ÖAW Austrian Academy of Sciences

MARIETTA BLAU INSTITUTE FOR PARTICLE PHYSICS

Agentic tools in CMS@MBI

Applications, results & lessons learned

Dorian Sloot

On behalf of the CMS Group at MBI

Content

Agentic Workflow

How we integrate AI agents into day-to-day research workflows, from problem formulation to implementation and validation.

- Human-in-the-loop research
- Planning before execution
- Tool-connected agents
- Iterative validation & refinement

Case studies

Examples from our group showing how agents can accelerate analysis, software development, and scientific problem solving.

- MC debugging
- Inclusive jets / analysis automation
- Efficiency parametrization
- Software / FPGA design-space exploration

Lessons & Conclusions

What works well, where agents still struggle, and how to use them effectively in scientific research.

- Strongest for coding & technical execution
- More iterations in less time
- Tool access and workflow design matter
- Scientific judgement remains essential

From assistant to agentic workflow

Human sets the scientific objective and defines what constitutes a valid result.

01

DEFINE

Physics goals, assumptions and constraints

02

PLAN

Explore dependencies and choose approach

03

EXECUTE

Implement, run tools and inspect outputs

04

VALIDATE

Tests, physics checks and iteration

Interpret → Refine → Iterate

Scientific judgement and final decisions remain with the researcher.

AI agents across the research workflow

Software development

CMSSW C++ exploration, merge requests, analysis frameworks, code explanations

Debugging & validation

Monte Carlo issues, physics consistency, closure tests, code reviews

Analysis & optimization

Jet energy resolution, trigger turn-on curves, efficiency parametrization, FPGA implementations

Research support

Literature retrieval, CRAB/Slurm jobs, documentation, notes and slides

MC debugging: from symptoms to root cause

~2 months

Manual search for a gridpack problem producing an unexpected φ modulation in MC.

~1 hour

Claude identified the root cause; follow-up tests and reports took another two days.

Another example: agent-assisted correction of a simulation bug making long-lived particles decay promptly in Geant4 (Ang).

Takeaway: rapid code exploration and systematic debugging enable agents to identify complex issues in a fraction of the time.

Efficiency parametrization: two months → two days

Problem

~2 months of manual trial-and-error without a satisfactory solution.

Agent exploration

Identified weakly correlated variables; more uniform training sample; 2D → 3D.

