Agentic Tools Part 1: OpenClaw, Hermes, Trinity

From the agent landscape to Trinity — ATPESC 2026

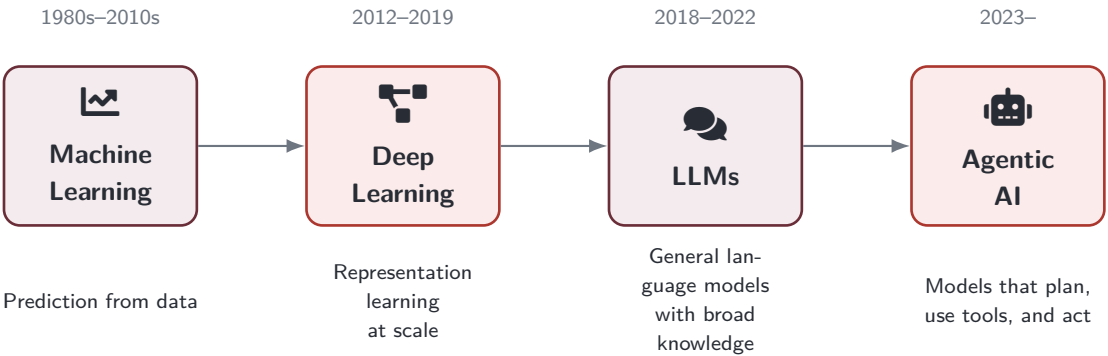
Huihuo Zheng, Computer Scientist
Argonne Leadership Computing Facility (ALCF), Argonne National Laboratory
August 4, 2026

huihuo.zheng@anl.gov

Outline

1. **History of AI and Agentic AI**
2. **What is an agent and an agentic system**
3. **Agentic platforms: OpenClaw, Hermes, Omnigent**
4. **Agentic system for HPC: Trinity**

History of AI and Agentic AI



*The shift is from **predicting** to **generating** to **acting**.*

LLMs, agents, and agentic systems are not the same thing

LLM

- Primarily **responds** to a prompt
- Generates text, code, or analysis
- Usually stops after one answer
- Does not manage execution by itself

Examples: GPT-5, Claude

Agent

- Wraps an LLM with **tools** + **memory** + **control logic**
- Can plan, act, observe, and continue
- Executes a task instead of only answering
- Can use tools and recover from failures

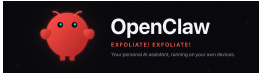
Examples: GPT-5 → Codex, Claude → Claude Code

Agentic system: the operational stack around one or more agents: tools, memory, permissions, policies, scheduling, user interaction, and recovery.

Examples of agentic platforms and systems

Codex

Claude Code



OpenClaw

Hermes

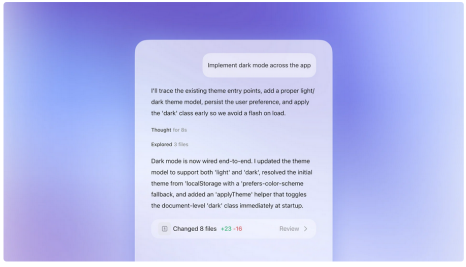
Omnigent

Trinity

These systems span persistent personal agents, coding agents, and shared agentic platforms.

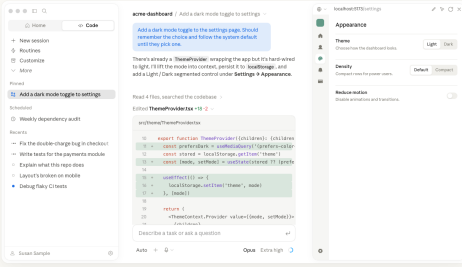
Codex and Claude Code: cloud coding agents that open a diff for review

Codex (OpenAI)



Launched from ChatGPT — traces the task, edits files, and surfaces a reviewable diff before it ships.

Claude Code (Anthropic)

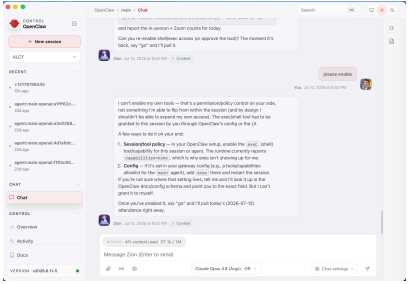


Same agent from terminal, IDE, Slack, or web — reads real files and renders a live diff alongside the running app.

Chat-driven, codebase-grounded, reviewed before merge — the direct precursor to persistent multi-tenant platforms like Trinity.

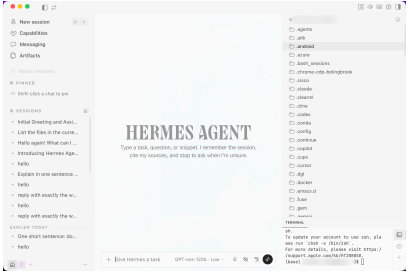
OpenClaw and Hermes: persistent personal agents that live on your machine

OpenClaw



Permission-gated: the agent can't grant itself new tools — the operator must approve exec/shell access. Runs on your own machine, not a vendor cloud.

Hermes Agent



This deck's own hands-on setup, live — Hermes running `GPT-oss-120b` through the ALCF Sophia endpoint.

Memory and routines that survive across sessions — one operator, always on, the mechanism stack the next few slides break down.

An agent harness is the runnable program that turns a model into an agent

Agent harness: the concrete software that wraps an LLM's API calls in a loop — plan, call a tool, observe the result, decide whether to continue — plus the memory, permissions, and UI around it.

Cloud coding harness

Codex, Claude Code — one task, one repo, one diff for review.

Persistent personal harness

OpenClaw HERMES-AGENT

OpenClaw, Hermes — one operator; memory and routines persist across sessions.

Multi-tenant harness

Trinity — many scientists and trust boundaries on one shared facility.

Same loop underneath — the harness is what changes as the trust boundary grows.

HERMES-AGENT

Hands-on: Setting Up Hermes Agent

Hands-on: connect Hermes to your own Claude Code subscription

Hermes can route through **Anthropic OAuth** — the same credential Claude Code already uses — instead of a separate API key.

Interactive setup

```
hermes model
# -> select "Anthropic OAuth" (routes as Claude Code)
```

config.yaml (manual)

```
# ~/.hermes/config.yaml
model:
  provider: "anthropic"
  default: "claude-sonnet-4-6"
```

⚠ Requires a **Claude Max** plan with extra usage credits purchased — your base Max allowance is not consumed, and **Claude Pro accounts cannot use this path**.

Hands-on: connect Hermes to the ALCF inference endpoint (gpt-oss)

>_ 1. Get a Globus access token

```
wget https://raw.githubusercontent.com
     /argonne-lcf/inference-endpoints/
     refs/heads/main/
     inference_auth_token.py
python inference_auth_token.py
     authenticate
echo "ALCF_INFERENCE_TOKEN=$(python
     inference_auth_token.py
     get_access_token)" >> ~/.hermes/.
     env
```

2. config.yaml

```
# ~/.hermes/config.yaml
providers:
  sophia:
    name: ALCF Sophia (gpt-oss)
    base_url: "https://inference-api.alcf.anl.
         gov/resource_server/sophia/vllm/v1"
    key_env: ALCF_INFERENCE_TOKEN
    transport: openai_chat

# Optional: make it the default
model:
  default: openai/gpt-oss-120b
  provider: sophia
```

⚠ Without the `model:` block, switch any time with `hermes model` or `hermes --provider sophia -m openai/gpt-oss-120b`. Token valid ∼**48h**; Sophia cold-starts in **10–15 min** if idle.

Known issue: tool calls on Sophia fail with HTTP 422 — one-line fix

Hermes attaches a legacy `name` field to every tool-result message; ALCF's vLLM enforces a strict schema and rejects it (`extra_forbidden: name`). Confirmed on **Hermes v0.20.0** (commit `a991dfc25`).

`~/.hermes/hermes-agent/agent/agent_runtime_helpers.py`

Add this line right before the final `return messages` in `sanitize_api_messages()`:

```
messages = [{k: v for k, v in m.items() if k not in ("name", "tool_name")} if isinstance(m,
      dict) and m.get("role") == "tool" else m for m in messages]
```

Provider-agnostic and safe for every backend — strips a field no provider actually needs on a tool message. Not yet upstreamed; re-apply after `hermes update`.

