
AI for ADC



Ilija Vukotic
University of Chicago



Wishes

I'm interested in exploring how AI can answer questions like:

Is my site running smoothly?

Are there any problems with my site?

Are my jobs running correctly?

What's the status of this task?

What problems are associated with this task?

How many slots with 32GB/core are currently available?

Essentially, we want AI to handle some of the standard questions typically directed to Rod, Ivan, or even Tadashi. And I would like the ability to communicate issues in advance before shifters and users.



Some general questions

Why Ivan, Rod, Tadashi (IRT) receive so many questions:

- insufficient/obsolete documentation? Users not reading it?
- Ineffective monitoring?
- System errors? (site busy, network issues, lost data, lost CVMFS, bad site config, bad xcache ...) Bad error reporting?
- No or ineffective automatic issue resolution (using other site, other replica,...)

Do we want to help IRT by answering instead of them?

Do we want to help users with job submissions?

Do we want to help sites run better?

Do we want to help HC detect issues faster?

All of the above?



Specific questions ...

1. Is my site running smoothly?
2. Are there any problems with my site?
3. Are my jobs running correctly?
4. What's the status of this task?

5. What problems are associated with this task?

6. How many slots with 32GB/core are currently available?

7. Essentially, we want AI to handle some of the standard questions typically directed to Rod, Ivan, or even Tadashi.

8. And I would like the ability to communicate issues in advance before shifters and users.

Who is you, which sites your own, what's your username?
Can't this be answered by looking at bigpanda?

Don't we have logging and error reporting?

Is this even available? Is our source trustworthy?

Do we have a way to get a list of questions directed at them in let's say last 3 years? Do we have their answers? Do we know if their answers helped?

Isn't HammerCloud made for that?



Options

1. Fix original issues.
2. A band-aid AI.
 - Helps in detecting, understanding issues
 - Does not solve or mitigate
3. Medium level AI.

Have it help WFMS in brokering tasks, moving datasets around, etc.

4. High level AI

AI interprets what a physicist want, creates queries, runs jobs/filters, returns to user a small chunk of data to be locally analysed.



Options - Fix original issues

An ideal solution.

If that's hard, then the system is probably over-engineered, complex, or grew “organically”.

Cons:

- A very significant effort.
- Big changes that we probably don't have a manpower or stomach for.



Options - band-aid AI

1. Is my site running smoothly?
2. Are there any problems with my site?
3. Are my jobs running correctly?
4. What's the status of this task?

Create an AI assistant that can lookup panda jobs and tasks information in ES, interpret results.

Can be done quickly (1-2 months) and cheaply using RAG approach

- We already have all the data.
- Create web frontend, create an assistant
- Make functions that assistant can use to mine the data
- Assistant in the cloud, no special infrastructure needed

Cons:

- Users might be completely unaware of its capabilities and simply not use it.



Options - band-aid AI

5. What problems are associated with this task?

- Have all logging info from panda, rucio, FTS, jobs streamed through a fancy filtering:
 - Noise reduction
 - Time synchronization
 - Enrichment, error code mapping
- Store it in Elasticsearch in vector form
- Continuously run anomaly detection (eg. isolation tree)
- Continuously run root cause analysis (LLM)
- Run issue prioritisation, tagging (LLM)



Options - band-aid AI

Cons:

- Effort to get ALL the logs. Effort to digest it.
- More hardware needed for ES.
- Large effort to make and custom tune a model. Quite a few GPUs for quite some time.
- Uncertain outcome.
- Running it would require special on-premise hardware (eg. NVidia DIGITS)
- It would be one more system needing a long-term support.



Options - medium level AI

Have AI agents helping running our systems.

- Each agent would have full access its system eg. run all Rucio commands - from looking up replicas, creating and deleting rules, etc. eg. submit and cancel tasks
- Have continuous access to system state and actions.
- Fine tune different agent for each system.
- Agents are independent but could communicate with each other.
- Run on-premise .



Options - medium level AI

Have AI agents helping running our systems.

Cons:

- While it is easy to make an agent run as a user or even a superuser, probably systems don't expose its state well enough.
- There is a possibility that things get out of control.
- Quite a bit of effort to develop.
- Large hardware requirements.
- Uncertain outcome.



Options - high level AI

AI interprets what a physicist want, creates queries, runs jobs/filters, returns to user a small chunk of data to be locally analysed.

- Have a fast, distributed system to filter/augment the data and return to users exactly what they need, where they need it, in the format they want. Something like ServiceX/Y.
- Have AI learn the language and enable it to create queries by itself.

