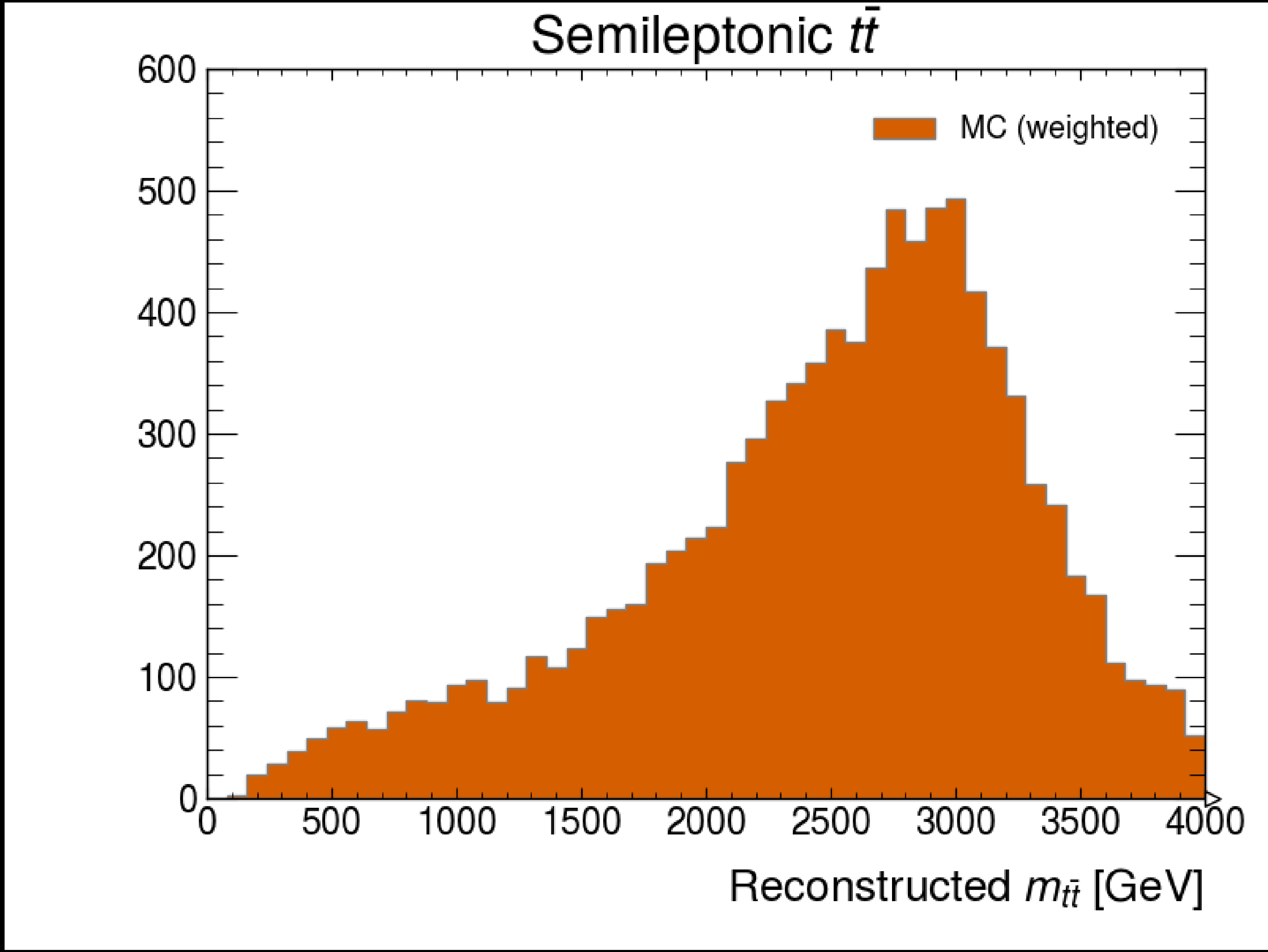
Vibe Plotting - Using LLM's to make physics analysis plots

G. Watts (UW/Seattle)

LLM/AI tool use in particle physics

2026-04-29



Context

arxiv


SciPost Physics — Submission

Agents of Discovery

Sascha Diefenbacher[1], Anna Hallin[2],
Gregor Kasieczka[2], Michael Krämer[3], Anne Lauscher[4], Tim Lukas[2],

1 Physics Division, Lawrence Berkeley National Laboratory, Berkeley, USA
2 Institut für Experimentalphysik, Universität Hamburg, Germany
3 Institute for Theoretical Particle Physics and Cosmology, RWTH Aachen University, Germany
4 Data Science Group, Universität Hamburg, Germany

February 18, 2026

Abstract

The substantial data volumes encountered in modern particle physics and other domains of fundamental physics research allow (and require) the use of increasingly complex data analysis tools and workflows. While the use of machine learning (ML) tools for data analysis has recently proliferated, these tools are typically special-purpose algorithms that rely, for example, on encoded physics knowledge to reach optimal performance. In this work, we investigate a new and orthogonal direction: Using recent progress in large language models (LLMs) to create a team of *agents* — instances of LLMs with specific subtasks — that jointly solve data analysis-based research problems in a way similar to how a human researcher might: by creating code to operate standard tools and libraries (including ML systems) and by building on results of previous iterations. If successful, such agent-based systems could be deployed to automate routine analysis components to counteract the increasing complexity of modern tool chains. To investigate the capabilities of current-generation commercial LLMs, we consider the task of anomaly detection via the publicly available and highly-studied LHC Olympics dataset. Several current models by OpenAI (GPT-4o, o4-mini, GPT-4.1, and GPT-5) are investigated and their stability tested. Overall, we observe the capacity of the agent-based system to solve this data analysis problem. The best agent-created solutions mirror the performance of human state-of-the-art results.

1 Introduction

Large-scale experiments in high-energy physics such as those at the Large Hadron Collider (LHC) produce increasingly complex datasets that require sophisticated, multi-stage analysis workflows to extract results – including subtle potential signals of new physics – from high-dimensional data representations (see [1,2] for two examples). Although mature tools and shared computing resources are available, the analysis process itself is becoming more demanding. Workflows must be repeatedly re-optimized for changing detector conditions, calibration procedures must be adjusted and background estimates must be carefully validated. These developments mean that an increasing proportion of scientific output relies on detailed coordination and significant human effort. Without improvements in efficiency, this growing complexity could limit the potential of current and future high-energy physics experiments.

Current automation efforts in high-energy physics tend to focus on individual tasks such as reconstruction, calibration, and statistical analysis. However, they provide limited support for coordinating entire analysis workflows. As a result, a significant amount of time is spent on integration and bookkeeping tasks, such as connecting tools, synchronizing configurations, converting formats and tracking parameters. This slows down the iterative process and makes

1

arXiv:2509.08535v2 [hep-ph] 17 Feb 2026

arxiv


Automating High Energy Physics Data Analysis with LLM-Powered Agents

Eli Gendreau-Distler,[1,2,*] Joshua Ho,[1,2,†] Dongwon Kim,[1,2,‡] Luc Tomas Le Pottier,[1,2,§] Haichen Wang,[1,2,¶] and Chengxi Yang[1,2,**]

[1]Department of Physics, University of California, Berkeley, Berkeley, CA 94720, USA
[2]Physics Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

We present a proof-of-principle study demonstrating the use of large language model (LLM) agents to automate a representative high energy physics (HEP) analysis. Using the Higgs boson diphoton cross-section measurement as a case study with ATLAS Open Data, we design a hybrid system that combines an LLM-based supervisor–coder agent with the `Snakemake` workflow manager. In this architecture, the workflow manager enforces reproducibility and determinism, while the agent autonomously generates, executes, and iteratively corrects analysis code in response to user instructions. We define quantitative evaluation metrics – success rate, error distribution, costs per specific task, and average number of API call for the task – to assess agent performance across multi-stage workflows. To characterize variability across architectures, we benchmark a representative selection of state-of-the-art LLMs, spanning the *Gemini* and *GPT-5* series, the *Claude* family, and leading open-weight models. While the workflow manager ensures deterministic execution of all analysis steps, the final outputs still show stochastic variation. Although we set the temperature to zero, other sampling parameters (e.g., top-p, top-k) remained at their defaults, and some reasoning-oriented models internally adjust these settings. Consequently, the models do not produce fully deterministic results. This study establishes the first LLM-agent-driven automated data-analysis framework in HEP, enabling systematic benchmarking of model capabilities, stability, and limitations in real-world scientific computing environments. The baseline code used in this work is available at https://huggingface.co/HWresearch/LLM4HEP. This work was accepted as a poster at the Machine Learning and the Physical Sciences (ML4PS) workshop of the 39th Conference on Neural Information Processing Systems (NeurIPS 2025). The initial submission was made on August 30, 2025.

I. INTRODUCTION

Large language models (LLMs) and agent-based systems are increasingly being explored in scientific computing, with applications appearing in areas such as genomics and software engineering [1, 2]. Despite the inherently structured nature of collider-related data analyses and the potential advantages of automated, reproducible workflows, the usage of LLMs in high energy physics (HEP) is still at an early stage. Existing efforts in HEP have focused mainly on event-level inference or on domain-adapted language models [3–5], leaving open questions about how LLMs might participate directly in end-to-end analysis procedures.

Related work in agentic methodologies, including the *Agents of Discovery* study [6], demonstrates growing interest in structuring collider physics tasks through modular agent behavior. While such efforts examine agent-based reasoning in HEP contexts, they address different objectives and operational settings. Our study is situated within a workflow-management environment and considers how agentic components may be incorporated into a directed, reproducible analysis pipeline.

We introduce a framework in which LLM-based agents are integrated into a `Snakemake`-managed workflow [7]. Agent interventions are bounded to well-defined tasks such as code generation, event selection, and validation while the underlying directed acyclic graph (DAG) maintains determinism and provenance. This design enables a controlled evaluation of the practicality and reliability of agent-driven steps within a full collider-analysis setting.

II. BACKGROUND AND MOTIVATION

Research connecting LLMs with HEP is rapidly developing, though the existing literature remains concentrated on a small number of application areas – for instance, enhancements to event-level prediction tasks, improvements to simulation, or tools for navigating domain knowledge. A substantial body of work applies transformer-based architectures to established HEP tasks such as classification, regression, and generative modeling [8–11]. These studies demonstrate the effectiveness of modern model architectures within conventional workflows but are not designed to automate or restructure the broader analysis process.

