Brookhaven National Laboratory

U.S. DEPARTMENT OF ENERGY

Bamboo MCP

Updates since the previous meeting

MEETING

REDWOOD General · October 12, 2026

SPEAKER

Paul Nilsson

PROJECT

U.S. DOE · REDWOOD

Highlights of Recent Developments

TOOL RETRIEVAL, ON BY DEFAULT

60% **less tool text* per call**

The planner (the LLM step that picks tools) now sees the 10 most relevant of 22 tools. On 120 test questions the needed tool was always among them.
*Tool text = name, description, parameter

Agent building blocks

Small building blocks let outside AI agents assemble their own log analysis. A reference agent matched panda_log_analysis in all 19 test scenarios.

Monitor integration

A web API lets the PanDA monitor offer an Analyze failure button. Results are cached, and repeated clicks on one job share one analysis.

T. Korchuganova

EVALUATION FRAMEWORK

2 of 5 **checks have results**

Checks the right tool is shortlisted, then chosen. First result: 60% smaller prompt, no measurable accuracy loss (gpt-oss-20b).

BigPanDA token support

Log analysis signs in to BigPanDA with an access token, sent only to the endpoints that need it.

Built for many users

One server can serve many clients at once, each with its own conversation state. Bamboo tracks daily LLM costs and can enforce a spending limit.

Currently using gpt-oss-20b and qwen3.8-27b-fp16 provided by CERN (will compare results presented here with Anthropic models later).

How Bamboo MCP answers a question

1 · INPUT

Question

Asked in plain English, for example:

```
Why did job 6612345678 fail?
```

2 · DECIDE

Planner

An LLM that picks which tools to call. Common questions skip it: fixed rules route those directly.

3 · ACT

Tools

Small programs with one job each, such as job status or log analysis, offered through MCP, a standard way to expose tools to AI systems.

4 · OUTPUT

Answer

The tool results, written up as a reply.

Not one-shot

Bamboo does not simply accept one LLM reply: plans are checked against a schema, and in log analysis each result decides the next file to read.

Selection

Choosing the right tool for a question. A wrong choice sends the question to a tool that cannot answer it.

Catalog

The list of every tool with its text. The planner reads the whole list before choosing, and that list is where this talk goes next.

How big should a tool be?

Fewer, larger tools

Best when an LLM planner picks the tools, one or several per question

STRENGTHS

- One call does the whole job: fewer round trips
- Expert knowledge stays inside the tool
- A short catalog keeps the planner accurate

COSTS

- The caller only receives the final answer
- Each new use case adds another option to the tool, so it keeps growing

More, smaller tools

Best when an agent chains the steps itself

STRENGTHS

- The caller sees the data fetched at every step
- Steps can be skipped, repeated or run for many jobs
- An outside agent can drive the work itself

COSTS

- More tools crowd the planner's catalog
- More calls instead of one: more delay and more tokens
- A caller that chains steps badly can still get a plausible but wrong answer

Tool size depends on who is calling, not on the task.

Bamboo ships both from one implementation: the planner gets the large tool, outside agents can get the small steps.

Log analysis split into five steps

BEFORE · ONE TOOL

`panda_log_analysis`

One call in, one answer out. Which files to read, how much of each and which error trace to trust are all decided inside, so the caller only receives the final verdict.

AFTER · THE SAME WORK AS FIVE STEPS

`fetch_metadata`	Get the job's details
`plan_fetch`	Choose which file to read next
`fetch_text`	Read that file
`classify`	Name the type of failure
`list_files`	See which log files exist

Our planner

Sees the single tool, so its catalog stays at 22 tools.

An outside agent

Another AI program can be offered the five steps instead: 26 tools, one swapped for five.

Same code

Both run the same code. The setting BAMBOO_TOOL_PROFILE chooses which is offered.

Other tools

We split a tool only when an outside agent needs to chain its steps.

The five steps give the same answer

Splitting the tool did not change the result.

plan_fetch chooses as it goes

It picks the next log file from the job's details and from what the previous file said.

