A python-based analysis pipeline for the ATLAS Integration Challenge

Artur Cordeiro Oudot Choi[1], Alexander Held[2]

[1] University of Washington

[2] University of Wisconsin-Madison

pyhep.dev 2026

https://indico.nikhef.nl/event/7873/

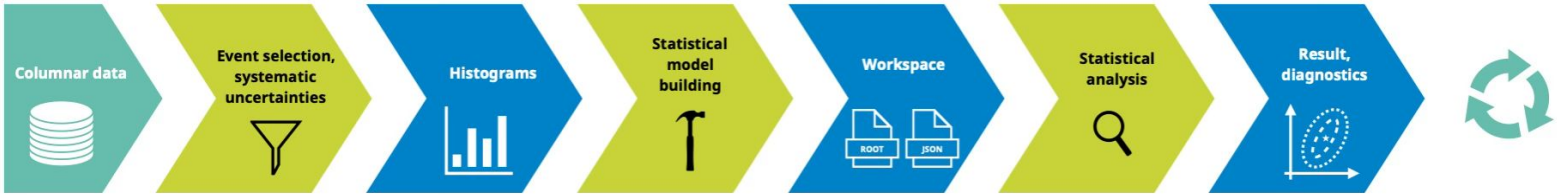
Sep 9, 2026

This work was supported by the U.S. National Science Foundation (NSF) under Cooperative Agreements OAC-1836650 and PHY-2323298.

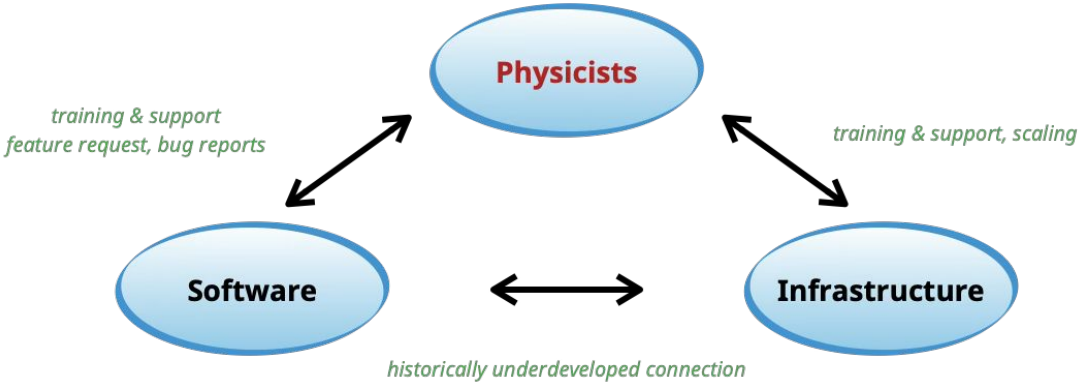
iris hep

Overall idea

- Most **software & computing** work in HEP ultimately has a shared goal: **maximize physics insights**
  - Define projects like the Analysis Grand Challenge (AGC) to test software & infrastructure end-to-end