Cons:

- Need to make Filtering work as advertised. Need it distributed. Need the language correctly and completely described.
- Teaching an LLM how to use it (relatively simple).
- User adoption can be shaky.



Conclusion

- AI is not a magic bullet. Can't fix broken systems.
- It could help with monitoring systems - but nothing that can't be done manually.
- It could help in understanding system specific issues and issues coming from interactions between systems.
- It could help running systems. First as a “shadow” admin, then as a full admin (with a human signoff for big destructive operations). Systems need to be ready for that.
- It could be a full analy replacement.
- It would require large effort to develop and non-negligible support cost.



Backup slides
UC AF assistant

Idea

We have three analysis facilities in US ATLAS: University of Chicago, BNL, and SLAC.

We offer a lot of services: login/interactive work, Jupyter, Batch on a lot of hardware (CPU, GPU, storage).

We are supporting more and more users.

Create a chatbot to help users who don't like reading the documentation, discourse forums eg.

How do I send a condor job? What nodes I can log in? How much scratch space do I get?

Make it able to answer runtime questions eg.

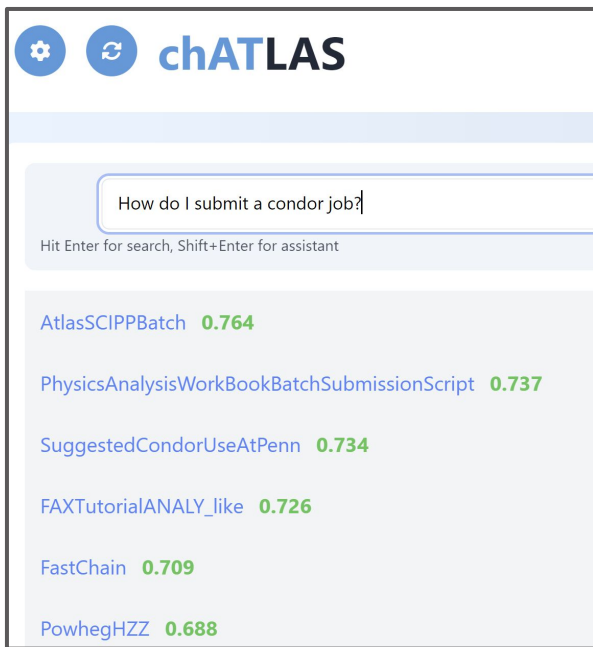
What are currently available GPUs, what state are my submitted jobs in.

Make it a able to answer accounting questions that are relevant to Analysis Facility managers eg.

Make a pie chart showing a number of unique users that logged in at different analysis facilities.



Other efforts



chATLAS

How do I submit a condor job?

Hit Enter for search, Shift+Enter for assistant

- AtlasSCIPPBatch 0.764
- PhysicsAnalysisWorkBookBatchSubmissionScript 0.737
- SuggestedCondorUseAtPenn 0.734
- FAXTutorialANALY_like 0.726
- FastChain 0.709
- PowhegHZZ 0.688

Uses different source: TWiki,
CDS, Indico
Does not RAG.

UC AF assistant

how do I submit a condor job?

To submit a Condor job at the University of Chicago Analysis Facility, follow these steps:

1. Prepare Your Job Script

First, create a script for the job you want to execute. Save it in a file called `myjob.sh`. Here is an example:

```
#!/bin/bash
export ATLAS_LOCAL_ROOT_BASE=/cvmfs/atlas.cern.ch/repo/ATLASLocalRootBase
export ALRB_localConfigDir=$HOME/localConfig
source ${ATLAS_LOCAL_ROOT_BASE}/user/atlasLocalSetup.sh
# Add further commands as needed, e.g., lsetup root, rucio, athena
```

2. Create the Submit File

Next, create a Condor submit file named `myjob.sub`:

Uses only AF documentation as a source.
Does RAG (retrieval augmented generation).