Pilot error	Files read, in order	Rounds
1151 · stage-in timeout	pilotlog.txt	1
1305 · payload failure	setup.stdout, then payload.stdout	2

A round is one choice of which file to read next. Most failed jobs are one of these two cases. The payload case asks again only after seeing that the setup log was clean.

Tested equivalent

19 scripted failure scenarios run through both the single tool and the five steps. Each checks the verdict, which files were read and in what order. All 19 match.

Running live

The five steps have run on real failed jobs at aipanda033: the two cases in the table. A live side-by-side with the single tool is still to do.

The planner reads every tool, every time

SIZE OF EACH TOOL'S CATALOG ENTRY (TEXT), 22 TOOLS

Orange: the prompt-log search tool, 9,929 characters, a third of the 29,903 in total. It is long because it carries a manual for writing OpenSearch queries: a schema, rules and 14 examples. Grey: the other 21 tools, 202 to 1,947 characters each; log analysis is one of the smaller ones, at 774.

More choices, more mistakes

Attention is spread over more candidates, and look-alike tools get confused: live job counts versus historical job statistics.

Every question pays

Each question carries the whole catalog in its prompt, however simple it is. Adding a tool makes every prompt longer and every choice harder, even for questions that never need that tool.

A common rule of thumb says tool choice gets unreliable beyond roughly 25 tools (not measured). The planner sees 22 today; will increase.

Shortlist 10 of 22 tools for each question

22 tools in the catalog. The 10 filled squares are the ones the planner is shown for this question.

Score every tool

Score each tool by how many of the question's distinctive words appear in its name, description and parameters; rare words count most. Keep the top 10. The method is called BM25. It dates from the 1990s and is the default ranking in search engines such as Elasticsearch and OpenSearch.

Keep a safety net

The two documentation-search tools are never dropped, so every question keeps a last resort. The prompt's hand-written “use tool X when…” advice is trimmed to match, so it never recommends a tool that was cut.

Why not embeddings?

Embeddings match meaning rather than words and are the usual basis of RAG: retrieve the relevant items first, then ask the LLM. Embeddings found the right tool slightly less often (97.5% vs 99.2%) and needed a larger prompt.

SECTION 02 · THE RESULT

Tested on 120 questions

Note: the perfect scores are preliminary and probably optimistic: the catalog is small (22 tools) and the questions were written by us. More tools are coming, which will make the task harder.

22 → 10

tools shown to the planner

40%

of the former prompt size

120/120

questions had the needed tool shortlisted (recall 1.000)

59/59

deliberately confusable questions, scored separately

What it shows

The 120 questions were written in-house against the ATLAS tool catalog, each labeled with the tools a correct plan needs. For each one: was every needed tool on the shortlist? Yes, every time, with 40% of the former prompt. Showing all 22 tools scores 100% by definition, so the aim was to match that with far less.

What it does not show

Whether the planner then picks that tool; the evaluation framework that follows measures this. Also, 87 of the 120 questions come from our own question cheat sheet, and the last failing case was fixed by rewriting a tool description, then checked on the same questions. Treat the perfect scores as optimistic.

Bamboo MCP Evaluation

Bamboo MCP answers questions by choosing from a list of tools. A longer list makes the choice harder, so a shortlist step was added.

THE INPUT
Question
“Why did my job fail?”

→

CHECK 1
Shortlist
Keep the 10 most relevant of 22 tools

→

CHECK 2
Choice
The LLM planner picks which tool to use

→

THE OUTPUT
Answer
Written from the tool’s results. Not measured yet.

Check 1 · Shortlisting
Was the right tool on the shortlist? [Yes, 120/120 s.9]

Check 2 · Choosing
Given the shortlist, did the planner pick it?

The Five Checks

The 120 questions are labeled with the tools a correct plan needs; 59 are flagged hard (confusable) — more than one tool is a plausible answer, so the planner has to discriminate rather than just recognize the topic.

All five use the same test questions, each marked with the tools a correct answer needs.