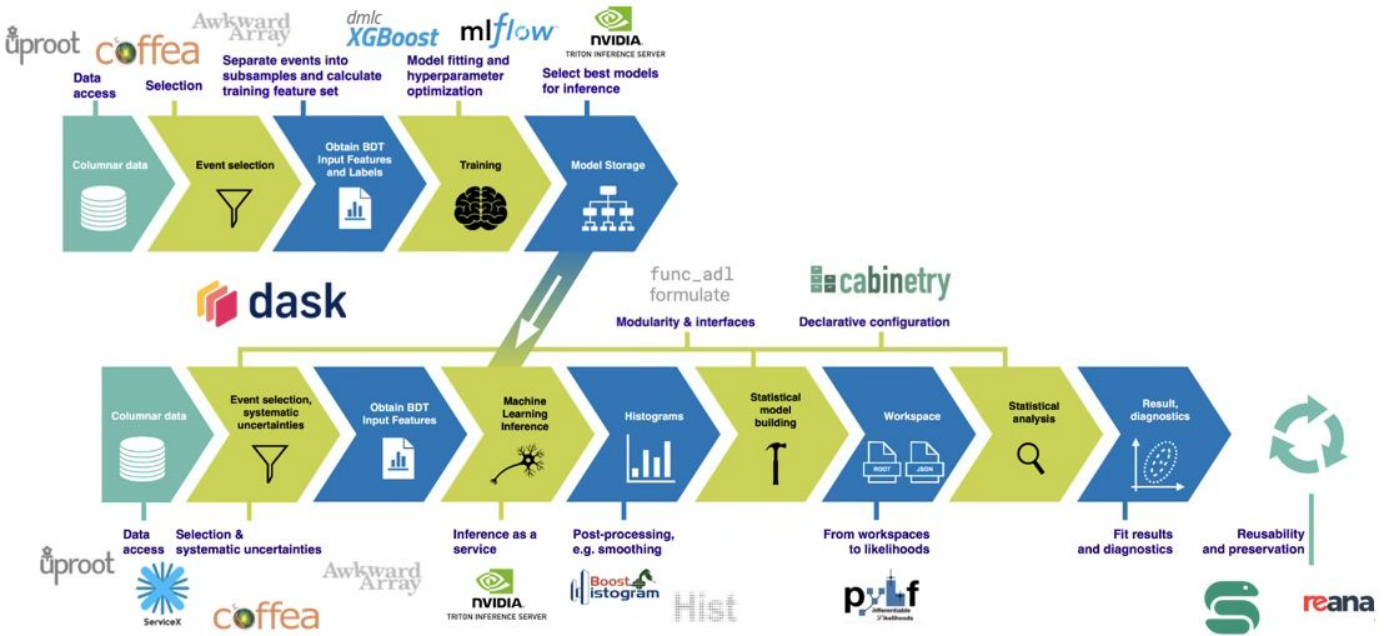


- Bring together different communities to define scalable solutions for **HL-LHC physics analysis** needs



The Analysis Grand Challenge

- **Analysis Grand Challenge** (AGC) demonstrates an analysis workflow from modestly sized Open Data (~ 2 TB)
  - Columnar data processing (`uproot` / `awkward` / `coffea`), histogramming (`hist`), statistical inference (`cabinetry` / `pyhf`), etc.
  - Multiple other implementations exist using different tech stacks



Scaling up

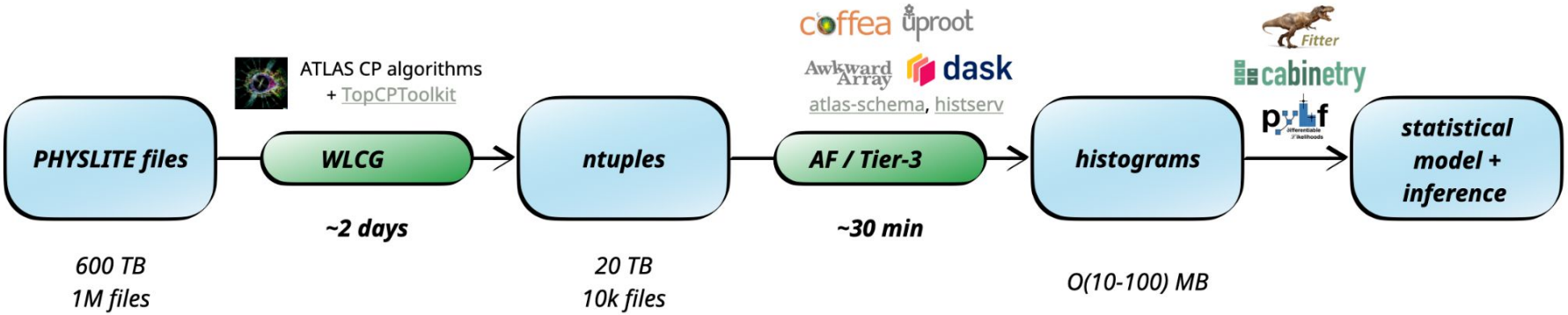
- After a throughput-focused **“200 Gbps Challenge”**, we wanted to define an **updated and expanded** AGC-like project

