ACADEMY

Framework Reference Card

Federated Actors and Agents — Build and Deploy Stateful Agents across Federated Resources • docs.academy-agents.org

Academy is a modular and extensible middleware for building, deploying, and managing stateful actors and autonomous agents across federated research computing infrastructure. In Academy, you can: express agent behavior and state in code; manage inter-agent coordination and communication; and deploy agents across distributed, federated, and heterogeneous resources.

1 · Why Academy?

Academy offers a powerful and flexible framework for building and scaling distributed agent-based systems, particularly well-suited for the complexities of scientific applications. The following features make Academy valuable:

Agent Autonomy: Academy allows agents to have autonomous control loops, integrating arbitrary local or remote LLMs, empowering agents to make decisions, react to events, and execute tasks independently.

Flexible Deployment: Academy provides tools for managing agent deployment, communication, and coordination in complex environments such that applications can leverage heterogeneous, distributed, and federated resources.

Stateful Agents: Academy enables agents to maintain state, crucial for managing long-running processes, tracking context across steps, and implementing agents that need to "remember" information.

Foundation for Sophisticated Applications: Academy primitives offer a strong foundation for building highly specialized and sophisticated agent-based systems that go beyond standard LLM use cases, allowing for fine-grained control and optimization tailored to specific scientific applications.

2 · What can be an agent?

An Academy agent is simply an entity that (1) has internal state, (2) performs actions, and (3) communicates with tools, humans, or other agents. This framing allows for a range of agent implementations — Academy agents are building blocks for constructing more complex agent-based systems. For example, Academy can be used to create:

Agent Type	Description
Stateful Actors	Manage internal state and respond to requests in a distributed system
LLM Agents	Integrate LLM-based reasoning and tool calling
Embodied Agents	Control a robot or simulated entity, with actions translated into motor commands or environment manipulations
Computational Units	Encapsulate a specific computational task, like running a simulation, processing data, or training a machine learning model
Orchestrators	Manage or coordinate the activities of other agents, distributing tasks and monitoring progress
Monitors	Monitor and observe the decisions and actions of other agents
Data Interfaces	Interact with external data sources, such as databases, file systems, or sensors, providing a consistent interface for data access and manipulation

3 · Installation

Academy is available on PyPI:

```
pip install academy-py
```

4 · Example

Agents in Academy are defined by an Python Agent class containing `@action`-decorated methods that can be invoked by users or peer agents and `@loop`-decorated methods that execute the autonomous control loops (e.g., reasoning) of the agent.

This sensor monitoring agent periodically reads a sensor in the `monitor()` loop and processes the reading if a threshold is met. Users or agents can invoke the `get_last_reading()` and `set_process_threshold()` actions remotely to interact with the monitor agent:

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

class SensorMonitorAgent(Agent):
    def __init__(self) -> None:
        super().__init__()
        self.readings: list[float] = []
        self.last_reading: float | None = None
        self.process_threshold: float = 1.0
        self.llm = ChatOpenAI(model='gpt-4o-mini')

    @action
    async def get_last_reading(self) -> float | None:
        return self.last_reading

    @action
    async def set_process_threshold(self, value: float) -> None:
        self.process_threshold = value

    @loop
    async def monitor(self, shutdown: asyncio.Event) -> None:
        while not shutdown.is_set():
            value = await read_sensor_data()
            self.last_reading = value
            self.readings.append(value)
            if value >= self.process_threshold:
                reply = await self.llm.ainvoke([{
                    'role': 'user',
                    'content': f'History: {self.readings[-10:]}. Is the last reading anomalous? Reply YES or
NO.',
                }])
                if 'YES' in reply.content:
                    await process_reading(value)

            await asyncio.sleep(1)
```

Users and agents communicate asynchronously through handles, sending messages to and receiving messages from a mailbox managed by an *exchange*. Users can deploy their own exchanges or use a globally-accessible cloud hosted exchange hosted by the Academy team. The *manager* abstracts the remote execution and management of agents using executors.

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

async with await Manager.from_exchange_factory(
    factory=LocalExchangeFactory(),  # Replace with other implementations
    executors=ThreadPoolExecutor(),  # for distributed deployments
) as manager:
    agent_handle = await manager.launch(SensorMonitorAgent)

    await agent_handle.set_process_threshold(2.0)
    await asyncio.sleep(5)
    value = await agent_handle.get_last_reading()
    print(value)

    await manager.shutdown(agent_handle, blocking=True)
```

Learn more: the Getting Started guide at docs.academy-agents.org covers concepts, tutorials, HPC agents, LLM agents,

and persistent agents in depth.

5 · Using Academy with MAG

LLM-backed Academy agents (e.g., agents built with OpenAI-compatible clients, LangChain, or other agentic frameworks) can use the Model Access Gateway (MAG) by setting the standard OpenAI-compatible environment variables to your MAG project key and endpoint:

Step 1 – Get an American Science Cloud (AmSC) Account

Follow the instructions listed here: https://amsc-docs-d762d2.gitlab.io/GM-getting-started#access--login to create an AmSC account. Once the account is approved, proceed to the next step.

Step 2 — Request MAG access and generate a Personal Access Token (PAT)

Follow the instructions listed here: https://amsc-docs-d762d2.gitlab.io/model-access-gateway#quick-start

```
export OPENAI_API_KEY=<your-MAG-project-key>
export OPENAI_BASE_URL= https://i2-api.genesis.american-science-cloud.org/
```

Note: MAG currently requires an AmSC Google account, and the endpoint URL above is expected to change. For key creation, model names, and verification steps, see the MAG Reference Card.

6 · Examples and Resources

Runnable examples and reference material live in the academy-agents documentation and GitHub organization.

Resource	Location
Source code	github.com/academy-agents/academy
Documentation	docs.academy-agents.org — Concepts, Case Studies, Guides, API Reference
Federated deployment examples	github.com/academy-agents/agentic_blueprint_catalog — federated/
HPC hierarchical tool calling	github.com/academy-agents/agentic_blueprint_catalog — hpc_hierarchical/
Reusable agents (PI_Calculator, Director)	github.com/academy-agents/agentic_blueprint_catalog — agents/
LLM model utilities	github.com/academy-agents/agentic_blueprint_catalog — model/
Observability dashboard	github.com/academy-agents/dashboard

7 · Citation

The Academy paper is available on arXiv at arxiv.org/abs/2505.05428.

```
@inproceedings{academy,
    author={Kamatar, A. and Pauloski, J.G. and Babuji, Y. and Chard, R. and Sakarvadia, M.
and Babnigg, D. and Foster, I. and Chard, K.},
    booktitle={IEEE International Parallel and Distributed Processing Symposium},
    title={Empowering Scientific Workflows with Federated Agents},
    year={2026},
    pages={1403-1418},
    doi={10.1109/IPDPS65963.2026.00114}
}
```