LLMs have also been used to improve access to experiment-specific documentation and technical resources. Systems such as `Chatlas` [12], `LLMTuner` [13], and `Xiwu` [14] demonstrate the usefulness of natural-language interfaces for accessing experiment-specific documentation and technical information. Their focus, however, is on retrieval and interpretation rather than direct

* egendreaudistler@berkeley.edu
† ho22joshua@berkeley.edu
‡ dwkim@berkeley.edu
§ luclepot@berkeley.edu
¶ haichenwang@berkeley.edu
** cxyang@berkeley.edu

arXiv:2512.07785v1 [physics.data-an] 8 Dec 2025

arxiv


AI Agents Can Already Autonomously Perform Experimental High Energy Physics

Eric A. Moreno*[†1,2], Samuel Bright-Thonney*[‡1,2], Andrzej Novak*[§1,2], Dolores Garcia[♯3], and Philip Harris[♭1,2]

[1]Department of Physics, Massachusetts Institute of Technology
[2]NSF AI Institute for Artificial Intelligence and Fundamental Interactions
[3]CERN

April 2, 2026

Abstract

Large language model-based AI agents are now able to autonomously execute substantial portions of a high energy physics (HEP) analysis pipeline with minimal expert-curated input. Given access to a HEP dataset, an execution framework, and a corpus of prior experimental literature, we find that Claude Code succeeds in automating all stages of a typical analysis: event selection, background estimation, uncertainty quantification, statistical inference, and paper drafting. We argue that the experimental HEP community is underestimating the current capabilities of these systems, and that most proposed agentic workflows are too narrowly scoped or scaffolded to specific analysis structures. We present a proof-of-concept framework, *Just Furnish Context* (JFC), that integrates autonomous analysis agents with literature-based knowledge retrieval and multi-agent review, and show that this is sufficient to plan, execute, and document a credible high energy physics analysis. We demonstrate this by conducting analyses on open data from ALEPH, DELPHI, and CMS to perform electroweak, QCD, and Higgs boson measurements. Rather than replacing physicists, these tools promise to offload the repetitive technical burden of analysis code development, freeing researchers to focus on physics insight, truly novel method development, and rigorous validation. Given these developments, we advocate for new strategies for how the community trains students, organizes analysis efforts, and allocates human expertise.

1 Introduction

A typical experimental high energy physics analysis is a years-long endeavor, often spanning the majority of a graduate student or postdoc's time at an academic institution. For large collider experiments the process is algorithmic: a physicist (a) identifies an interesting measurement channel or new physics signature, (b) studies existing literature to understand how similar measurements have been done, (c) designs and validates an event selection procedure using Monte Carlo (MC) simulation, quantifies all relevant sources of systematic uncertainty, and finally (d) "unblinds" on real experimental data and performs statistical hypothesis tests to extract measurements or limits.

This process invariably involves writing thousands of lines of code to process data and apply standard, though not off-the-shelf, analysis techniques, most of which is structurally similar to code written in dozens of other analyses within the same collaboration. While this can be a useful exercise for developing programming skills, it often consumes a substantial fraction of a physicist's working time and demands little physics reasoning or insight. This is frequently exacerbated by the absence of high-quality, centralized, or up-to-date documentation for much of the core software infrastructure within experimental collaborations. The result is

*These authors contributed equally.
†emoreno@mit.edu ‡sambt@mit.edu §novaka@mit.edu ♯dolores.garcia@cern.ch ♭pcharris@mit.edu

1

arXiv:2603.20179v2 [hep-ex] 31 Mar 2026

Narrow Focus for this talk…

Can a LLM make plots we want?

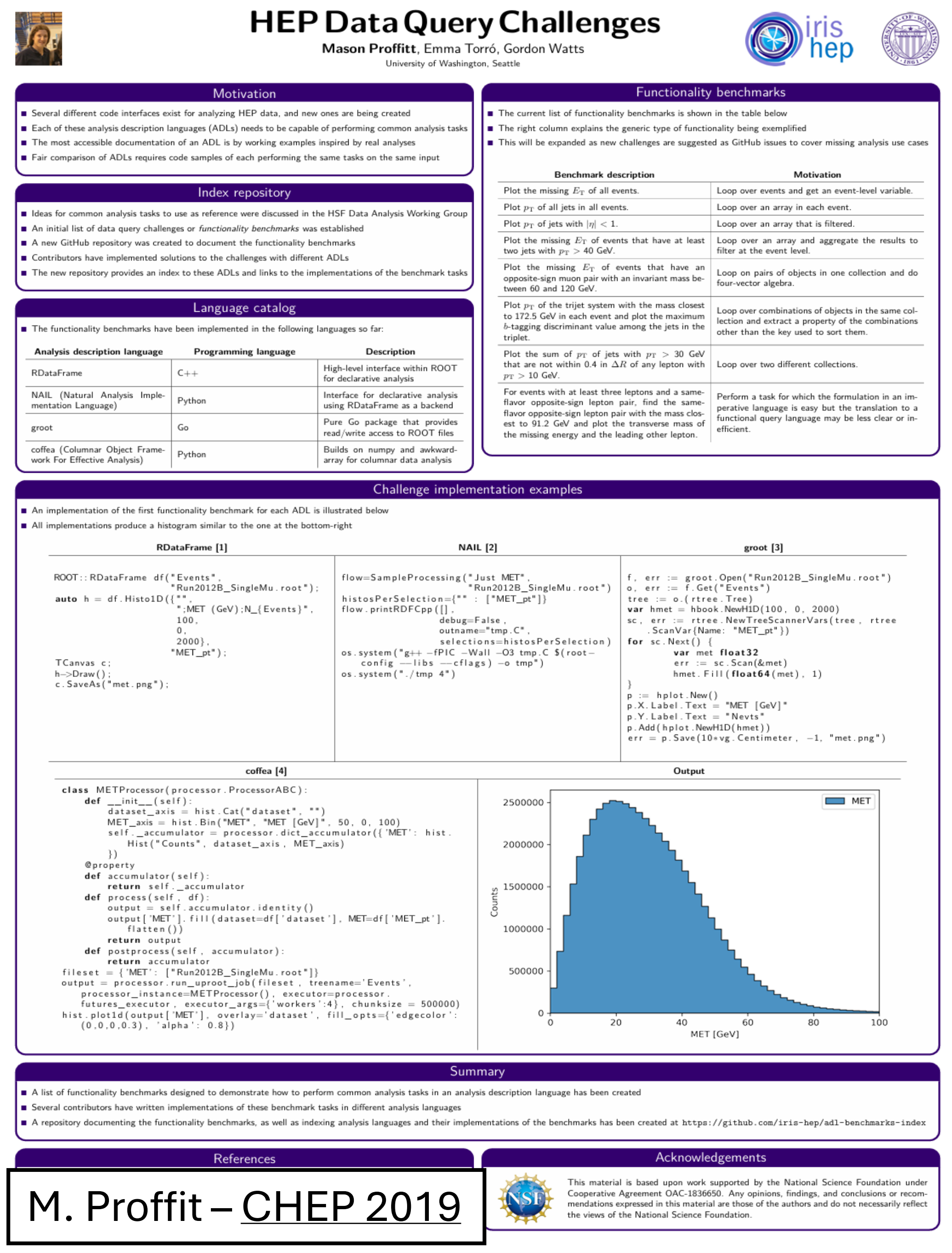
Can a LLM write working code to make a complex physics plot?

In the context of a specific experiment?

With just simple English queries?

What Plots?

Benchmark description	Motivation
Plot the missing E_T of all events.	Loop over events and get an event-level variable.
Plot p_T of all jets in all events.	Loop over an array in each event.
Plot p_T of jets with $\lvert\eta\rvert < 1$.	Loop over an array that is filtered.
Plot the missing E_T of events that have at least two jets with $p_T > 40$ GeV.	Loop over an array and aggregate the results to filter at the event level.
Plot the missing E_T of events that have an opposite-sign muon pair with an invariant mass between 60 and 120 GeV.	Loop on pairs of objects in one collection and do four-vector algebra.
Plot p_T of the trijet system with the mass closest to 172.5 GeV in each event and plot the maximum b-tagging discriminant value among the jets in the triplet.	Loop over combinations of objects in the same collection and extract a property of the combinations other than the key used to sort them.
Plot the sum of p_T of jets with $p_T > 30$ GeV that are not within 0.4 in ΔR of any lepton with $p_T > 10$ GeV.	Loop over two different collections.
For events with at least three leptons and a same-flavor opposite-sign lepton pair, find the same-flavor opposite-sign lepton pair with the mass closest to 91.2 GeV and plot the transverse mass of the missing energy and the leading other lepton.	Perform a task for which the formulation in an imperative language is easy but the translation to a functional query language may be less clear or inefficient.



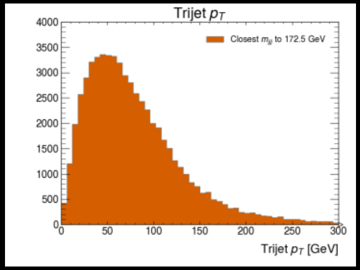
M. Proffit – CHEP 2019

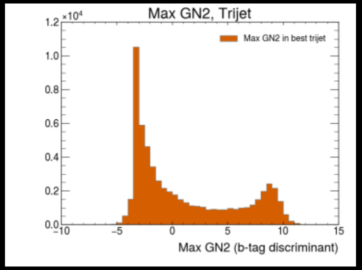
What Does The LLM Have To Think Through?

Question 6: For events with at least three jets, plot the pT of the trijet four-momentum that has the invariant mass closest to 172.5 GeV in each event and plot the maximum b-tagging discriminant value among the jets in this trijet

1. **Loop** over all the events
2. **Fetch** the jets from the event if it has at least 3 jets
3. **Form** all 3-pair combinations
4. **Pick** the 3-pair combination closest to mass 172.5 GeV (top mass)
5. **Fill** an appropriately defined histogram
6. **Fetch** the b-tagging discriminate for all three jets
7. **Fill** an appropriately defined histogram with the largest discriminate value.
8. **Plot** the histograms

"Loop over combinations of objects in the same collection and extract a property of the combinations other than the key used to sort them."





Pick a Technology Stack

Workflow

ServiceX

Get data (ATLAS) off the GRID and convert it
to a columnar format in a scalable way.
(GRID Data/Skimming Access via REST API)

Awkward Array

VECTOR

Manipulate columns, filter, 4-vector algebra, etc.
"numpy" for jagged data

Boost Histogram

Histogram manipulation, plotting (matplotlib),
HEP formatting, etc.

Workflow

ServiceX

Get data (ATLAS) off the GRID and convert it to a columnar format in a scalable way.
(GRID Data/Skimming Access via REST API)

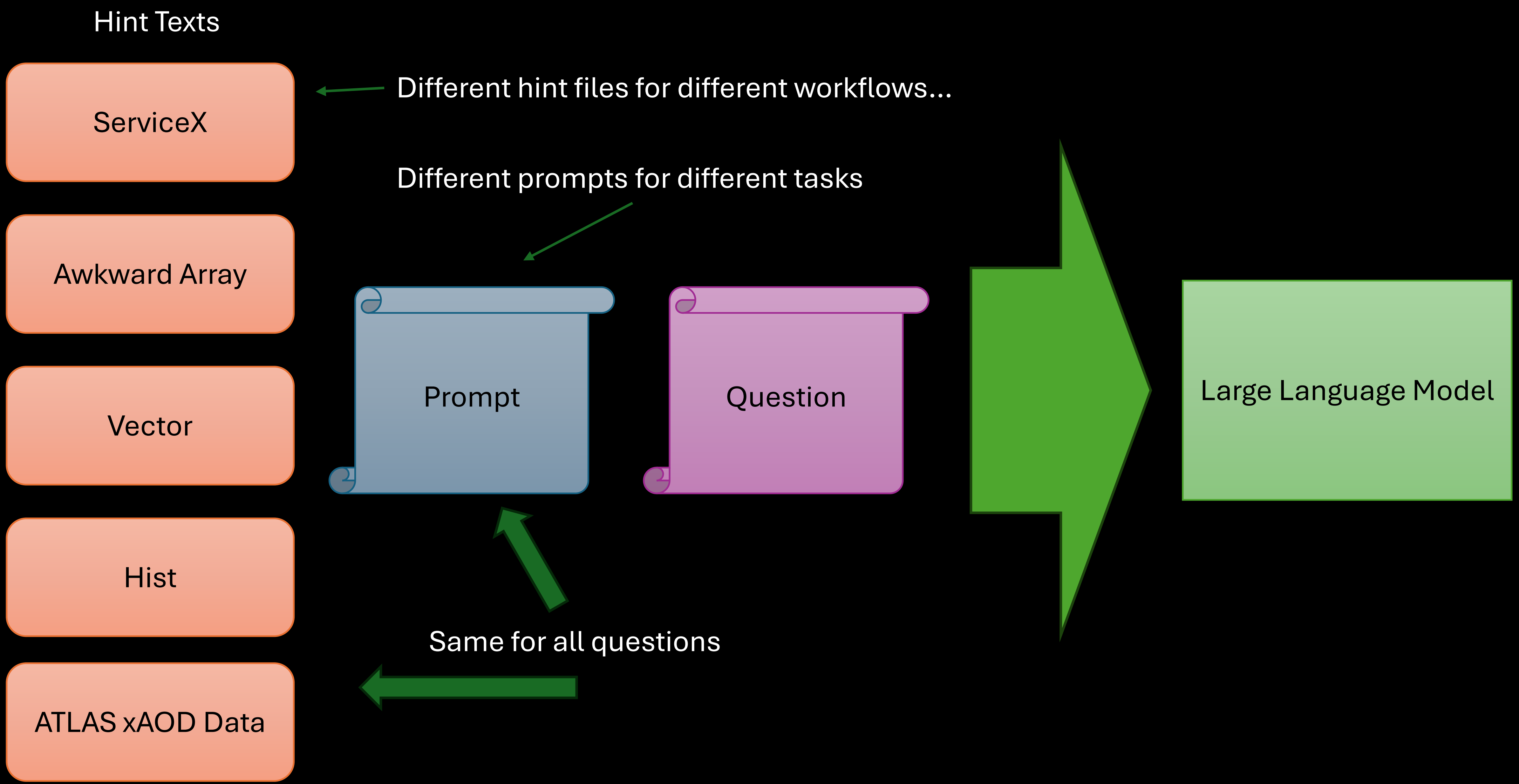
ROOT Data Analysis Framework

RDataFrame
Manipulate columns, filter, 4-vector algebra, etc.

Histogram manipulation, plotting (matplotlib), HEP formatting, etc.

ServiceX

Because we get quick, easy, immediate access to experiment GRID data


Hint Texts
ServiceX
Awkward Array
Vector
Hist
ATLAS xAOD Data
Different hint files for different workflows…
Different prompts for different tasks
Prompt
Question
Same for all questions
Large Language Model

[Github with all hint files](#)

Awkward Array Hints File

Flattening Nested Arrays

Use `ak.flatten` to reduce nested list structure. By default, `ak.flatten(array)` removes one level of nesting (flattens along the axis=1). Setting `axis=None` will completely flatten an array, erasing all nesting (turning it into 1D):

```
import awkward as ak

array = ak.Array([[0, 1, 2], [], [3, 4], [5, 6, 7]])
flat_level1 = ak.flatten(array, axis=1)          # Flatten one level (axis=1 by default):contentReference[oaicite:2]{index=2}
flat_all    = ak.flatten(array, axis=None)  # Flatten all levels into 1D:contentReference[oaicite:3]{index=3}

print(flat_level1)  # [0, 1, 2, 3, 4, 5, 6, 7]
print(flat_all)     # [0, 1, 2, 3, 4, 5, 6, 7]  (same here because only one level)
```

Notes:

- Always explicitly set the `axis` in the arguments to `flatten`. This will force us to think about what axis we are flattening and if the data has that access.
- If a variable has no array structure or is a 1D array, then `ak.flatten` will cause an error. Be sure your data has the requested `axis` if you are going to flatten.

Combining Multiple Fields into Records

Use `ak.zip` to join parallel arrays into an array of records (structured data). The `ak.zip` function can combine arrays of equal length into one record per entry. For example, zipping an age array and a name array yields an array of records with fields `"age"` and `"name"`:

```
import awkward as ak

ages = ak.Array([18, 32, 41])
names = ak.Array(["Dorit", "Caitlin", "Bryn"])
people = ak.zip({"age": ages, "name": names})   # Combine fields into records:contentReference[oaicite:5]{index=5}
```

ATLAS xAOD Data Model

xAOD Event Data Model Hints

Some hints to help with accessing xAOD objects.

Important note about units:

- All momentum, energy, and mass units are in MeV (e.g. `pt, E, m`). Unless asked otherwise, convert them to GeV by dividing by 1000 as early as possible.
- All distance units are in millimeters. Convert them to meters as early as possible.

Jets, Muons, Taus, Photons, Electrons

It is possible to access all these objects from the top event level, e.g.,

```
query = FuncADLQueryPHYS() \
    .Select(lambda e: e.Jets()) \
    .Select(lambda jets: {
        'pt': jets.Select(lambda j: j.pt()/1000.0),
        'eta': jets.Select(lambda j: j.eta()/1000.0),
    })
```

The only odd one are tau's, which use `e.TauJets("AnalysisTauJets")`.

These objects all have `pt, eta, phi` (with methods by those names). To access `px, py, pz` you have to get the 4-vector first:

```
query = FuncADLQueryPHYS() \
    .Select(lambda e: e.Jets()) \
    .Select(lambda jets: {
        'pt': jets.Select(lambda j: j.p4().px()/1000.0),
    })
```

[Github with all hint files]

Awkward Array Hints File

Flattening Nested Arrays

Use `ak.flatten` to reduce nested list structure. By default, `ak.flatten(array)` removes one level of nesting (flattens along the axis=1). Setting `axis=None` will completely flatten an array, erasing all nesting (turning it into 1D):

```
import awkward as ak

array = ak.Array([[0, 1, 2], [], [3, 4], [5, 6, 7]])
flat_level1 = ak.flatten(array, axis=1)         # Flatten one l
flat_all   = ak.flatten(array, axis=None)  # Flatten all levels

print(flat_level1)  # [0, 1, 2, 3, 4, 5, 6, 7]
print(flat_all)     # [0, 1, 2, 3, 4, 5, 6, 7]  (same here becau
```

Notes:

- Always explicitly set the `axis` in the arguments to `flatten`. This that access.
- If a variable has no array structure or is a 1D array, then `ak.flatte` are going to flatten.

Combining Multiple Fields into Records

Use `ak.zip` to join parallel arrays into an array of records (structured record per entry. For example, zipping an age array and a name array y

```
import awkward as ak

ages = ak.Array([18, 32, 41])
names = ak.Array(["Dorit", "Caitlin", "Bryn"])
people = ak.zip({"age": ages, "name": names})   # Combine fields
```

ATLAS xAOD Data Model

xAOD Event Data Model Hints

Some hints to help with accessing xAOD objects.

hem to GeV by dividing by 1000 as

get the 4-vector first:

Writing Hint Files

- Used a LLM to draft many of the bigger ones first ("deep research")
  - Some evidence these are too wordy
- Wrote some by hand (e.g. xAOD Data Model) – no good public source of documentation
- Iterate – when the LLM gave a confusing answer, could often trace it to incorrect or unclear hint files! [Found some errors in source documentation]

These hint files end up being quite useful for humans – they can be written in a very condensed way!

You will be asked to write python code that will access ATLAS data, extract some items, and then make one or more plots.

If the user's prompt doesn't fit that request, please complain. **Generate** code using ServiceX to fetch the data, Awkward array to **manipulate** the data that comes back (with vector for doing any physics calculations like invariant mass), and Hist to **generate and plot** the histogram. **Write** the histogram/plots to `png` files.

If hints contain instructions to tell the user some bit of information, make sure to do that. ***If you are manipulating from ServiceX, use only Awkward array - do not use python lists****, etc.* Just give the user the code, do ***not try to create a pull request in any repo***.

** Hints **

{hints}

** User Question **

{question}

** Final Instructions **

Please reply in the following format:

>>start-reply<<

<text describing what you have done as a bulleted list>

` ` ` python

<code>

` ` `

>>end-reply<<

Added to clean up the output!

Modern LLM's are fine-tuned to explain – if you don't give them an outlet, they will just write it anyway!

All you actually want...

Prompt

"Simple" prompt in github

Question 1: *Plot the ETmiss of all events*

- Used ServiceX FuncADL (PHYSLITE) to fetch only the event MissingET magnitude (converted to GeV) to minimize data movement.
- Queried the specified rucio dataset and delivered NFiles=1 (kept small for a quick example; you can increase NFiles yourself when ready).
- Converted the ServiceX result to an Awkward Array and filled a Hist histogram (50 bins, 0–200 GeV as a reasonable MET range).
- Plotted with mplhep ATLAS style and saved to etmiss.png.

(notes from LLM response)

```
import awkward as ak
from servicex import deliver, ServiceXSpec, Sample, dataset
from servicex_analysis_utils import to_awk
from func_adl_servicex_xaodr25 import FuncADLQueryPHYSLITE

from hist import Hist
import matplotlib.pyplot as plt
import mplhep as hep

# Use ATLAS plotting style
plt.style.use(hep.style.ATLAS)

# Build the ServiceX query: get MissingET (first object), then select its magnitude in GeV
base_query = FuncADLQueryPHYSLITE()
met_query = (
    base_query
    .Select(lambda e: e.MissingET().First())
    .Select(lambda m: {"met": m.met() / 1000.0})  # Convert to GeV
)

# Dataset name (from user)
ds_name = "mc23_13p6TeV:mc23_13p6TeV.801167.Py8EG_A14NNPDF23LO_jj_JZ2.deriv.DAOD_PHYSLITE.e8514_e8528_a911_s4114_r15224_r15225_

# Deliver the data (use NFiles=1 for a quick test run)
results = to_awk(
    deliver(
        ServiceXSpec(
            Sample=[
                Sample(
                    Name="etmiss_fetch",
                    Dataset=dataset.Rucio(ds_name),
                    NFiles=1,
                    Query=met_query
```

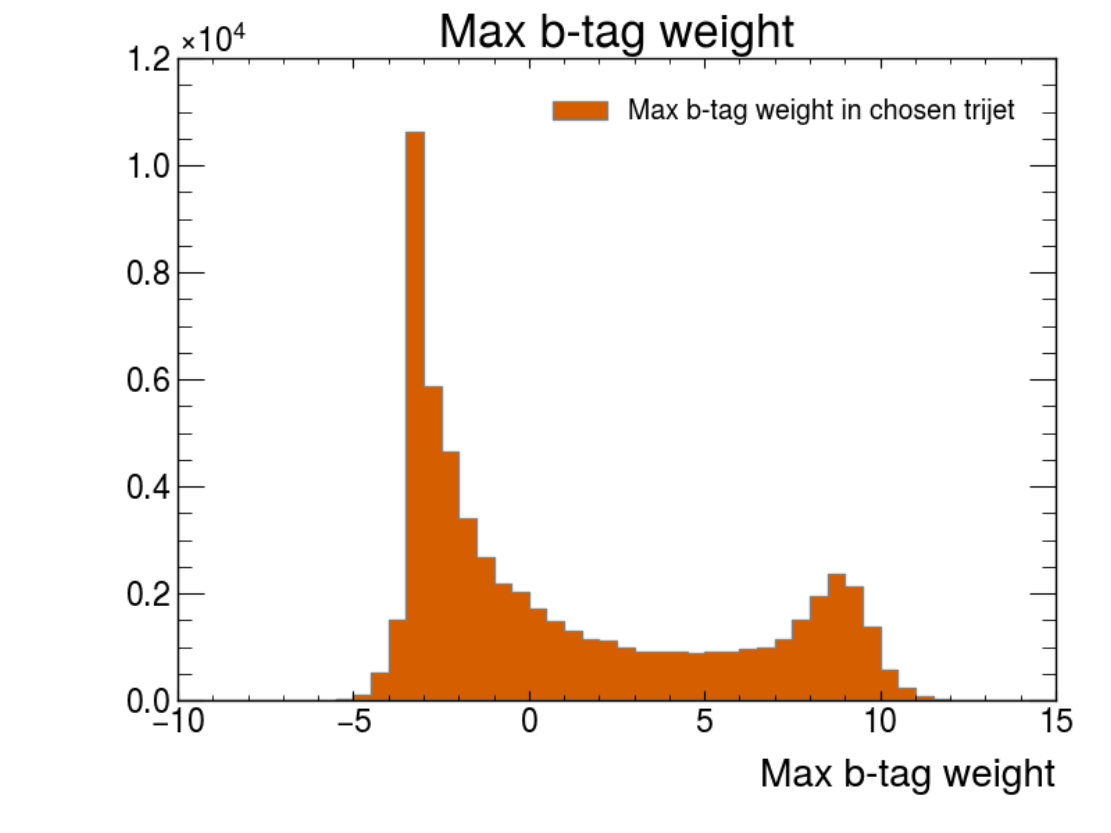
s / adl-benchmarks / results / direct-query / **direct-question-01.md**

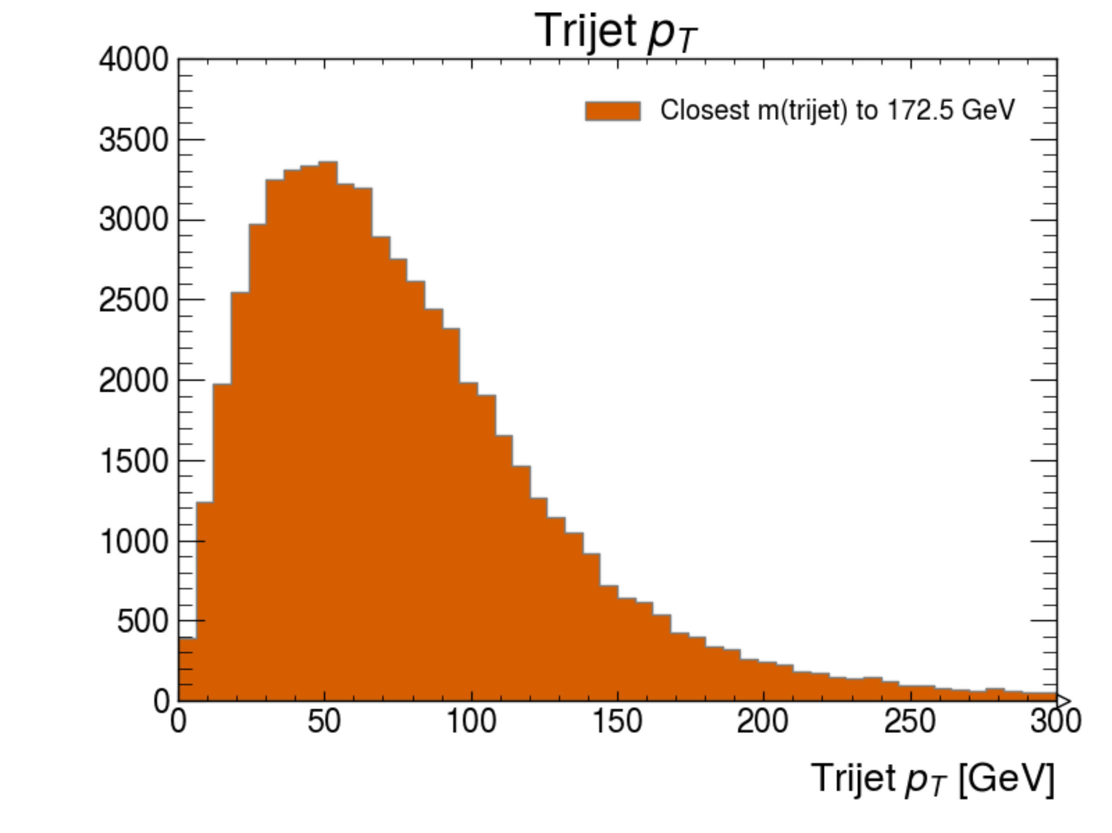
loc) · 135 KB

```
)

# Extract the MET array (Awkward)
met = results["etmiss_fetch"].met  # units: GeV

# Build histogram: 50 bins from 0 to 200 GeV
h = (
    Hist.new
        .Reg(50, 0.0, 200.0, name="met", label=r"$E_T^{\mathrm{miss}}$ [GeV]")
        .Int64()
)

# Fill histogram (Hist accepts array-like; Awkward array works or can use ak.to_numpy(met))
h.fill(met=met)

# Plot and save
fig, ax = plt.subplots()
h.plot(histtype="fill", linewidth=1, edgecolor="gray", label=r"All events")
ax.legend()
ax.set_xlabel(r"$E_T^{\mathrm{miss}}$ [GeV]")
ax.set_title("Missing $E_T$")
fig.savefig("etmiss.png")
plt.close(fig)
```

Question 6: For events with at least three jets, plot the pT of the trijet four-momentum that has the invariant mass closest to 172.5 GeV in each event and plot the maximum b-tagging discriminant value among the jets in this trijet

- Uses ServiceX (FuncADL R25 PHYSLITE) to fetch, in one delivery (NFiles=1), only what's needed: per-event jet kinematics (pt, eta, phi, mass in GeV) and the b-tagging discriminant (**GN2v01** tagger weight) via the BTaggingSelectionTool (FixedCutBEff_77 operating point).
- Filters events at the ServiceX level to require at least three jets, reducing data moved.
- Builds Awkward Momentum4D vectors, forms all trijet combinations per event, selects the combination with invariant mass closest to 172.5 GeV, then:
  - Plots the pT of the selected trijet four-momentum.
  - Plots the maximum b-tagging discriminant among the three jets in the selected trijet.
- Uses Awkward + Vector for all physics calculations and Hist for histogramming; saves plots to PNG files.
- Dataset: mc23_13p6TeV:mc23_13p6TeV.601237.PhPy8EG_A14_ttbar_hdamp258p75_allhad.deriv.DAOD_PHYSLITE.e8514_s4369_r16083_p6697.
- Note: Only one file is processed (NFiles=1) here; when you are ready, you can scale up yourself.


Max b-tag weight
×10^4
1.2
1.0
0.8
0.6
0.4
0.2
0.0
Max b-tag weight in chosen trijet
−10
−5
0
5
10
15
Max b-tag weight


Trijet p_T
4000
3500
3000
2500
2000
1500
1000
500
0
Closest m(trijet) to 172.5 GeV
0
50
100
150
200
250
300
Trijet p_T [GeV]

Got it!

What about errors?

stdout:

```
etmiss: Transform ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0/0 --:--
         Download ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 0/0 --:--
```

stderr:

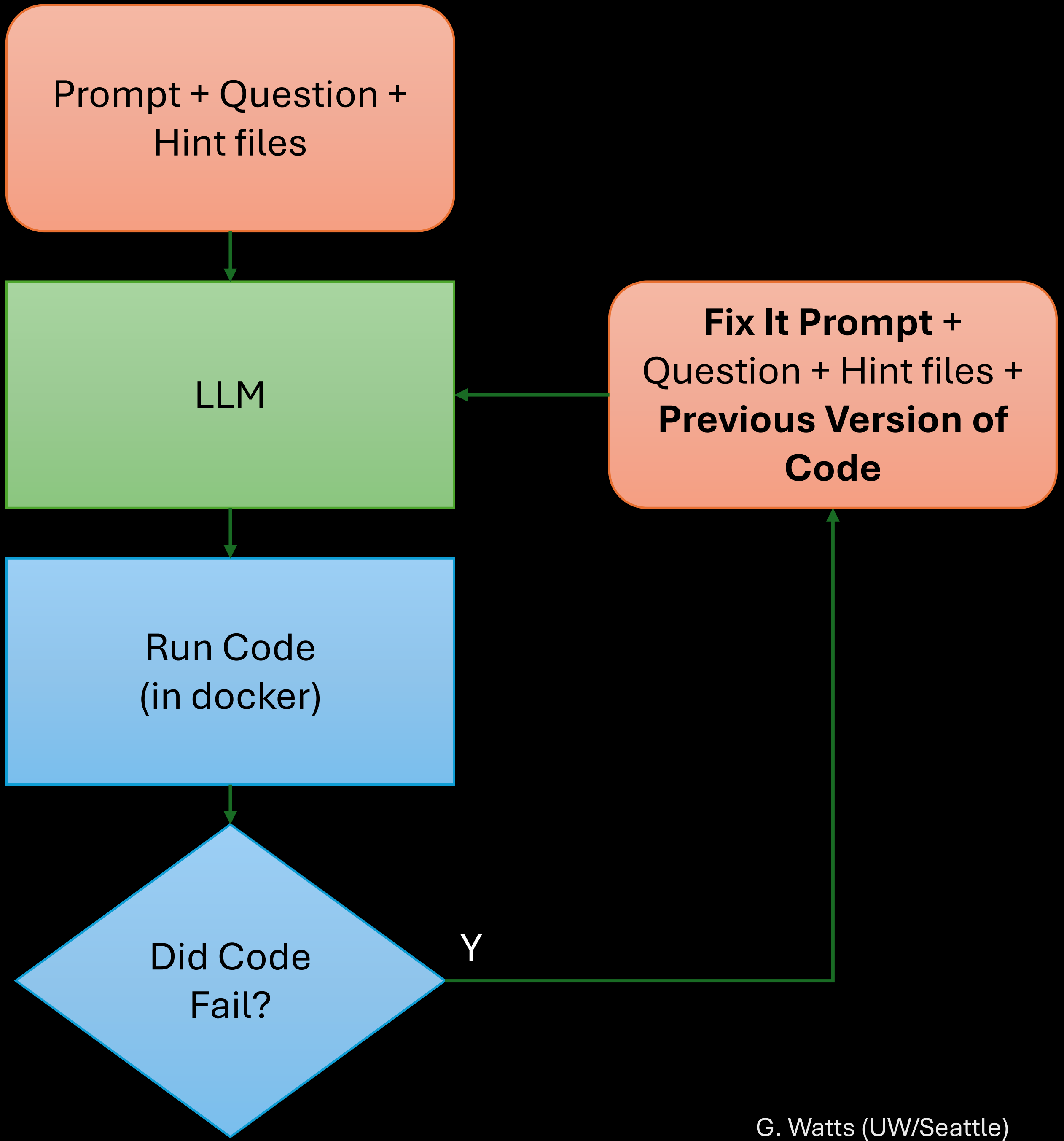
```
bash: cannot set terminal process group (-1): Inappropriate ioctl for device
bash: no job control in this shell
/work/.venv/lib/python3.13/site-packages/func_adl_servicex_xaodr25/trigger.py:127: SyntaxWarning: invalid escape sequence '\D'
  object. Close match is done as a function of $\Delta R < `dr`$.
Traceback (most recent call last):
  File "/app/script.py", line 36, in <module>
    result = to_awk(
        deliver(
    ...<10 lines>...
        )
    )
  File "/work/.venv/lib/python3.13/site-packages/servicex_analysis_utils/materialization.py", line 57, in to_awk
    raise ValueError(f"ServiceX result path list for {sample} cannot be empty.")
ValueError: ServiceX result path list for etmiss cannot be empty.
```

This is a crash in ServiceX

- No data was returned
- Note the “0/0” – no files were found for ServiceX to process.
- This means that the dataset name is likely bad.

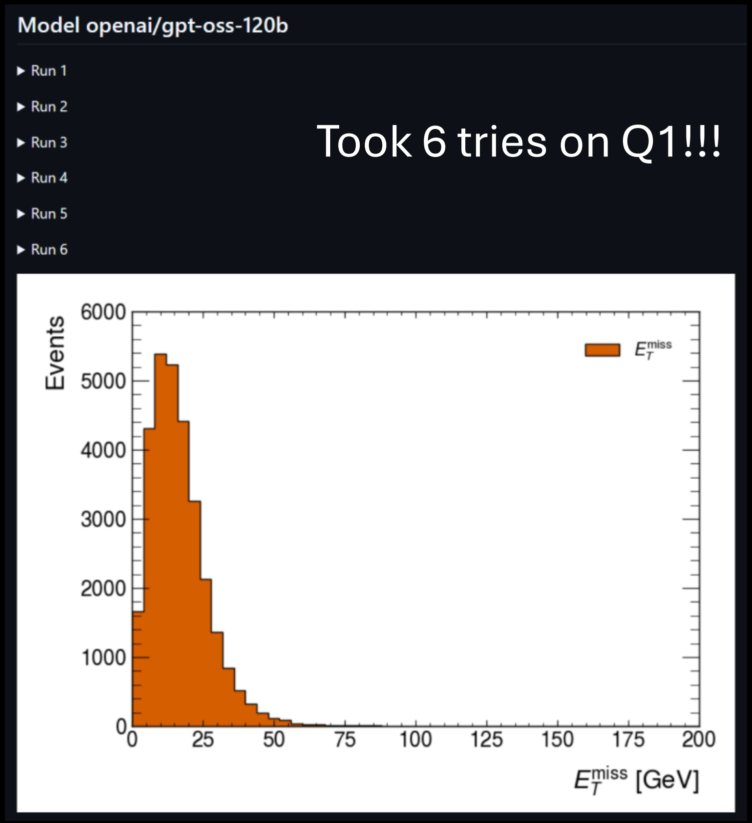
BUG:

- Note there is no informative error message…
- I had to use my expert knowledge of the system to realize what was going on.
- **Just add this to the hint file and now the LLM is also an expert!**



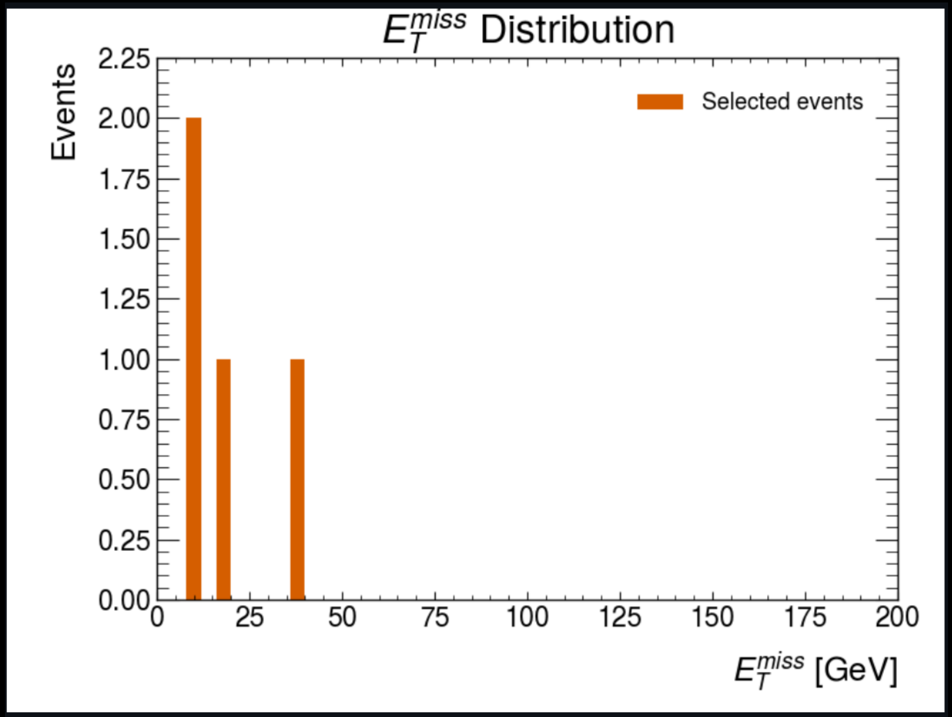
- Fix It Prompt is very similar to first prompt
  - Includes old code
  - stderr, stdout
- Iterations are allowed up to some limit

In general, smaller models have more trouble than larger models!

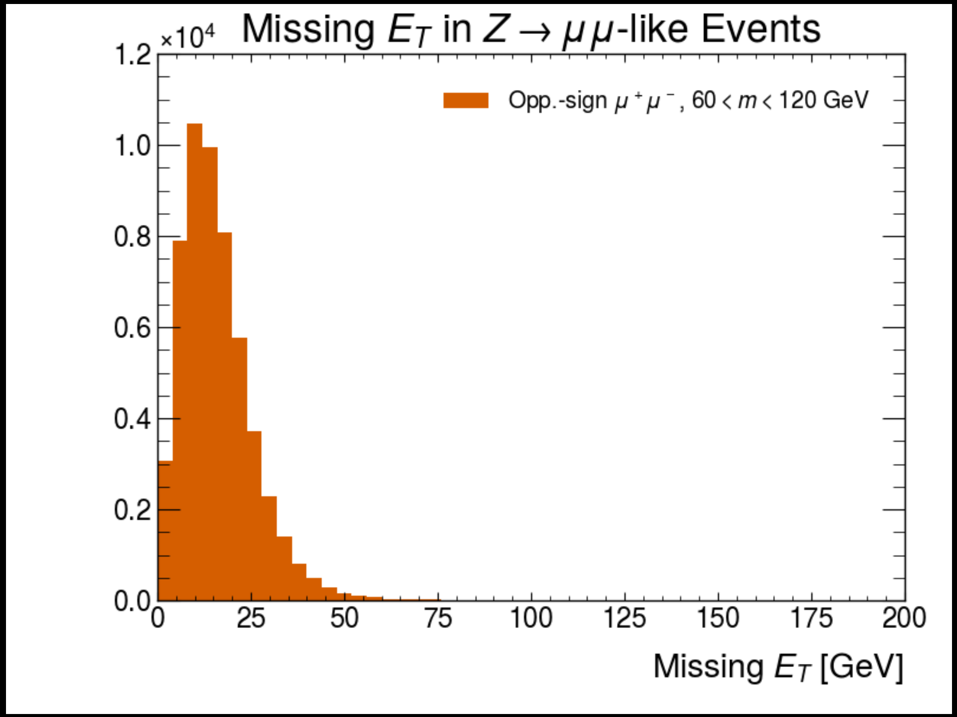


And “Hallucinations”…

gpt-4o Question 5

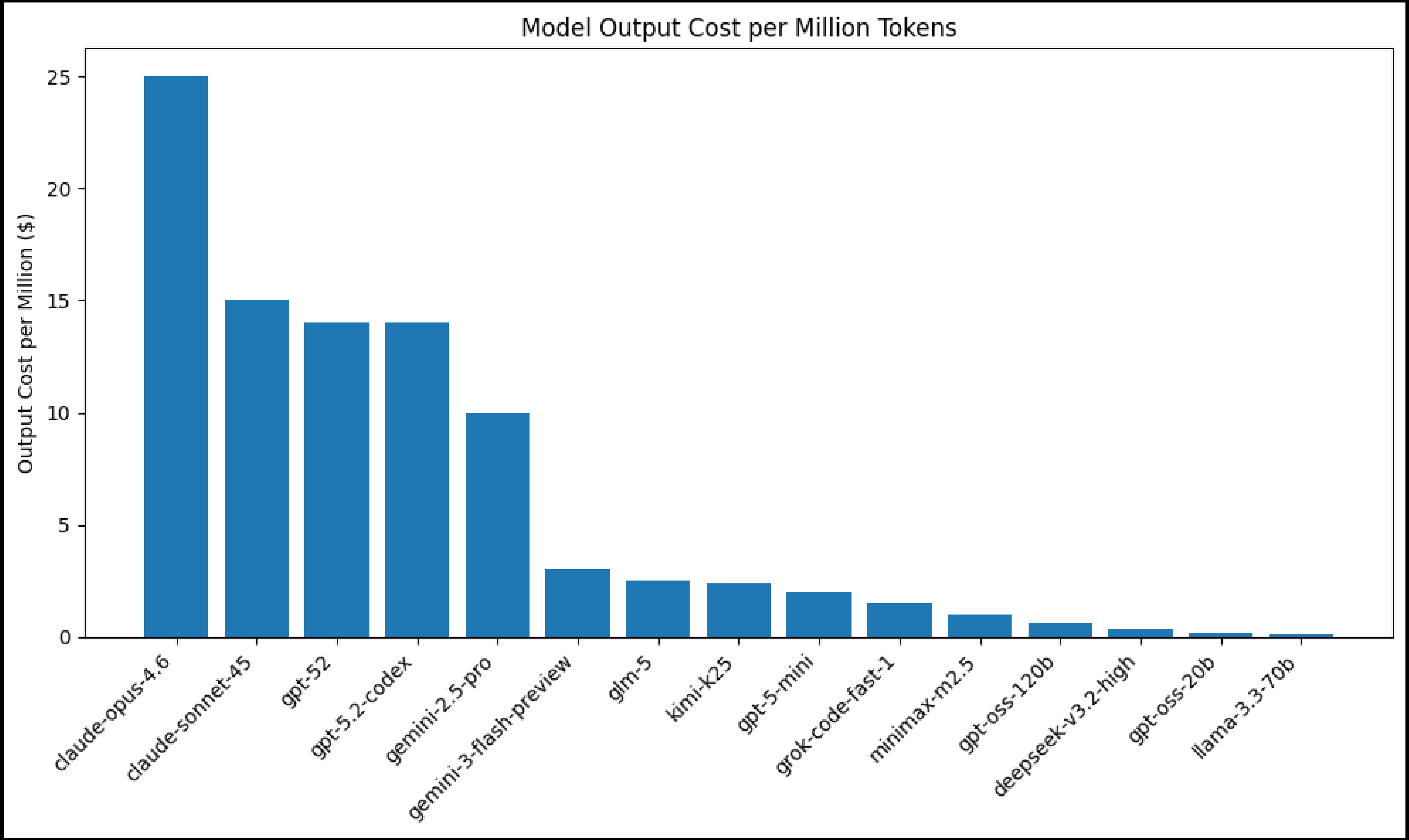


Correct Question 5 Answer



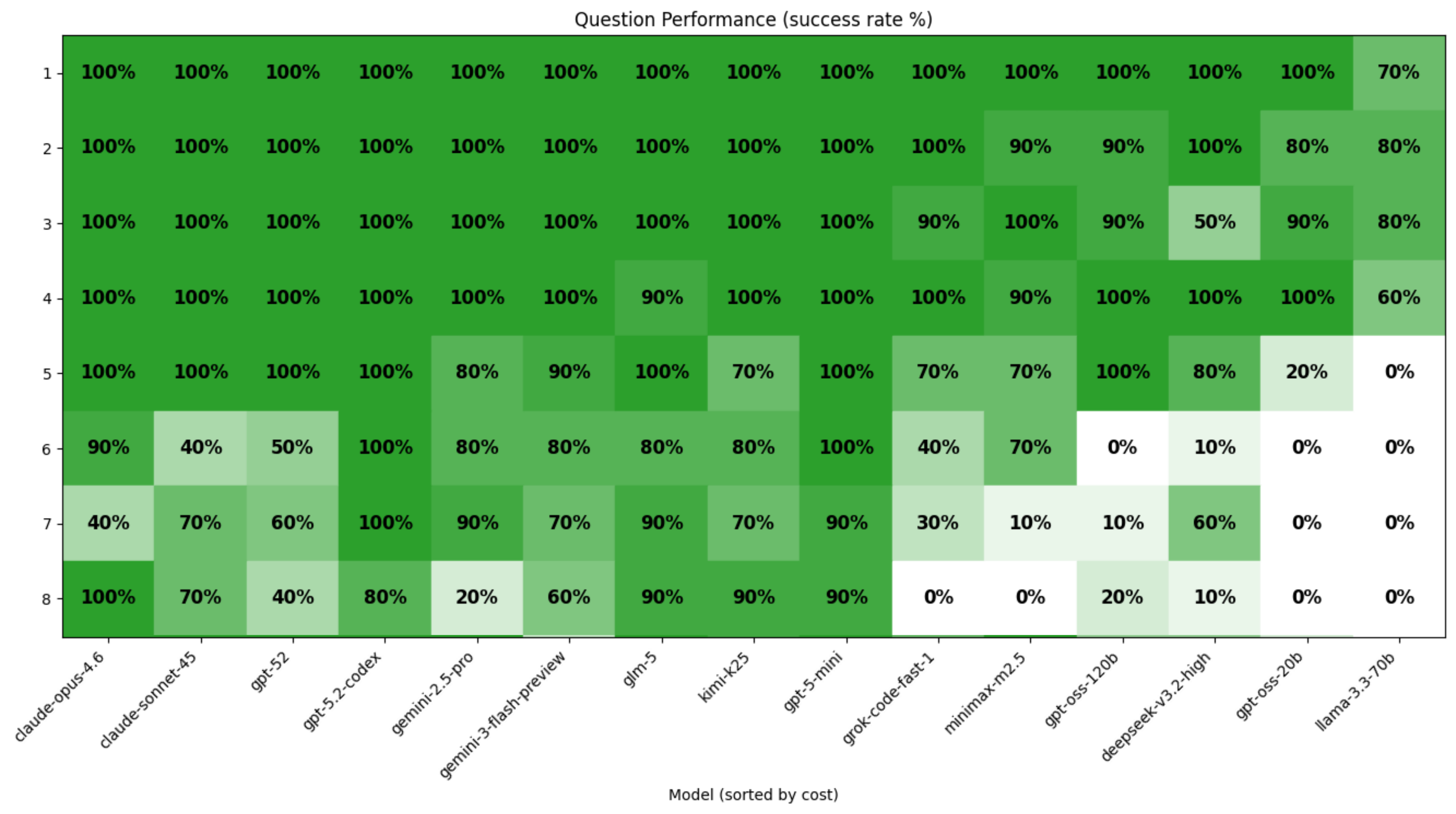
All models answers to question 5

Cost (USD)

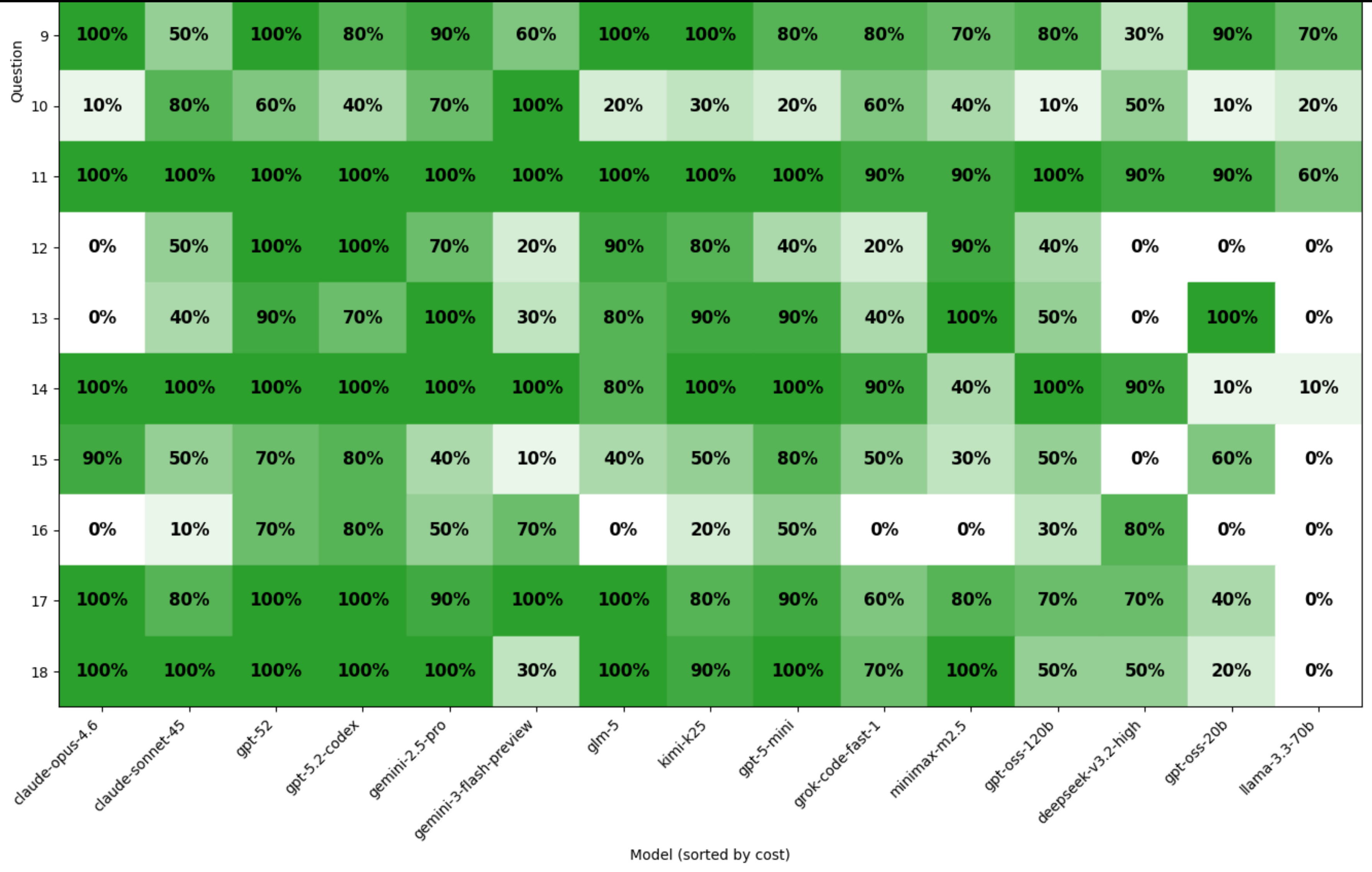


Model Output Cost per Million Tokens
Output Cost per Million ($)
0
5
10
15
20
25
claude-opus-4.6
claude-sonnet-45
gpt-52
gpt-5.2-codex
gemini-2.5-pro
gemini-3-flash-preview
glm-5
kimi-k25
gpt-5-mini
grok-code-fast-1
minimax-m2.5
gpt-oss-120b
deepseek-v3.2-high
gpt-oss-20b
llama-3.3-70b

10 Trials

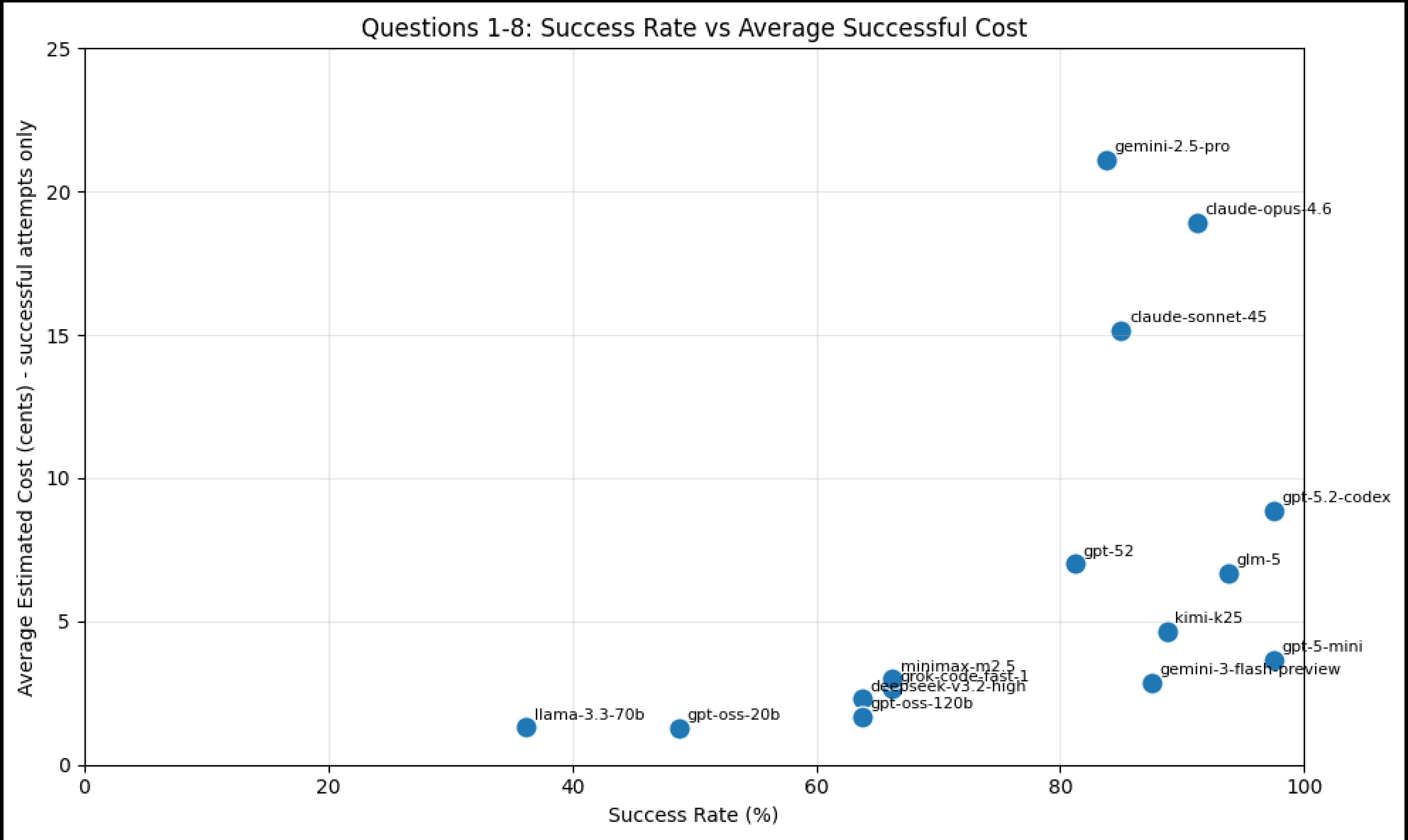

Question Performance (success rate %)
1 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 70%
2 100% 100% 100% 100% 100% 100% 100% 100% 100% 100% 90% 90% 100% 80% 80%
3 100% 100% 100% 100% 100% 100% 100% 100% 100% 90% 100% 90% 50% 90% 80%
4 100% 100% 100% 100% 100% 100% 90% 100% 100% 100% 90% 100% 100% 100% 60%
5 100% 100% 100% 100% 80% 90% 100% 70% 100% 70% 70% 100% 80% 20% 0%
6 90% 40% 50% 100% 80% 80% 80% 80% 100% 40% 70% 0% 10% 0% 0%
7 40% 70% 60% 100% 90% 70% 90% 70% 90% 30% 10% 10% 60% 0% 0%
8 100% 70% 40% 80% 20% 60% 90% 90% 90% 0% 0% 20% 10% 0% 0%
claude-opus-4.6
claude-sonnet-45
gpt-52
gpt-5.2-codex
gemini-2.5-pro
gemini-3-flash-preview
glm-5
kimi-k25
gpt-5-mini
grok-code-fast-1
minimax-m2.5
gpt-oss-120b
deepseek-v3.2-high
gpt-oss-20b
llama-3.3-70b
Model (sorted by cost)

10 Trials

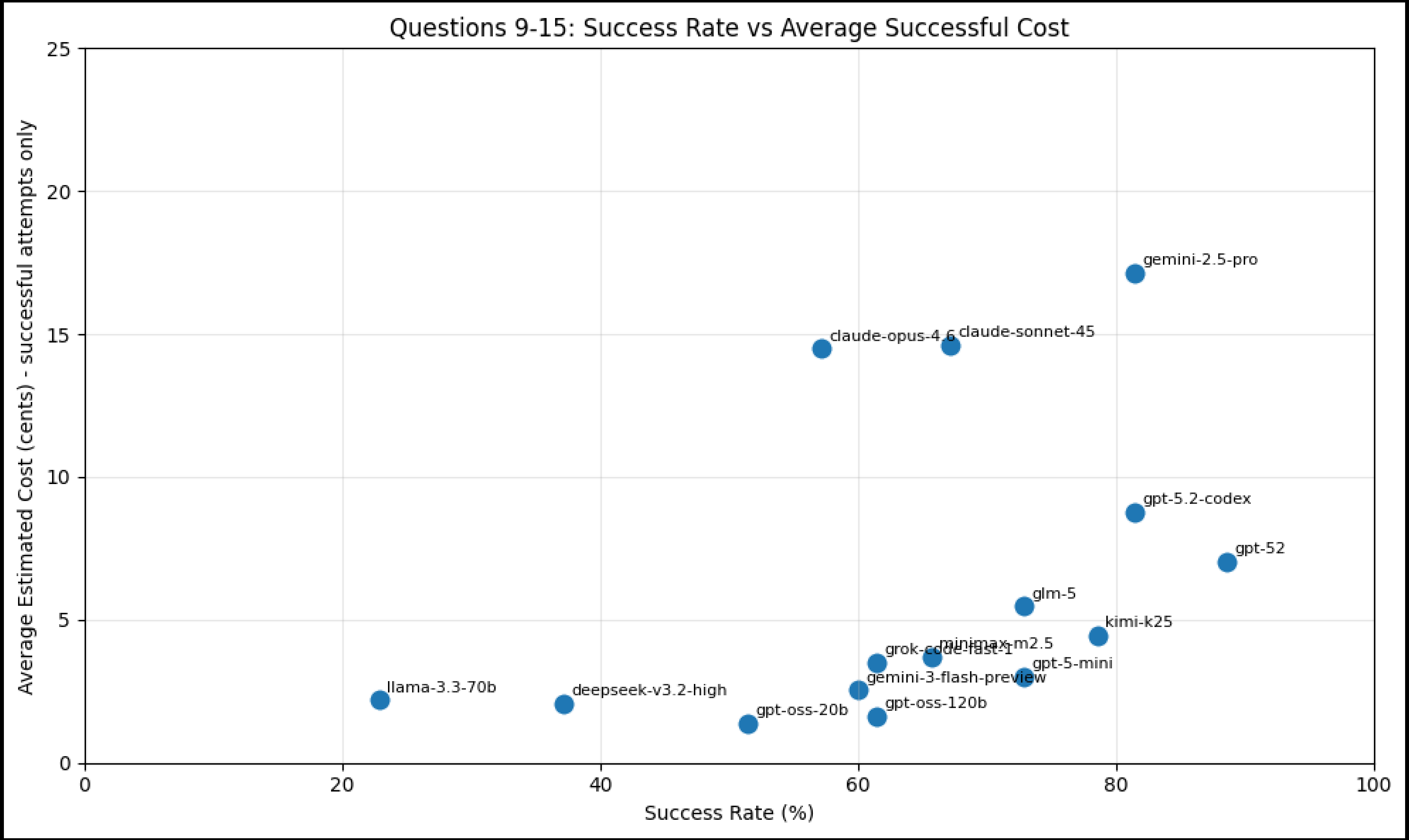


Correctly calculated with # of input and output tokens (as reported by the API)

Cost


Questions 1-8: Success Rate vs Average Successful Cost
Average Estimated Cost (cents) - successful attempts only
Success Rate (%)
0
5
10
15
20
25
0
20
40
60
80
100
gemini-2.5-pro
claude-opus-4.6
claude-sonnet-45
gpt-5.2-codex
gpt-52
glm-5
kimi-k25
gpt-5-mini
gemini-3-flash-preview
minimax-m2.5
grok-code-fast-1
deepseek-v3.2-high
gpt-oss-120b
llama-3.3-70b
gpt-oss-20b