Technology choice

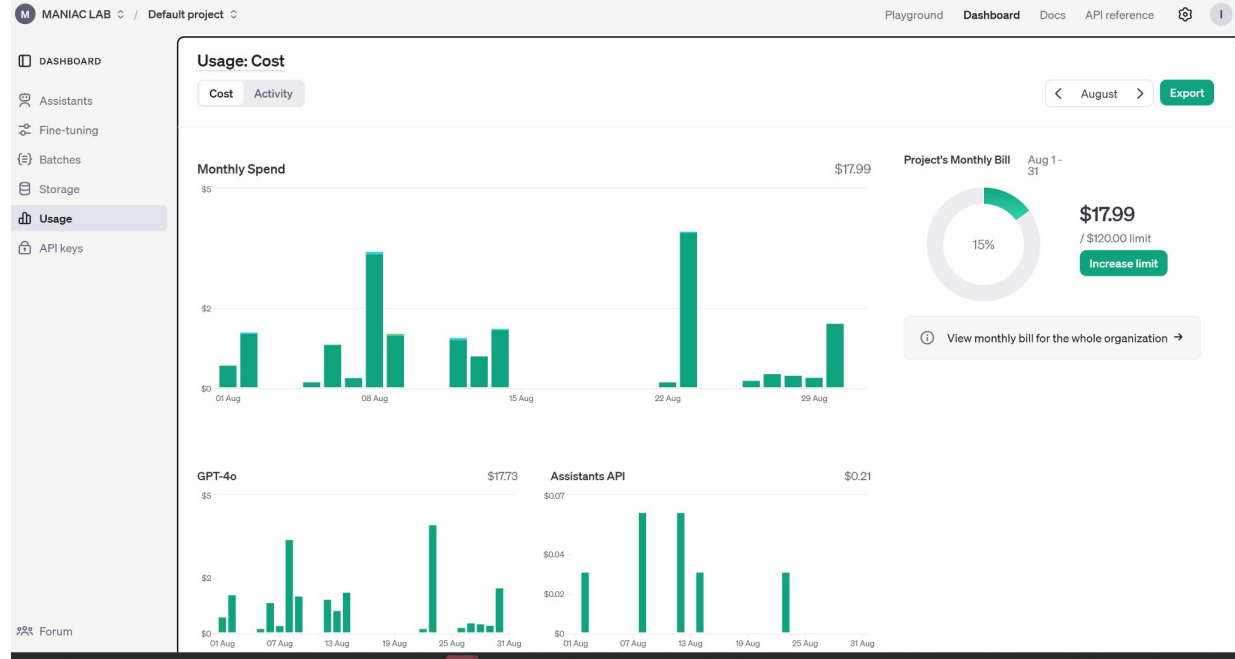
These days there are options.
We went with the easiest -
OpenAI.

Using GPT-4o.

Cost is not prohibitive.

The only data stored in
OpenAI is our documentation.

They don't learn from our
prompts, but we do.



Frontend

- A node.js application with Express and Pug.
 - Completely async, multithreaded, event driven.
- We store all prompts/requests in Elasticsearch.
- Access limited to users logged in to AF.
 - Assistant already knows who you are.



Answering based on documentation

Sounded trivial... just upload .md files from our [mkdoc site](#) add it to vector store and that's it.

- Then it appeared some of the data was in [HTML](#) (used a web converter to transform it to md).
- But we had an another set of [documents](#) with wider scope and hard to limit to only UC AF.
- Have to keep track of which imported md file can be accessed at which URL so that references in responses would be correct.
- Even a brief testing revealed deficiencies in the documentation that were fixed by Lincoln B.

We use Discourse forums to provide user support. All of the relevant data (topic, date, state, all replies, etc.) have been pulled out of the database by Louis Pelosi. We will try to use it to improve answers. Many potential issues: some threads are obsolete, some issues were temporary, some problems were user caused.



Answering based on Elasticsearch data

aka RAG (Retrieval Augmented Generation)

We don't ship our data to OpenAi (it would be way too costly).

We provide it with functions it can call if it needs data from the EL.

We explain what are parameters and what the function will return.

We had to add some scripted variables, change names to make them selfdescriptive.

Example function description I

```
function: {
  name: "get_elasticsearch_data",
  description: `This function assists in answering queries that require access to an Elasticsearch
database. It returns results for a given Elasticsearch DSL query. Before creating the query, use other
tools to obtain the document mapping. Typically, you should use aggregation searches.` ,
  parameters: {
    type: "object",
    required: ["query"],
    properties: {
      query: {
        type: "object",
        description: "Elasticsearch query as a json document.",
      }
    }
  }
}
```

Basically saying: if you tell me your query, I will execute it and return results as JSON.