Validation

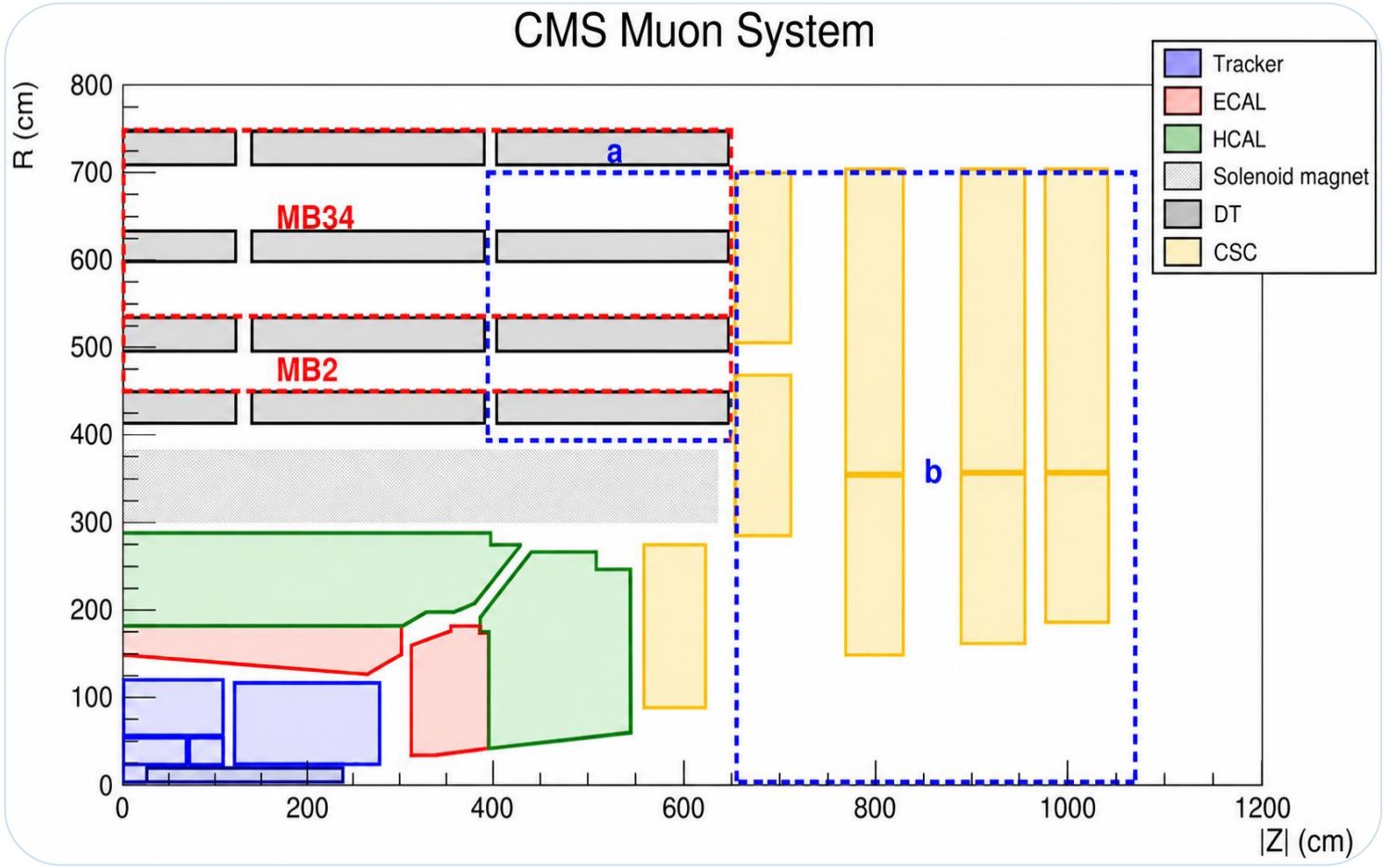
Robustness checks and closure tests built into the workflow.

Outcome

Satisfactory solution in ~2 days of agent-assisted work.

Takeaway: agent-driven exploration can overcome long-standing scientific challenges in a fraction of the time.

Moving from Sonnet to Opus:



Creation

Human & AI collaboration

Model

Claude Opus 5

Output

Substantial upgrade to Sonnet

FPGA exploration: inspect → synthesize → compare

Challenge: many implementation choices across a large hls4ml / Vitis HLS stack.

INSPECT

Trace generated firmware and resource bottlenecks

MODIFY

Explore BitNet-style architectural alternatives

SYNTHESIZE

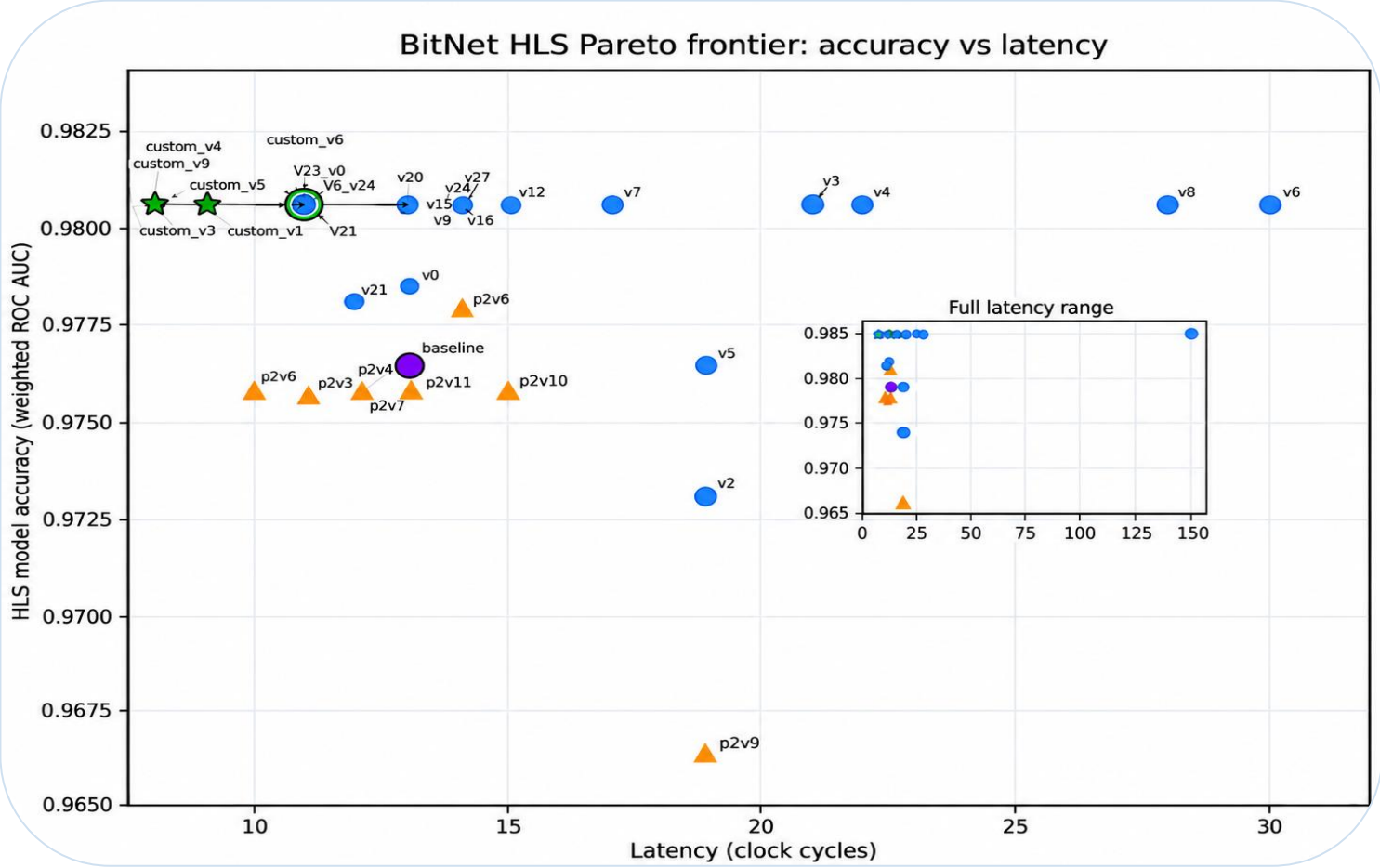
Run hardware synthesis with various approaches

VALIDATE

Latency, LUT, FF, BRAM, DSP and accuracy

Takeaway: agents increases the number of alternatives that can be investigated and evaluated.

FPGA exploration: results



Explore

Investigate multiple variants

Compare

Check synthesis results

Improve

Use human intuition

Inclusive jets: agentic workflow & impact

Agent-supported analysis

Development

Extended the analysis to Run 2 (~10k lines) in ~1 day.

Automation

Trigger turn-ons (×4 bins), including rapidity dependence (×6 bins), JER calibration, CRAB/Slurm jobs, plotting and fits.

Collaboration

Prepared merge requests with human sign-off and helped respond to code review.

Documentation

Well-curated analysis notes, code documentation and traceability to a high standard. [link]

Research throughput

~1 year → 2–3 weeks

Estimated postdoc + PhD effort compressed into weeks.

Scientific impact

~8× more data

Finer calibration increased accessible data; detailed data issues were resolved.

Takeaway: agents can heavily accelerate an entire precision-physics analysis workflow, not just individual coding tasks.

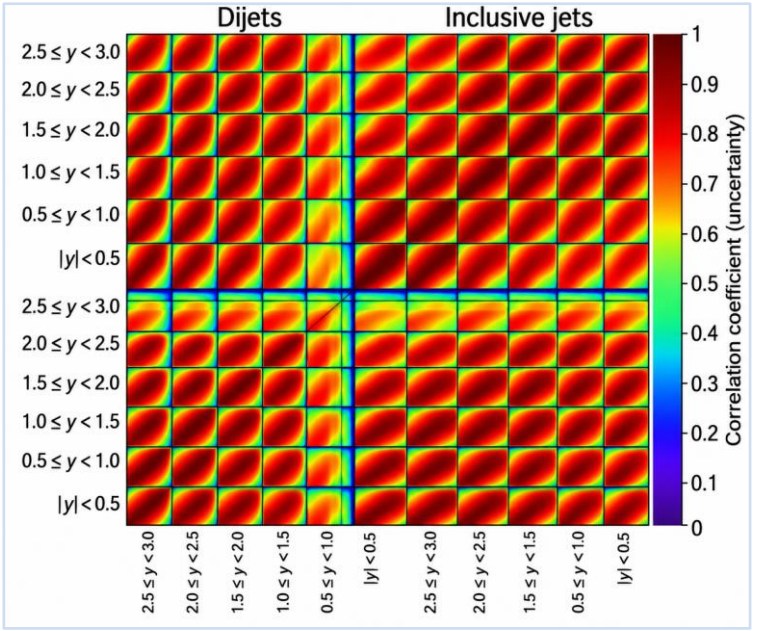
Inclusive jets: scale of the problem

Scientific challenge

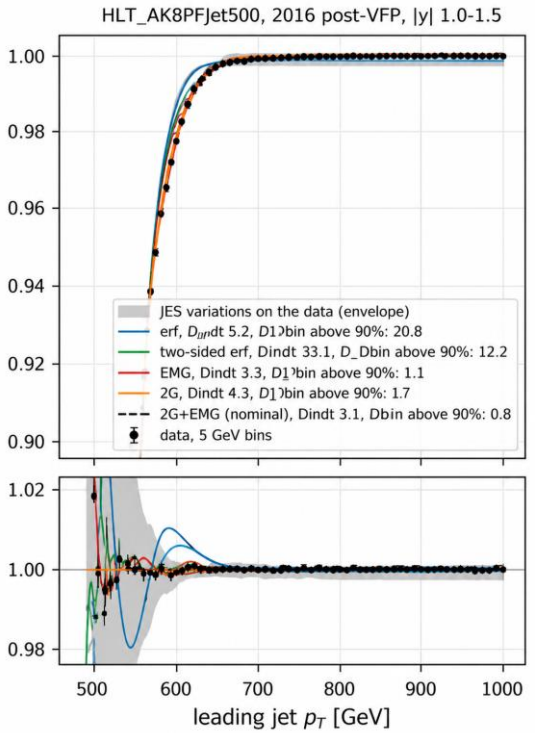
- Legacy inclusive jets+dijets Run 2 measurement.
- O(10^8) events across hundreds of bins.
- 10^{-3} precision target on the gluon PDF.
- Hundreds of trigger turn-ons, scale factors and systematics across ~8 data-taking periods.
- Large-scale assembly and bookkeeping required.

Agentic setup

- Small framework of 2–3 agents on a laptop.
- Claude Opus/Fable, GPT, and tool access to CLIP/CBE, WLCG (CRAB), GitLab and the web.



Example covariance matrix — one of O(100) such matrices in the analysis workflow.



Example trigger turn-on — O(100)

Inclusive jets: the human role

What agents cannot replace

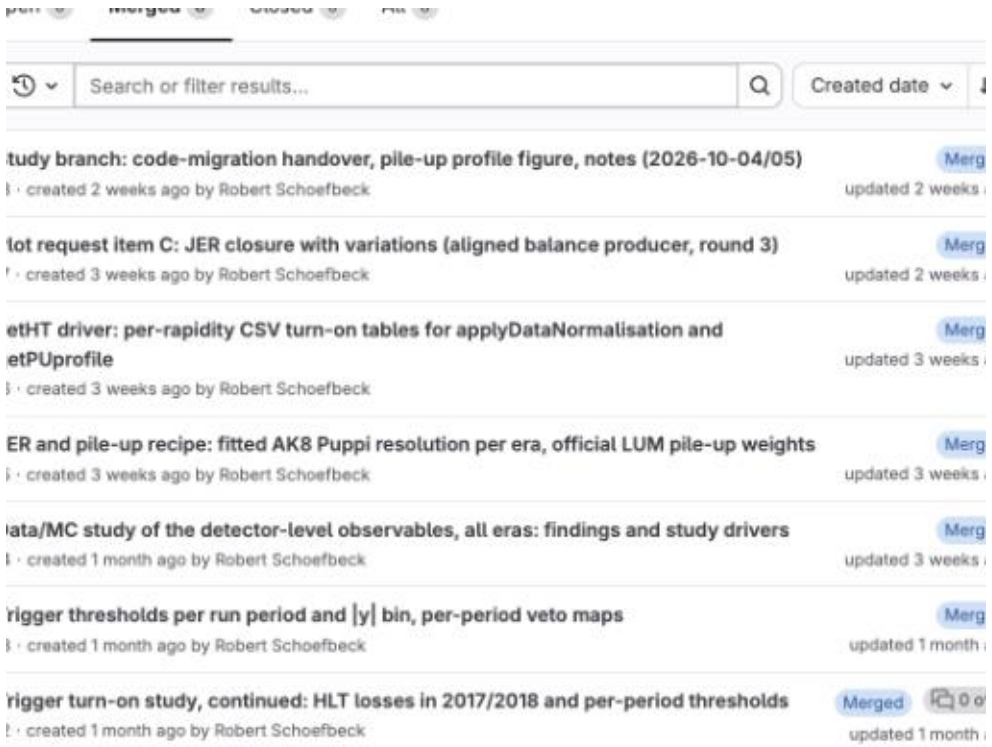
Scientific understanding
Deep detector knowledge, experimental intuition, and understanding (like experimentalists) how recorded events become physics measurements.

Scientific judgement
Evaluating results, identifying shortcomings, making trade-offs, and deciding which improvements to prioritize.

Collaboration & knowledge transfer
Building a team around shared scientific goals and training the next generation of experimental physicists.

Validation & accountability
Independent cross-checks, code reviews, physics interpretation, and final sign-off.

AI-assisted merge requests



Takeaway: interpretation, sign-off and long-term understanding remain human responsibilities.

What works well — and where agents struggle

Strengths

- Execution-heavy work
- Code navigation and explanations
- Debugging and environment setup
- Repetitive analysis and automation
- Fast implementation comparisons
- Bridging knowledge gaps

Limitations

- Overcomplicated designs and scope drift
- Confident but incorrect physics claims
- Overly verbose documentation
- It uses data-science jargon
- Missed follow-ups or irrelevant avenues
- Human review remains necessary

Best practices I: plan, constrain, document

- **Plan before execution**
  Inspect the codebase; specify requirements, boundaries and tests.

- **Make project context persistent**
  Use a concise CLAUDE.md to define project instructions and ensure feature-specific plans are saved with descriptive names directly in the codebase

- **Record decisions**
  Keep rejected alternatives and reasons in version-controlled .md files to avoid revisiting them.

- **Constrain changes and review**
  Avoid unrelated fixes; independently check meaningful modifications.

Practical workflow: explore → plan → implement → test → review → commit

Best practices II: tools, context & guardrails

Connect the right tools

Integrate into workflows

VS Code / CLI, terminal access and MCP connections.

Support long-running work

Use tmux to maintain persistent terminal sessions.

Retrieve and maintain sources

alphaxiv MCP for papers; Evernote MCP for notes.

Manage execution & context

Match model to task

Strong reasoning model for planning; lighter models for routine execution.

Keep context focused

Use /compact or /rewind; save important decisions in files.

Automate follow-ups

/goal for completion criteria; /loop for repeated checks.

Add guardrails

Validate before committing

Run tests and independently review important changes.

Control agent actions

Restrict unrelated edits, destructive commands and access to secrets.

Automate safeguards

Use hooks and code review for repeatable checks.

Conclusions: three takeaways

1 Agents expand research throughput

More ideas, variants and tests become feasible in the same time.

2 Workflow design matters

Planning, persistent context, connected tools and validation are key.

3 Scientific responsibility remains human

Physics meaning, prioritization and acceptance cannot be delegated blindly.

ÖAW Austrian Academy of Sciences

MARIETTA BLAU INSTITUTE FOR PARTICLE PHYSICS

/init & CLAUDE.md

PROJECT SETUP · Give Claude reliable context on your repository

What it does

Initialize project context

/init inspects the repository and generates a starting CLAUDE.md.

Loaded in later sessions

CLAUDE.md supplies reusable project instructions, conventions and common commands.

Recommended content

Build/test commands, code structure, domain assumptions, constraints and known pitfalls.

How to use it

Start in the repository

Run /init, inspect the proposed file and edit it to remove generic or inaccurate information.

Make decisions persistent

Save feature plans and design rationale under meaningful names in the repository.

Share selectively

Version-control team-wide instructions; never store credentials or private data.

CMS example: record CMSSW release, cmsRun test command, expected ROOT output and “do not change physics selections.”

/review & independent checks

CODE QUALITY · Get a second opinion before submitting changes

What it does

Review recent changes

/review can examine a pull request or current work, looking for bugs, risks and missing tests.

Separate perspectives

A fresh-context reviewer can spot assumptions carried forward from a long implementation session.

Version note

/code-review may be a separately installed skill; /review is documented as built in.

Practical review workflow

Inspect the diff first

Check changed files and scope with /diff or git diff.

Request targeted scrutiny

Ask about physics logic, edge cases, reproducibility and test coverage.

Triage findings

Fix confirmed issues, investigate questionable findings, and defer low-impact suggestions.

CMS example: review a trigger turn-on change for bin boundaries, event weights and unexpectedly modified selections.

MCP servers & external tools

TOOL CONNECTIONS · Let agents retrieve external evidence and perform actions

What MCP adds

Beyond the repository

MCP connects Claude to external services such as literature, notes and APIs.

Connection types

HTTP: remote hosted service. stdio: local process launched by the client.

Manage connections

Use /mcp to inspect status or authentication; CLI setup uses claude mcp add.

Practical configuration

Choose the right scope

Project for shared configuration, user for personal defaults, local for project-specific private settings.

Minimize permissions

Connect only trusted services and grant the least access necessary.

Watch context & provenance

Many tools increase complexity; verify retrieved papers and figures against original sources.

CMS example: search papers using an alphaXiv MCP, inspect the actual paper, and link the reference to the implementation.

/compact & /rewind

CONTEXT MANAGEMENT · Recover from long or unproductive sessions

What each command does

/compact

Summarizes accumulated conversation context so work can continue with fewer tokens.

/rewind

Returns to a checkpoint; depending on choice, restore conversation, files or both.

Useful alongside /context

Inspect context use before deciding whether to compact or restart.

When & how to use

Compact after a long investigation

Tell Claude which decisions, known failures and next actions must survive the summary.

Rewind after a wrong turn

Return to the state before unwanted modifications or misleading assumptions.

Persist durable conclusions

Put important design decisions in version-controlled files, not only chat history.

Example: /compact Keep validated JER findings, tested commands, rejected hypotheses and the next closure test.

/goal & /loop

AUTOMATION · Set completion criteria and repeat checks when useful

What each command does

/goal [condition]

Defines a verifiable completion condition and continues work across turns until satisfied or stopped.

/loop [interval] [prompt]

Repeats a prompt while the session is open; useful for periodic polling and status checks.

Stop or modify

/goal clear removes the active goal; stop an active loop from the session interface.

How to apply safely

Make success measurable

Specify tests, exit codes, report paths and limits; avoid vague “make it good” goals.

Prefer read-only polling

Check batch or CI status without repeatedly submitting the same jobs.

Limit automation

Require approval for destructive actions, expensive runs or external publishing.

CMS example: /goal all unit tests pass and no output histogram differs beyond the agreed tolerance.

/hooks & safety controls

GUARDRAILS · Enforce checks at predictable tool events

What hooks do

Event-triggered automation

Hooks run scripts or checks at defined lifecycle events, such as before/after tool use.

Pre-tool protection

A PreToolUse hook can reject or modify risky tool inputs before execution.

Post-tool feedback

Run linting or lightweight tests after edits to surface problems immediately.

Practical guardrails

Protect sensitive information

Block leaking secrets in commands, logs or network requests.

Constrain filesystem changes

Prevent writes outside approved directories and guard dangerous commands.

Preserve oversight

Use permissions plus hooks; inspect logs and test the hooks themselves.

CMS example: block accidental deletion of ntuples, require confirmation for CRAB submissions and run formatting after edits.