- **Requirements**
  - Demonstrate scale: as large as possible (to prepare for huge HL-LHC data sizes)
  - Capture full experiment-specific analysis complexity (systematic uncertainties, metadata, …)
    - Define a CMS-flavored and an ATLAS-flavored version
  - Provide ready-to-use blueprint for users, but also retain flexibility as R&D platform to test new ideas in realistic environment

- This resulted in the creation of the (two) “**Integration Challenge(s)**” (https://github.com/iris-hep/integration-challenge)
  - (The name was meant as a working title but stuck around…)

The ATLAS Integration Challenge

- Designed as a **search for charged Higgs bosons** in **ttbar phase space**
- **Two-stage workflow**
  - Produce intermediate ntuples (= tables of columnar event data) with systematic variations evaluated (infrequent)
  - Create histograms & statistical models on local Analysis Facility (day-to-day analysis work)
    - This part is heavily based on Scikit-HEP tooling



Data access: convenient interface

- atlas-schema provides a coffea schema to understand ATLAS CP-algorithms-style output
  - Clean and simple access to systematic variations
  - Extended coffea-style pre-processing to gather more relevant data [coffea#1401]

input file content example:

```
jet_pt_NOSYS
jet_pt_JET_JER_EffectiveNP_1__1up
[…]
jet_eta
jet_phi
jet_e_NOSYS
jet_e_JET_JER_EffectiveNP_1__1up
[…]
```

```
events = NanoEventsFactory.from_root(...).events()
h = hist.new.Regular(...).Weight()

sumw = events.metadata["sumw_dataset"]

for variation in events.systematic_names:

    cut = events[variation]["pass"]["ejets"] == 1
    mjj = (events[variation][cut==1].jet[:, 0] + events[variation][cut==1].jet[:, 1]).mass

    weight = events[variation][cut==1].weight.mc * ... / sumw
    h.fill(mjj, variation=variation, weight=weight)
```

from histogram during preprocessing

understands existing variations

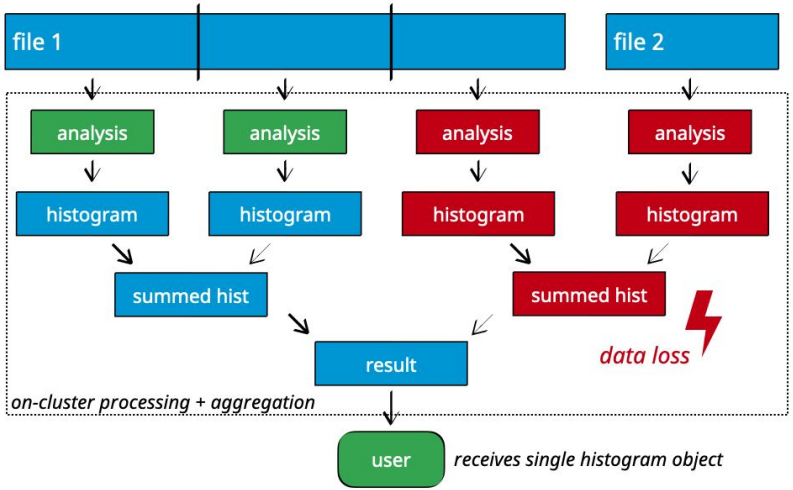
combines correct branches for access to systematically shifted objects