Hands-on: give Hermes a real two-turn coding task

You — turn 1

"Please create a PyTorch MNIST training script."

You — turn 2 (follow-up)

"Please run that code."

*Watch for: does it write a file, then actually **execute** it — not just describe what it would do? Does it install missing packages (`torch`, `torchvision`) on its own? Does training loss actually drop over epochs?*

A genuine **plan** → **act** → **observe** loop across two turns — code generation (file write) then code execution (terminal tool). First run downloads the MNIST dataset (∼10 MB) — needs outbound internet from wherever the terminal tool runs.

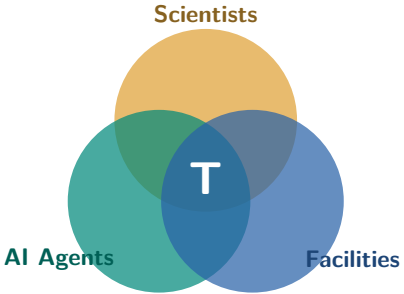
Hands-on: push it onto a real DOE system — discuss what you find

You — turn 3 (follow-up)

"Could you create a submission script on Polaris @ ALCF for this?"

Discuss with your neighbor: did it get the queue, account/allocation flag, node & rank count, module loads, and the `#!/bin/bash -l` shebang right — or did it **hallucinate** ALCF-specific details it was never grounded in?

Keep this answer in mind — it is exactly the gap the rest of this talk is about closing.



Argonne NATIONAL LABORATORY

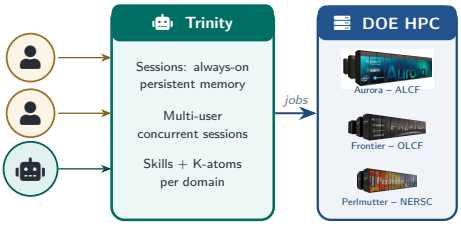
Introducing Trinity

An agentic platform for orchestrating HPC science

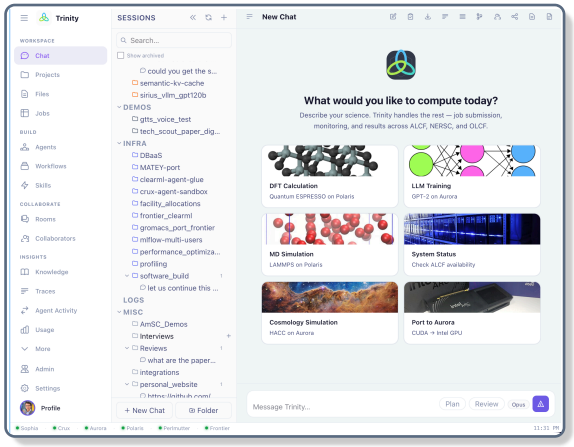
Watch the Trinity introduction video

huihuo.zheng@anl.gov

Serving HPC computation across DOE facilities via one Trinity platform

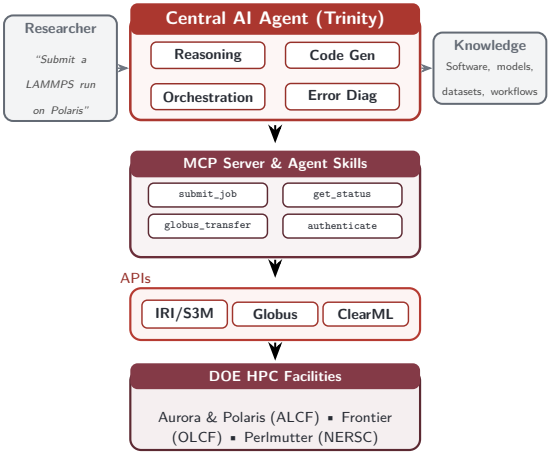


Serving HPC via AI agents — always-on, multi-user.



Trinity today: one chat interface, direct access to ALCF, NERSC, and OLCF.