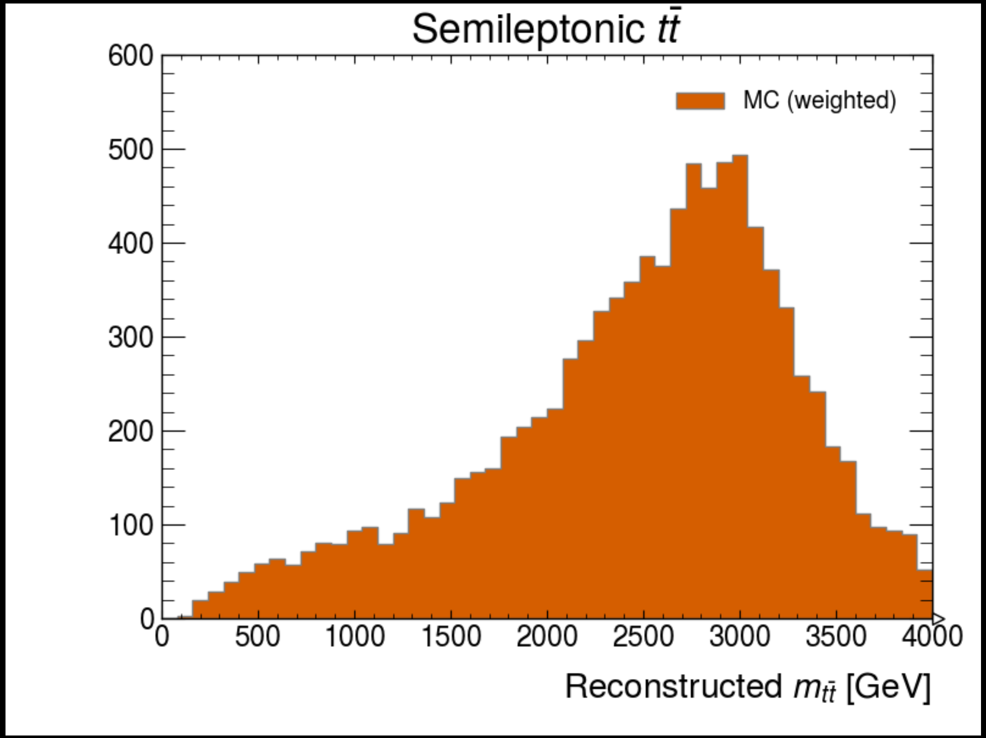
Cost



Questions 9-15: Success Rate vs Average Successful Cost
Average Estimated Cost (cents) - successful attempts only
Success Rate (%)
0
5
10
15
20
25
0
20
40
60
80
100
gemini-2.5-pro
claude-opus-4.6
claude-sonnet-45
gpt-5.2-codex
gpt-52
glm-5
kimi-k25
minimax-m2.5
grok-code-fast-1
gpt-5-mini
gemini-3-flash-preview
llama-3.3-70b
deepseek-v3.2-high
gpt-oss-20b
gpt-oss-120b

3 TeV Z-Prime

Plot the reconstructed top quark pair mass in events with one charged lepton in the rucio dataset <z-prime-mc-3000-gev>



- "Not unreasonable" – after examining the code
- Could have used b-tagging to improve further
- Used m_W mass constraint to choose the smaller neutrino p_z
- GPT-5 made two coding errors
  - That it had never made in other questions
  - But many other models had
  - Getting to the limits of what it can do in one shot?
- BUT… injects preconceptions
  - Initial Object Selection Cuts, etc.
  - So plots aren't identical to reference
  - But still is getting "physics" right
  - Reproducible? Not really!

What Dataset Should We Use?

The actual question #6 fed to our models:

> **For events with at least three jets, plot the pT of the trijet four-momentum that has the invariant mass closest to 172.5 GeV in each event and plot the maximum b-tagging discriminant value among the jets in this trijet** ***in the rucio dataset mc23_13p6TeV:mc23_13p6TeV.601237.PhPy8EG_A14_ttbar_hdamp258p75_allhad.deriv.DAOD_PHYSLITE.e8514_s4369_r16083_p6697.***

Why can't we say "in semileptonic ttbar"?

- Every experiment has a wealth of metadata associated with its datasets
- Twiki pages often maintain which samples to use for what
- Imagine sorting between a Run 2 and Run 3 analysis!
- Or full sim and fast sim!
- Cross sections and MC production efficiencies!

That said:

- There are rules!
- Even experiment scripts…

How hard is it to write a tool that the llm can use? (not that hard…)

What Might Be Next?

What should we be writing?

- Tools (mcp-based)
- Prompts that can be used to accomplish goals

We should **shy away from writing big complex integrated systems** except as the final step!

Tools, prompts, hint files are reusable by many system!

- Even humans can use them
- The plug into existing IDE's and could help a user in real time!

We need a HEP repository of tools, served to the web in general

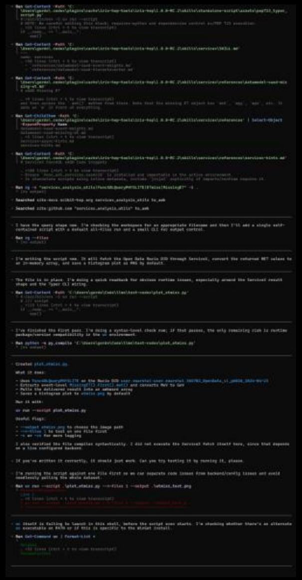
- Tricky: want authentication so a user can use these in their experiment's privileged context!

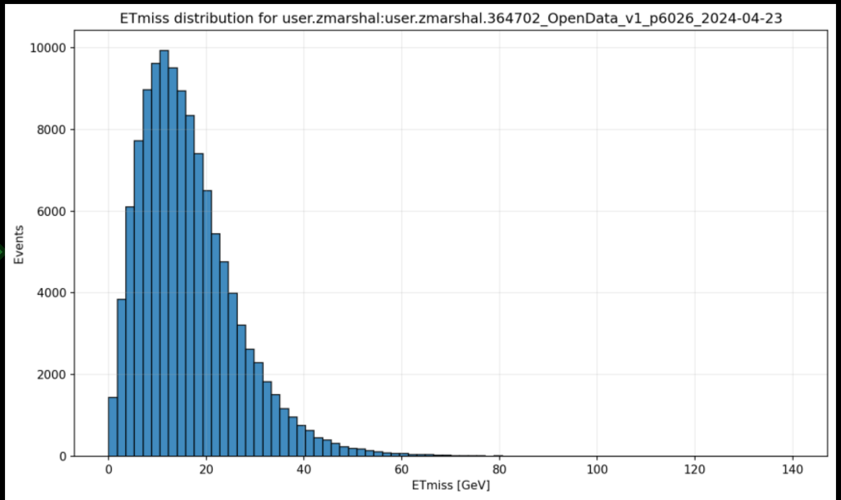
Build lego bricks, not buildings!

Write Our Own Query Framework?

Install plugins in codex/claude/copilot
(tested in codex/claude)
Plugin for coding with SX/AWK, etc.
(thanks to Ben G!!!)

```
>_ OpenAI Codex (v0.125.0)

model:     gpt-5.4-mini medium   /model to change
directory: ~\Code\llms\test-codex

Tip: New Use /fast to enable our fastest inference with increased plan usage.

› $iris-hep Please write a stand-alone python file that we can use uv to run (and auto install) that will do the
  following: "Plot the ETmiss of all events in the rucio dataset
  user.zmarshal:user.zmarshal.364702_OpenData_v1_p6026_2024-04-23."
```






```python
#!/usr/bin/env -S uv run --script
# /// script
# requires-python = ">=3.13"
# dependencies = [
#   "typer>=0.12",
#   "matplotlib>=3.9",
#   "awkward>=2.6",
#   "servicex>=2.8",
#   "servicex-analysis-utils>=1.2",
#   "func-adl-servicex-xaodr25>=0.6",
#   "jinja2>=3.1",
# ]
# ///
from __future__ import annotations

import logging
from pathlib import Path

import awkward as ak
import matplotlib

matplotlib.use("Agg")
import matplotlib.pyplot as plt
import typer
from func_adl_servicex_xaodr25 import FuncADLQueryPHYSLITE
from servicex import Sample, ServiceXSpec, dataset, deliver
from servicex_analysis_utils import to_awk

DEFAULT_DATASET = "user.zmarshal:user.zmarshal.364702_OpenData_v1_p6026_2024-04-23"
DEFAULT_OUTPUT = Path("etmiss.png")

app = typer.Typer(add_completion=False)


def _build_query() -> object:
    # MissingET is stored as a sequence; take the first element and convert MeV -> GeV.
    return (
        FuncADLQueryPHYSLITE()
        .Select(lambda evt: {"met": evt.MissingET().First()})
        .Select(lambda cols: {"etmiss": cols.met.met() / 1000.0})
    )
```

$ uv run --script plot_etmiss.py

NO

Move to using CLI’s and other frameworks made by the Open Source Community

Next Steps

One Person Efforts

- Very easy to get started, but…
- Proof this can be useful is done (and published)

Work at Scale?

- Now – move away from smaller contrived demonstrations
- Use 100 TB datasets, modern analysis facilities, at scale questions and analysis

Complexity Of Full Analysis?

- Manage context over full range of experiment specific tools and facilities
- Manage context and intent over weeks of work
- Cost?

Build An Ecosystem…

- Of Skills
  - How to perform things like statistical anslysis
  - How to extract data (ntuple)
  - Top-jet boosted tagging…
- Of Tools
  - Experiment meta-data (x-section)
  - Datasets, reprocessing versions, etc.
- Of Context
  - How far can it go on its own towards an analysis?
  - How to integrated a human?

Specific “Next” Goals

OpenData: Run the AGC

1. Develop the skills, tools, git repo structure (e.g. maintaining context) so they are generic enough the AGC prompt has to specify the analysis
2. Run 10 times. Cost? How different are the answers?

ATLAS: Repeat a simple non-standard final state analysis

1. Similar as to above, but unlike a canned analysis, target Run 3

Conclusions

- Bigger Models Are Better
  - Not worth wasting time with the smaller ones?
  - Certainly nothing under 100B
  - Cost is *now* an issue!
- Hint Files Are Universal
  - We will need these for whatever we do next
  - Helpful for humans as cheat sheets
  - Optimize hint files for brevity (big models know a lot!)
  - Tools for hint files (mcp)?
- This used ServiceX, Awkward, Hist as its tech stack
  - No reason another stack couldn't work!
  - Mix/Match hint files, replace prompt
  - GRID data being available in minutes… seems like a requirement for "chat"…
- Biggest Issues
  - Validation
  - How do we fit it into our current workflows?
  - What new workflows does it enable?
  - Is this just another framework experts are already writing?
  - Can we figure out how to make "plans" work?
- Next Steps
  - Find small tasks that experiments will find immediately useful.
  - Something that want to do, but is just too much annoying trouble with the current software stack
  - Web site to demonstrate new tools is possible hack-a-thon next step