four-vector behavior attached

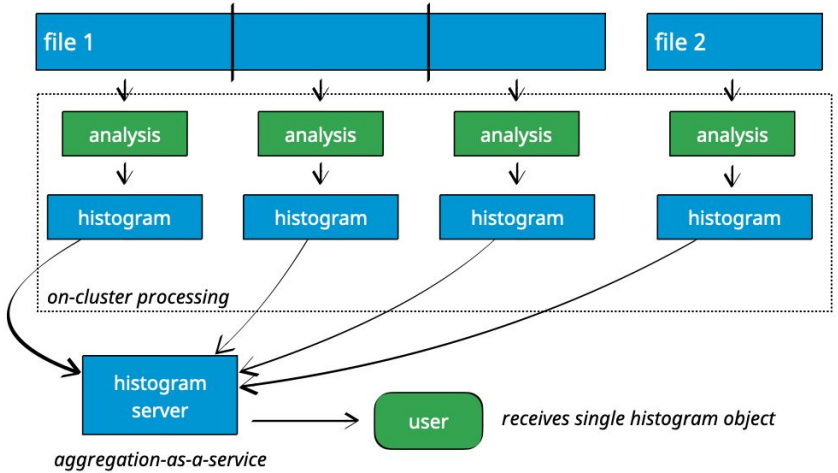
Processing patterns

- Traditional Dask approach is **map + tree reduction**, convenient but fragile when data is lost at scale (O(10k)+ tasks)
- Adopted new solution based on histserv, "**histogramming-as-a-service**": central server receives & combines histograms
  - Significant stability gains, Dask distribution is simple map and failed task can be re-run individually as needed

Traditional map + tree reduction

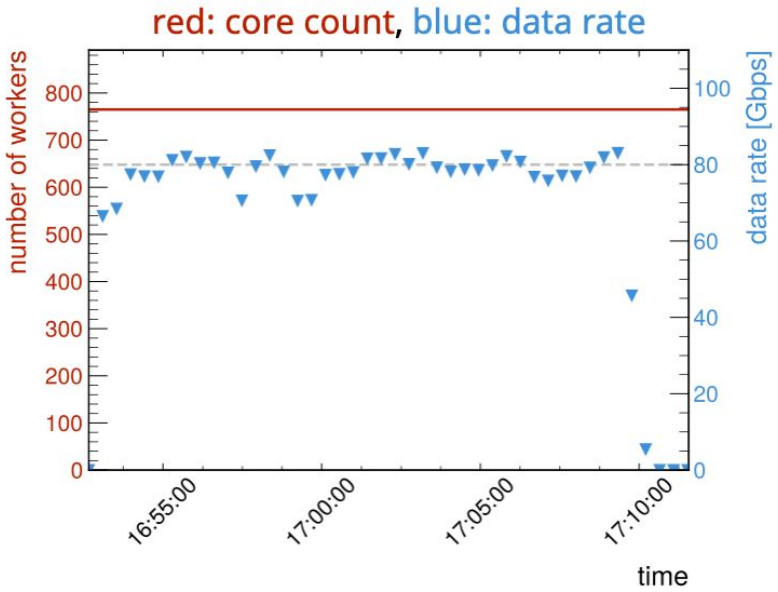


Histogramming-as-a-service

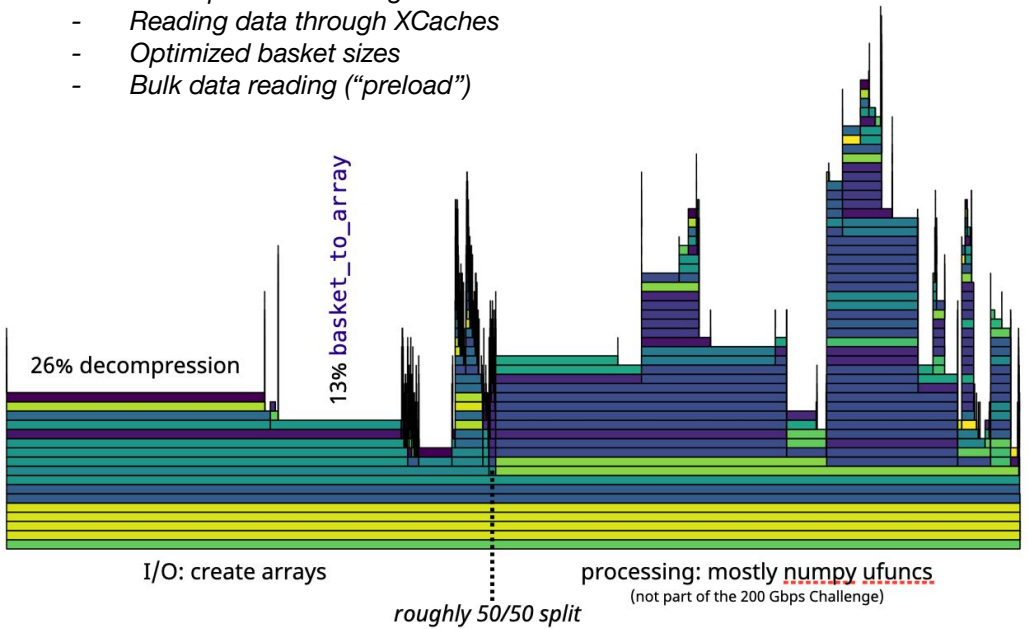


Some performance results

- Performance is good: **20-minute turnaround** for Run-2 + Run-3 ATLAS data analysis
  - Physics analysis at 900 kHz event rate / 80 Gbps data rate at UChicago production Analysis Facility