Check	The plain question	Status
Tool retrieval	Was the right tool put on the shortlist?	Shipped
Selection accuracy	Did the LLM planner then choose it?	First results
Fast-path routing	Do simple questions take the quick route?	Next
Scale curve	Does it still work with 40–50 tools?	Needs more tools
Answer quality	Right document found, then right answer?	Planned

Results for check 2

CHECK 2 · CHOOSING (see slide 9 for check 1 results)

82% vs 83%

of attempts chose the right tool: all tools shown (82%) vs. shortlist only (83%)

Same AI model both times (gpt-oss-20b). The AI can answer differently each time, so each of the 120 questions was asked 5 times: 120 × 5 = 600 attempts per setup. The gap is smaller than that natural variation: a tie.

Cutting the prompt by 60% did not cost accuracy for gpt-oss-20b.

60% = the share of tool text the shortlist removed.

What the numbers showed

1 in 10

Broken or missing plans

About 1 in 10 gpt-oss-20b attempts ended before any tool was chosen: a bigger lever than the shortlist.

60%

Higher score, less steady

qwen3.8-27b-fp16 scored higher than gpt-oss-20b (86% vs 82-83%), but gave the same outcome in all 5 attempts on only 60% of questions (73% for gpt-oss-20b) (different tool-list versions; re-run pending).

8–9 pts

Hard questions are harder

Questions flagged hard (confusable) scored 8–9 points lower for both models, so the flag is doing real work.

Why every number is labeled, and what comes next

Same test, two scores

The same method scored 91% on one copy of the tool list and 92% on another. The list had changed slightly: 5 versions in about a week.

What the label does

Each result is saved with a code, like 8f67488c6d95, for its exact tool list, questions, model and settings. Same code: compare. Different code: don't.

Not measured yet

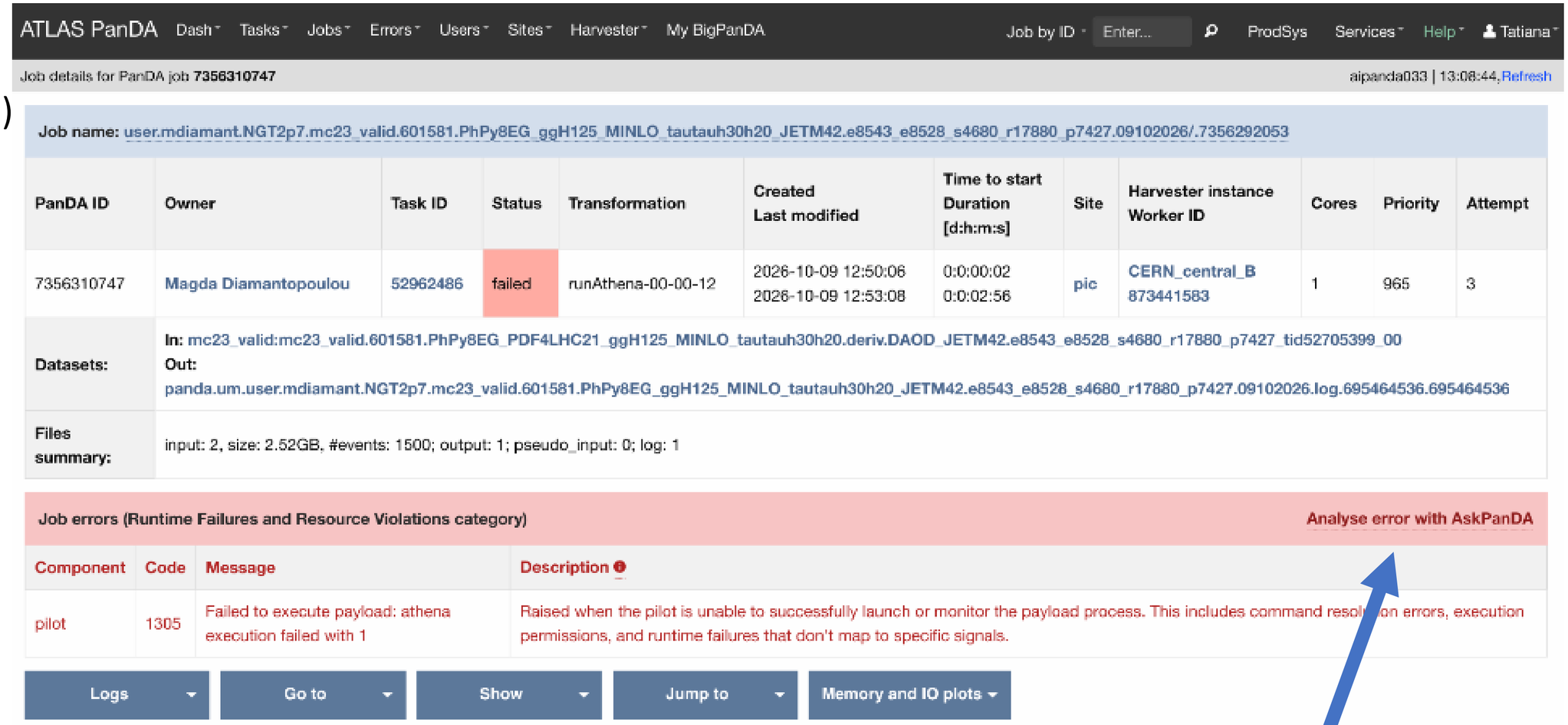
The second model (qwen) with a shortlist: re-run pending. Quality of the final answer: needs a trusted AI judge first.

