

Building the HEP Agent Stack

MCP Servers, CLI Tools, and Skill-Based Workflows for Scientific Analysis

Mohamed Elashri, Mike Sokoloff, Conor Henderson

University of Cincinnati • LHCb • IRIS-HEP



Three projects: `root-mcp`, `cerngitlab-mcp`, and `inspirehep-mcp`.

Focus of this talk: what these tools enable for HEP, what changed recently, and why I am increasingly interested in pairing MCP with lighter SKILL + CLI interfaces.

Key concepts

Terms used throughout this talk

MCP

Model Context Protocol

A standard JSON-RPC protocol that lets LLMs call tools, access resources, and retrieve structured data from external servers.

CLI

Command-Line Interface

A shell-based tool that exposes the same backend as the MCP server. Useful for scripting, debugging, and direct terminal use.

SKILL / SKILL.md

A small markdown file that teaches an LLM agent which CLI commands exist and how to compose them — no server process needed.

Shared backend pattern: one library → MCP server + CLI tool + SKILL file. All three projects in this talk follow this design.

Talk map

This talk covers

1. Why agentic AI matters in HEP workflows
2. What MCP solved for early LLM tooling
3. Why I now care about SKILL + CLI as a complement
4. What each project contributes
5. Deployment patterns relevant to IRIS-HEP
6. Open problems and next steps

Audience promise

This is not a hype talk. The goal is practical infrastructure for:

- ▶ reading HEP data,
- ▶ searching experiment software,
- ▶ navigating literature,
- ▶ and reducing analysis boilerplate.

Why agentic AI for HEP?

The bottleneck is usually access, not text generation

Traditional LLM loop

1. Describe problem, receive script suggestion
2. Copy, edit, run, debug manually
3. Repeat until it works

HEP makes this harder

ROOT files, experiment software, private Git-Lab instances, and paper metadata all sit outside the model's context.

Agentic loop

- ▶ **Perceive:** access data and code via tools
- ▶ **Reason:** plan multi-step actions
- ▶ **Act:** run tool calls or shell commands
- ▶ **Report:** return structured results

Key idea

A useful scientific assistant is a thin orchestration layer over domain tools that already exist.

MCP: why it mattered

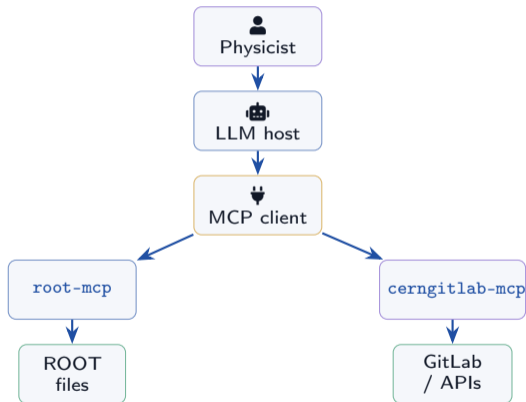
A standard way to let models use tools

What MCP gives us

- ▶ standard JSON-RPC tool interface
- ▶ interoperability across clients
- ▶ structured inputs and outputs
- ▶ local or remote server deployment

Why that helped in HEP

It gave a clean way to wrap ROOT, GitLab, or literature APIs as tools that an LLM can call without inventing a custom integration each time.



MCP and SKILL + CLI: two complementary paths

Why I started pairing a lighter interface alongside MCP

Observation from real usage

For many analysis tasks, full MCP traffic is verbose:

- ▶ JSON overhead per tool call
- ▶ Structured payloads for all results
- ▶ Full tool catalog stays in context

Alternative pattern

Use a small `SKILL.md` to teach commands, then drive a normal CLI. Outputs are shorter and easier to compose.

Practical split

MCP: strict schemas, tool autonomy, remote services.

SKILL + CLI: lighter context, simpler debugging, shell composition, lower token overhead.

My current view

Layered design: *shared backend + two front ends + one skill description.*

The project portfolio

Three servers, one general theme: domain-specific agency for HEP

root-mcp

ROOT-aware layer for file inspection, branch reading, histograms, selections, and plots.

- ▶ MCP server
- ▶ `root-cli`
- ▶ local analysis focus

cerngitlab-mcp

GitLab-aware layer for project search, file retrieval, code search, and stack-aware discovery.

- ▶ 14 MCP tools
- ▶ `cerngitlab-cli`
- ▶ stdio and HTTP modes

inspirehep-mcp

Literature-aware layer for papers, citations, references, and collaboration searches.

- ▶ paper discovery
- ▶ citation graph access
- ▶ reference generation

Shared design philosophy

Expose existing HEP infrastructure through interfaces that agents can actually use. All projects include **comprehensive documentation** and **extension guides** for adding new tools.

Deep dive 1: root-mcp

Turning ROOT interaction into an agentic workflow

What it covers

- ▶ Inspect ROOT structure, trees, branches
- ▶ Read data with selections, compute histograms
- ▶ Plot 1D/2D results, simple statistics
- ▶ MCP or CLI on same backend

Why it matters: Model becomes thin assistant for first-pass analysis.

Update: Presented to the **ROOT team**; ongoing collaboration for deeper C++ integration.

Prompt: *"Plot J/ψ mass from B2JpsiK.root"*

```
Reading B2JpsiK.root (Tree: '
    DecayTree')
Variable: 'jpsi_M'
Entries: 12,402

mean = 3096.2 MeV, rms = 12.4 MeV
Fit (Crystal Ball):
    peak = 3095.8 +/- 0.2 MeV
    sigma = 11.2 +/- 0.3 MeV
Saved: jpsi_mass_fit.png
```

Deep dive 2: cerngitlab-mcp

Software archaeology and collaboration knowledge as tools

High-value use cases

- ▶ Locate analysis examples across repos
- ▶ Inspect CI/build config, search code
- ▶ Discover software in experiment stacks
- ▶ Expose public repos without mandatory auth

What's new

- ▶ Direct CLI + SKILL.md support
- ▶ Dual stdio/HTTP modes, multi-user deployment

Prompt: *"Search for 'RooFit' usage in lhcb/-Moore"*

```
Found 3 matches in lhcb/Moore:  
Hlt/Hlt2Conf/python/Hlt2Conf/  
  Selections.py:  
    import RooFit  
Hlt/Hlt2Conf/src/VertexFitter.cpp:  
  // Uses RooFit for mass constraint
```

```
Project: lhcb/Moore (ID: 1234)  
Star count: 42 | Last activity:  
  2026-04-20  
Stack: sim11 | Ref: master
```

Deep dive 3: inspirehep-mcp

Literature can also be an agentic surface

Capabilities

- ▶ Search papers by topic/author/collaboration
- ▶ Get metadata from Inspire/DOI/arXiv IDs
- ▶ Traverse citations, generate BibTeX

Why it fits: Analysts navigate literature too—agents should span data, code, *and* papers.

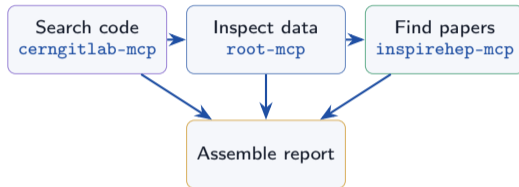
Prompt: *"Find 2026 LHCb papers on vertexing with BibTeX"*

```
Found: "SemiCharmTag: a tool for..."  
  Authors: Arata, C. et al. (2026)  
  arXiv: 2604.11574 [hep-ex]
```

```
@article{Arata:2026suo,  
  author = "Arata, Carolina and ...",  
  title = "{SemiCharmTag: a tool  
  ...}",  
  eprint = "2604.11574",  
  year = "2026" }
```

One end-to-end scientific workflow

The interesting part is orchestration across tools



Code: Find relevant implementation or analysis pattern.

Data: Inspect ROOT sample, quick plot, or statistic.

Literature: Pull references and citation graph.

Takeaway: Composable HEP agent stack where code, data, and literature are first-class tool surfaces.

MCP vs SKILL + CLI: detailed comparison

Two front-ends, one shared backend — choose by task

MCP (Model Context Protocol)

- ✓ Strict schemas, validated inputs and outputs
- ✓ Interoperable across LLM clients and hosts
- ✓ Native support for remote and multi-user services
- ✓ Well-defined tool discovery protocol
- ✗ Verbose JSON overhead per call
- ✗ Full tool catalog loaded into model context
- ✗ More layers to configure and debug

SKILL + CLI

- ✓ Tiny context footprint (one markdown file)
- ✓ Direct shell output, easy to inspect
- ✓ Natural shell composition and piping
- ✓ Lower token overhead for routine tasks
- ✗ No strict schema enforcement
- ✗ Coupled to a specific CLI tool
- ✗ Less interoperable across clients

Design direction: one shared backend → MCP server + CLI + SKILL file. Use MCP for autonomous agents and remote services; use SKILL + CLI for analyst workflows and lower overhead.

Deployment patterns for IRIS-HEP

From personal workflows to shared experiment services

Pattern A: local researcher setup

- ▶ laptop or workstation
- ▶ local MCP client or CLI-enabled agent
- ▶ direct access to local files and repos
- ▶ lowest ops burden

Good for

prototyping, analyst productivity, single-user workflows, teaching, and debugging the tool surface.

Pattern B: shared service

- ▶ HTTP-backed MCP for many users
- ▶ central auth and audit layer
- ▶ experiment-aware backends
- ▶ can sit next to internal storage or APIs

IRIS-HEP opportunity

Standardize agent infrastructure. **Success case:** CERN ML Service now hosts these projects and covers API costs for experiment users.

What I think IRIS-HEP can help standardize

Infrastructure questions are now as important as model questions

1. Interfaces

When should a project expose MCP, CLI, or both? What belongs in a skill file versus in strict schemas?

2. Evaluation

How do we measure success: analyst time saved, onboarding speed, correctness, reproducibility, or adoption?

3. Shared backends

Which HEP tool integrations should be built once and reused across experiments: ROOT access, Git hosting, literature search, metadata catalogs, and plotting?

4. Security and governance

Auth, quotas, audit logging, and data locality will decide whether these systems become real collaboration tools.

5. Reproducibility

Can agent actions leave an explicit trail: commands executed, files read, plots produced, references fetched, and prompts used?

Current lessons learned

What has held up well, and what still feels open

What seems robust

- ▶ Domain-specific tools > generic chat
- ▶ Shared backends with multiple interfaces scale
- ▶ Literature/code/data should be connected
- ▶ Users prefer direct CLI workflows

What is still open

- ▶ Compact yet structured output patterns
- ▶ Autonomy vs. safety balance
- ▶ Evaluation beyond anecdotal demos
- ▶ Packaging/deployment at collaboration scale

Working conclusion: Winning abstraction = stack: APIs/libraries underneath, CLIs/MCP on top, skill files as lightweight instruction layer.

Summary: Building the HEP Agent Stack

The Strategy

- ▶ **Unified Interface:** Data, Code, and Papers as tools.
- ▶ **Layered Access:** MCP servers + CLI + Skill files.
- ▶ **Extensible:** Built to be expanded by the community.

The Ecosystem

- ▶ **ROOT Team:** Collaboration for C++ data access.
- ▶ **CERN ML:** Official hosting and API support.
- ▶ **IRIS-HEP:** Path to shared infrastructure.

Get Started

`root-mcp`

`cerngitlab-mcp`

`inspirehep-mcp`

 **Docs & Guides**

Discussion: Which HEP tool surface should we standardize next?

Thank you

Questions and discussion

Mohamed Elashri

mohamed.elashri@cern.ch