A central AI agent turns requests into MCP calls across three DOE facilities



Domains (600+ jobs)

Materials (DFT, QMC)	Biology (protein)	Imaging (ptycho/tomo)
Fusion (plasma)	AI Training (AuroraGPT)	Cosmology (HACC)
Software Porting	Perf. Optim.	Perf. Modeling

600+ autonomous campaigns across 13 DOE systems, 3 facilities — domain agnostic.

A plain-language request becomes a monitored PBS job

Scientist

"Submit a 4-node LAMMPS run on Polaris under my allocation, then let me know when it is done."

Trinity

Resolved account & run directory via `run_dir.py`, generated the job script below, and submitted it through the `submit-job` skill.

`>_ run.sh (bash)`

```
#!/bin/bash -l
#PBS -A <allocation> -q prod -l select=4:system=polaris -l walltime=01:00:00
WORKDIR="$(cd "$(dirname "$0")/.." && pwd)"; cd "$WORKDIR"
mpiexec -n 16 --ppn 4 lmp -in in.lammps
```

Trinity: Job 1234567 completed in 42 min on 4 Polaris nodes; outputs saved under `jobs/polaris/lammps_run/outputs/`.

Trinity does two things: orchestrates HPC jobs, and hosts human-agent collaboration

Orchestration

An **agentic system for orchestrating HPC simulation** — natural-language job submission, monitoring, and result synthesis across ALCF, NERSC, and OLCF facilities.

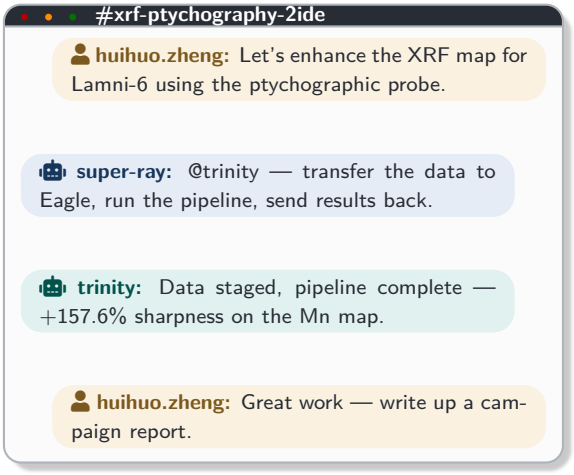
Collaboration

A **platform for scientists and agents to collaborate** — shared chat rooms where humans and AI agents work side by side, backed by a shared skill and knowledge catalog.

"Multiple simultaneous users, strict tenant isolation, — sharing reusable skills and an HPC & domain-science knowledgebase."

Shared rooms let humans and agents work in one conversation

- **Shared rooms** where humans and AI agents co-exist as first-class participants
- **Every participant sees updates instantly** — no refresh, no polling
- Agents run under **the owner's own tokens**, with a **budget gate the owner sets**



Takeaways for ATPESC

1. **Agentic systems are a spectrum.** Chat assistants, task agents, persistent personal agents, and multi-tenant platforms solve different problems.
2. **Persistence is the first big systems step.** Memory, routines, steering, and recovery are what make agents operational instead of theatrical.
3. **Trinity is the platform step for HPC.** It takes persistent-agent mechanisms and re-hardens them for shared DOE facilities, shared knowledge, and human-agent scientific collaboration.

If the first wave of agents was about "can a model use tools?", the next wave is about "can many scientists trust and steer those agents on real systems?"

Would like to test Trinity? Please email Trinity <trinity.agent.zion@gmail.com>, and cc me <huihuo.zheng@anl.gov>.

ARGONNE
ATPESC2026
EXTREME-SCALE COMPUTING

ARGONNE TRAINING PROGRAM ON EXTREME-SCALE COMPUTING

Produced by Argonne National Laboratory, a U.S. Department of Energy Laboratory managed by UChicago Argonne, LLC under contract DE-AC02-06CH11357.

Special thanks to the National Energy Research Scientific Computing Center (NERSC) and Oak Ridge Leadership Computing Facility (OLCF) for the use of their resources during the training event.