Use below index mapping and reference document to build Elasticsearch query concerning condor jobs:

Index mapping:

```
{
  "Runtime": { "type": "integer" },
  "@timestamp": { "type": "date" },
  "state": { "type": "keyword" },
  "cluster": { "type": "keyword" },
  "Id": { "type": "keyword" },
  "kind": { "type": "keyword" },
  "users": { "type": "keyword" },
}
```

Reference elasticsearch document:

```
{
  "Runtime": 312,
  "@timestamp": "2024-08-09T15:30:14.112673488Z",
  "state": "held",
  "cluster": "BNL-AF",
  "Id": "3893714.0",
  "kind": "condorjob",
  "users": "4395b9c3"
}
```

Always require field "kind" to have value "condorjob".

Here meaning of the variables:

- * 'Runtime' is a number seconds that the job was running
- * 'state' is a state of the job in that moment. It can have values "held", "idle", "finished", "running", "removed".
- * 'users' contains username of user that submitted the job
- * 'cluster' denotes analysis facility site

If any field has a "keyword" type, just use field name instead of field.keyword.`;

Explaining data

Formatting is important: new lines,
tables, etc.
Good english helps but it is not very
sensitive to typos, grammar.



Here a few examples of Query DSL concerning Condor Jobs:

1. User Query - Number of condor jobs per analysis facility and per day during last 7 days.

Elasticsearch Query DSL:

```
{
  "size": 0,
  "query":{
    "bool":{
      "must":[
        {"term":{"kind":"condorjob"}},
        {"range":{"@timestamp":{"gte":"now-7d/d","lte":"now/d"}}}
      ]
    }
  },
  "aggs":{
    "average_users_per_cluster":{
      "terms": {"field": "cluster"},
      "aggs":{
        "jobs_per_day":{
          "date_histogram":{"field":"@timestamp", "fixed_interval":"1d"},
          "aggs":{
            "status": {
              "terms": { "field": "state" }
            }
          }
        }
      }
    }
  }
}
```

2. User Query - Average run time of condor jobs per analysis facility and per day during last week.

Example queries (few shots learning)

- Examples are important otherwise it could use older ES syntax.
- It takes serious time to create correct and meaningful queries.



Function calls

OpenAI events tells us what function it wants called and with what parameters. We evaluate it and return results to OpenAI. OpenAI interprets the result.

It frequently asks for simultaneous execution of several functions. This significantly improves response time.

A user session creates a single OpenAI thread and the thread is following user over time and tabs. Some functions will be called only once per user (eg. function explaining ES index), so subsequent queries are slightly faster.



Development experience

Application is relatively straightforward to make.

Takes some experience to make it optimal.

RAG part is very different from regular programming:

- Can't really automatize testing as responses even when correct are always a bit different.
- LLM will hide a lot of errors in code:
 - phrase an answer to be technically correct
 - will try multiple times in different ways until it gets result
 - give vague approximate answers
- Once RAG code is debugged, it is amazingly reliable. We haven't noticed any hallucinations. Sometimes it is a bit overconfident and just look at query examples and not at all at document examples.

Development experience II

We did not spend time in trying to limit nefarious prompts

- That should be on OpenAI
- We have records of all prompts/responses with user names.

Data discovery

- Users should have an idea of what kind of data we have so they know what they can ask.
- Will they try prompt like: “What data do you know about?”

Correct question phrasing

- Even users that know data inside out will frequently ask questions that have no sense or can not be answered with with the data we have.

Eg. “Who is the biggest UC AF user?”

- Is a person with 1000 condor job a bigger user than someone with 30 jupyter notebooks or someone logged in 24/7?
- Over what period?
- Most CPU spent? What about GPU usage?



Future - AF assistant

- Now it is open for users. Learn from questions and responses:
 - What and why it does not answers correctly
 - What are we missing in our documentation
 - Look for signs of abuse
- Get more experience and check scaling to more complex datasets.
 - Make it more robust against OpenAI issues eg.

APIError: The server had an error processing your request. Sorry about that! You can retry your request, or contact us through our help center at help.openai.com if you keep seeing this error. (Please include the request ID req_c779a1a154c805c3151598cb22e99ab0 in your email.)
- Check effects of fine-tuning
- Test o1 LLM model

Future - next applications

- [Soteria project](#) - answer questions about docker images, CVEs, etc. ([chatbot](#))
- PhysLite analyzer
 - Answers prompts like:
Make a plot of Higgs invariant mass where Higgs decays into four leptons based on period X datasets.
 - Teach it about variables in physlite files.
 - Make a query language (or reuse func_adl)
 - Make an infrastructure that will extremely quickly extract the data (something akin to ServiceX).