Coming next

Both models on one tool list. Claude models from Anthropic as a further planner. Quick-route check. Testing with 40–50 tools.

Screenshots

PanDA Monitor integration (T. Korchuganova)



”Analyze” link

PanDA Monitor integration

- Response on same page

(T. Korchuganova)

Analysis can be rated

Error analysis summary:

Job 7356310747 — Failure Analysis

Classification: `payload_error`

Root Cause

The job failed immediately at **payload startup** with a Python `ModuleNotFoundError`. The traceback is unambiguous:

```
File "/srv/workDir/bin/Taus_final", line 5, in <module>
    from SQUEAK.hub import *
ModuleNotFoundError: No module named 'SQUEAK'
```

The payload script `Taus_final` attempts to import a package called `SQUEAK` at line 5, but this module is not present in the Python environment provided by Atlas-25.0.68 / LCG_109a_ATLAS_11 on the `pic` site.

Key Facts

- **Failure type:** `payload_error` (pilot error code 1305, exe error code 5406)
- **Exception:** `ModuleNotFoundError: No module named 'SQUEAK'` — raised at the very first import in the payload script, so **no actual processing occurred**
- **Runtime:** Only ~3 minutes — the job died instantly on startup
- **No core dump:** confirmed absent
- **Attempt:** 3 of 5 — this has failed repeatedly, suggesting a systematic environment issue rather than a transient one
- **Transformation:** `runAthena-00-00-12` with a custom payload script `Taus_final`

Diagnosis

The `SQUEAK` package is a **non-standard, user-supplied Python module** that is not part of the ATLAS release or LCG stack. It must be either:
1. Shipped as part of the job's input sandbox/tarball and is missing or not being unpacked correctly, **or**
2. Expected to be installed in a custom Python environment that is not being set up before the payload runs

Suggested Next Steps

1. **Check the job's input sandbox/tarball** — verify that the `SQUEAK` package (or its wheel/egg) is included and being correctly extracted into the working directory or `PYTHONPATH`
2. **Check** `PYTHONPATH` in the log — the environment dump is present but `SQUEAK` does not appear in any visible path; confirm it is being added before `Taus_final` is invoked
3. **Inspect the task configuration** (task ID 52962486) — check whether the `SQUEAK` module is supposed to be bundled via `--extFile`, `--useAthenaPackages`, or a similar mechanism
4. **Check earlier attempts** (attempts 1 and 2) — if they failed identically, the issue is in the task definition, not site-specific
5. **If** `SQUEAK` **is a private package**, ensure it is properly packaged and submitted with the task; it cannot be assumed to exist on worker nodes

Links:
- BigPanDA Monitor
- Setup Log
- Payload Log

Rate this analysis: