rapid fire)

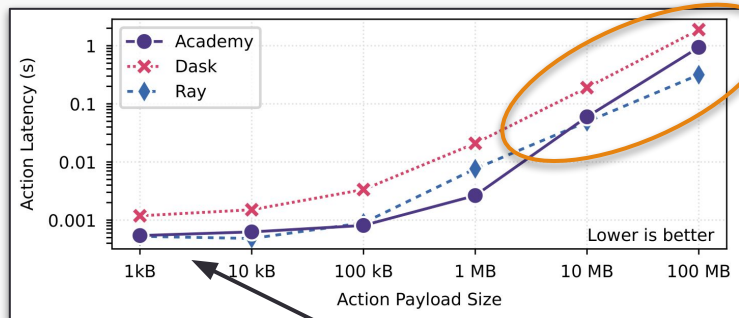
- Any number of actions & control loops
- Special purpose control loop decorators
- Async/non-blocking action execution
- Startup and shutdown callbacks
- State persistence plugins
- Re-execution on failure
- Agents can launch other agents
- Discovery/lookup based on behavior

Comparisons to Actor Systems

Why we need ProxyStore!

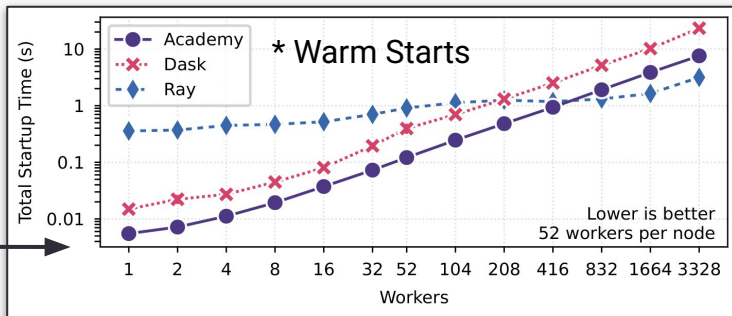


Low-memory overhead



Low-latency messaging

Fast start-up



Experiments performed on Aurora @ ALCF

Writing Apps in Academy

Agents defined
by a class

```
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)
```

Clients & other
agents can
request actions

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

```
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:
```

Launch agent
and get handle

```
behavior = Example() # From the prior slide
handle = await manager.launch(behavior)
```

Interact with
agents via
handles

```
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

```
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

```
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



Hands-on: Getting Started

9:30 – 10:00 am

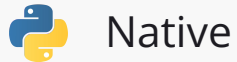
Setup

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 .
```



Conda

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



UV

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

Your First Academy Application

Goal: Write an Counter Agent

- **State:** Count
- **Actions:** Increment, Get_Count
- **Tips:**
 - Use `LocalExchangeFactory` for communication

Define an agent class with state

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

class Example(Agent):
    count: int

    async def agent_on_startup(self) -> None:
        self.count = 0

    @action
    async def increment(self, value: int = 1) -> None:
        self.count += value

    @action
    async def get_count(self) -> int:
        return self.count
```

Actions define methods that can be invoked over the exchange

State can be initialized on `__init__` or with `on_startup` method

Manage the launch and communication of agents

```
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

Adding a Control Loop

Goal: Add “autonomous” behavior to agents

Change increment to count every second

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

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

Agent Cooperation

Goal: Call agent internally from another agent

- Create a Coordinator Agent
 - **Actions:** process: lowers a string
 - Use **Lowerer** to accomplish task

Handles bind to
mailboxes
based on
context

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

class Coordinator(Agent):
    def __init__(self, lowerer, reverser):
        self.lowerer = lowerer
        self.reverser = reverser

    @action
    async def process(self, text: str) -> None:
        reversed = await self.reverser(text)
        return await self.lowerer(reversed)
```

Pass Agent
handles

Agents can
communicate
via the
exchange too!

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

async with await Manager.from_exchange_factory(
    factory=LocalExchangeFactory(),
    executors=ThreadPoolExecutor(),
) as manager:
    ...
    coordinator = await manager.launch(
        Coordinator,
        args=(lowerer, reverser),
    )
    ...
    result = await coordinator.process(text)
    print(result)
```

Agent initialized
with handle

LLM Integration

Academy interoperates with existing LLM wrappers

Goal: Use LangChain + Academy to build a team of distributed agents

- Tell the LangChain agent about Academy actions as **tools**
- Create a complete communication network between launched agents

Actions can be passed as tools to LLMs

```
from langchain import create_agent

...
self.agent = create_agent(
    model=self.model,
    tools=[self.share_information,],
)
...

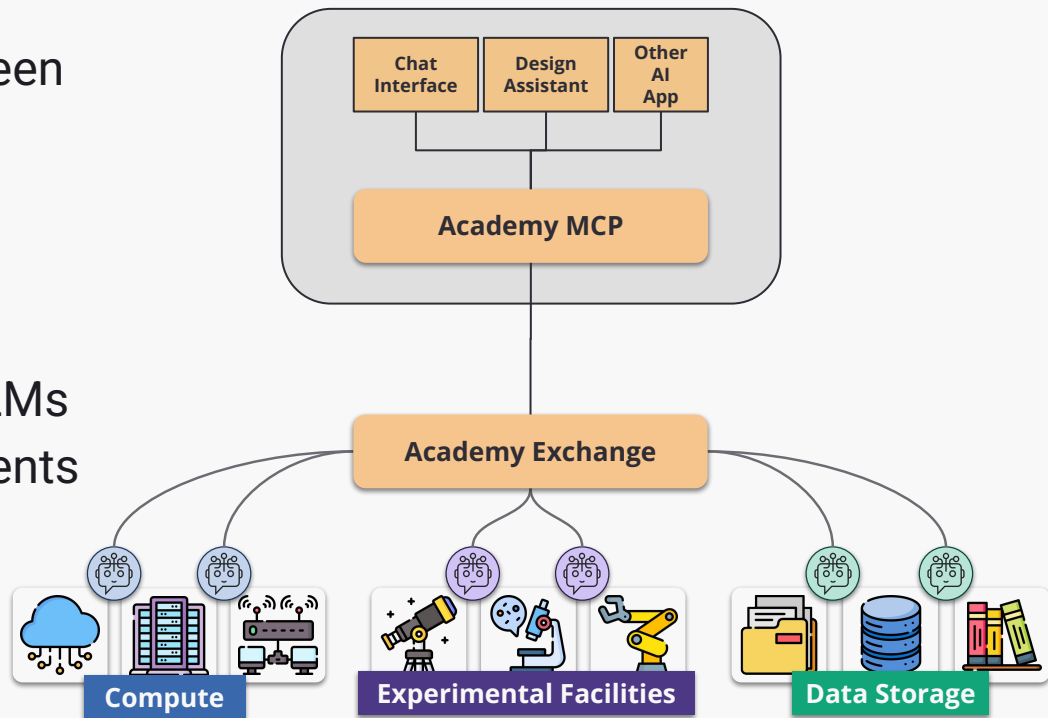
...
agent_registrations.append(
    await manager.register_agent(WordleAgent),
)

```

Mailboxes are minted before launch to create circular dependencies

Other features: 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



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()`



Break

10:00 – 10:30 am



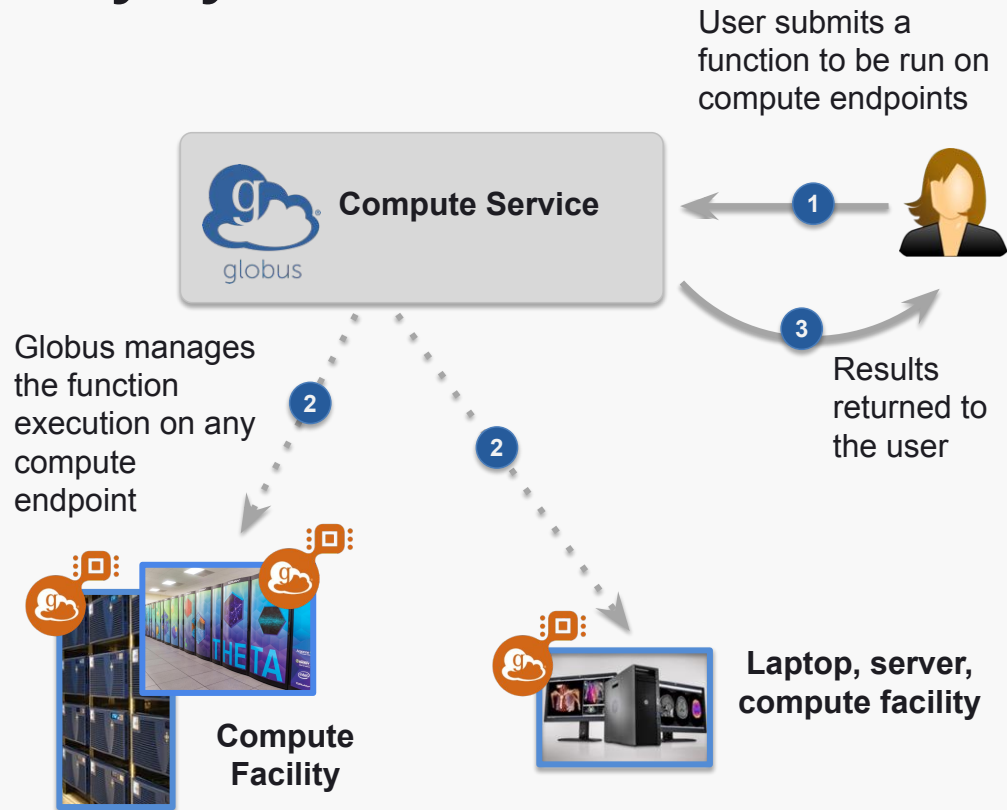
Hands-on Remote Agents

10:30 – 11:00 am



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/>

Launching agents

Goal: Launch agents on remote computers

- Academy supports various methods of deploying agents
 - Manual: `Runtime.run_until_complete()`
 - Thread/Processes: local deployment
 - Parsl: scalable deployment on HPC systems
 - Globus Compute: distributed and scalable deployment on arbitrary systems

Distributing Agents

Goal: Run and Communicate with Agents 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>')
```

Questions?

Up next → Hands-on: Battleship



Hands-on: Battleship

11:00 – 11:30 am

Example: Battleship (30 Minutes)

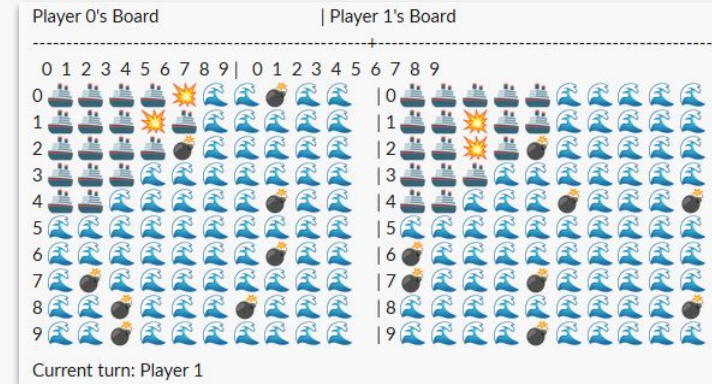
Battleship Rules:

- 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 (30 Minutes)

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:

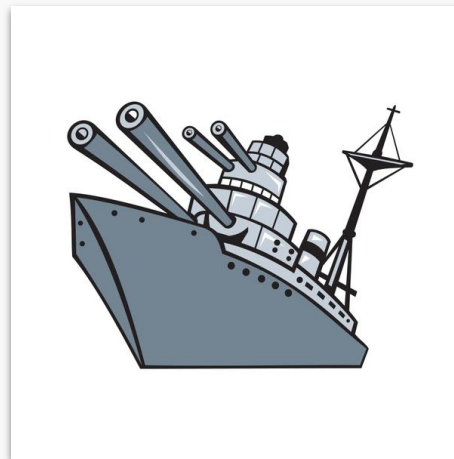
- Guess in Clusters or in Rows based on hits/misses
- Keep track of opponents previous guesses to place ships next time
- Query LLM for next move?
 - How good is ChatGPT at War Games...



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

```
source ipdps.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>)
```



Demo: What's Next

11:30 – 11:45 am

Dashboard

How do you monitor and interact with remote agents?

- Monitoring and observability through `academy-dashboard`

```
$ pip install academy-dashboard
```

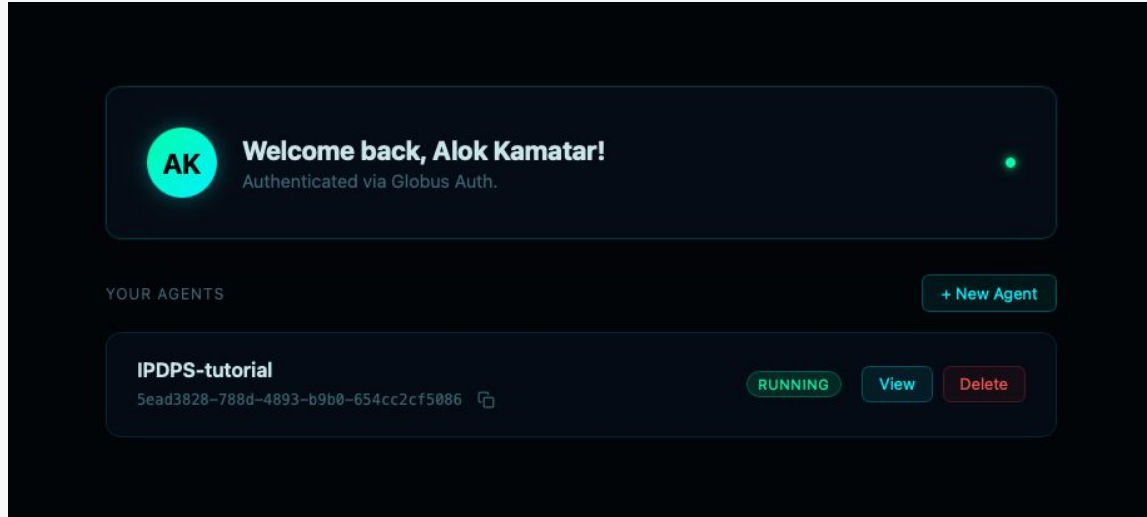
- Locally hosted or service based solution

<https://dashboard.academy-agents.org>

Upcoming Features: Javascript agents, More control features

Demo: Dashboard

Linked to the
hosted
exchange or
local exchange



Observability structured as `Agents`

Monitored Agent

MonitoredAgent
sends logs back
to UserAgent

Can query
UserAgent
before
performing a
task

```
$ pip install academy-dashboard
```

```
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(
            ...
        )
```



Wrap up

11:45 – 12:00 pm

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

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



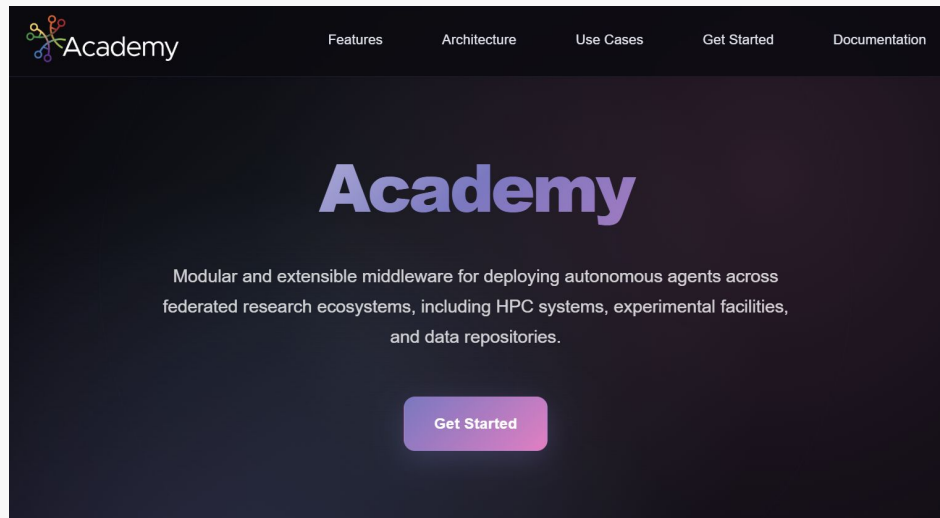
Read the paper



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



Join the open community meeting
Wednesday 1pm CT



Questions?



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