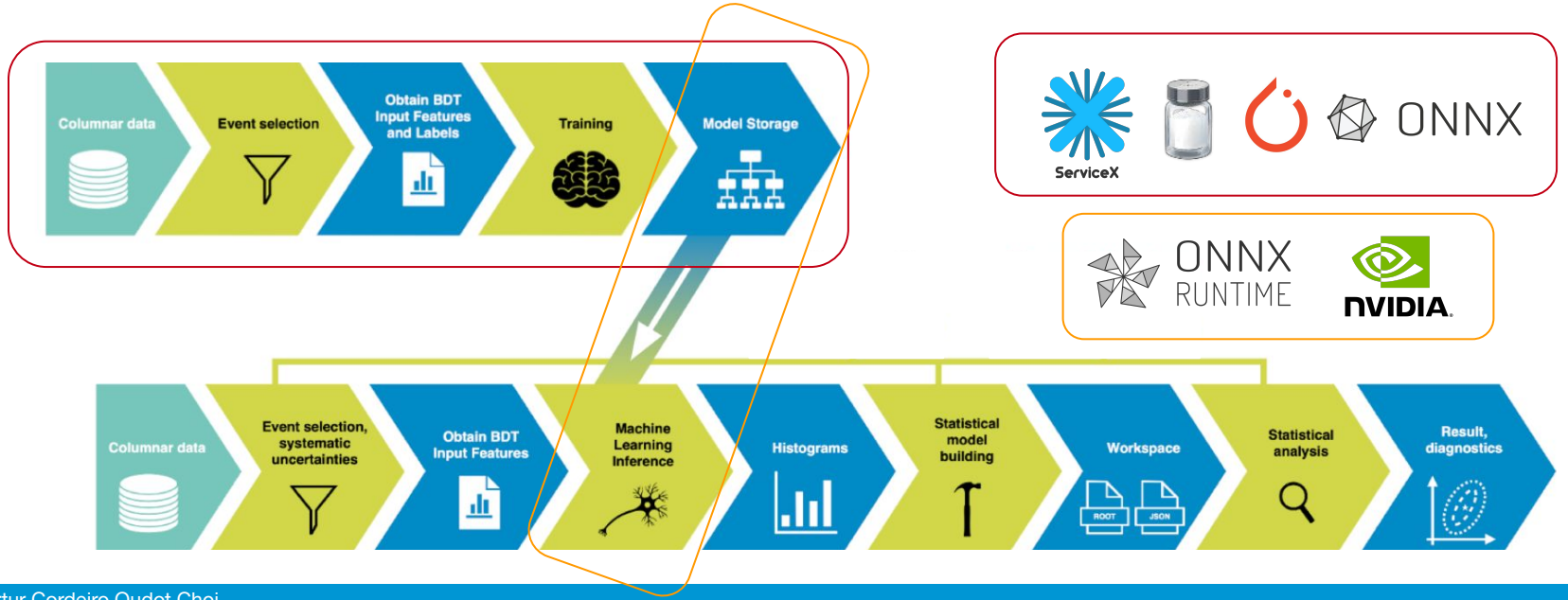


Performance optimizations to get here:
- *Reading data through XCaches*
- *Optimized basket sizes*
- *Bulk data reading ("preload")*



ML Framework

- The Integration Challenge features a **ML framework** combining:
  - Know libraries in ATLAS (Pytorch, ONNX, Salt,…)
  - Modern dataset builders (training, testing) using ServiceX
  - Inference capacities on CPU & GPU (distributed work, inference-as-a-service)



ServiceX & ML datasets

- **ServiceX is query-based data delivery service**
  - Dataset lookup, column extraction, event cuts, variable transformation

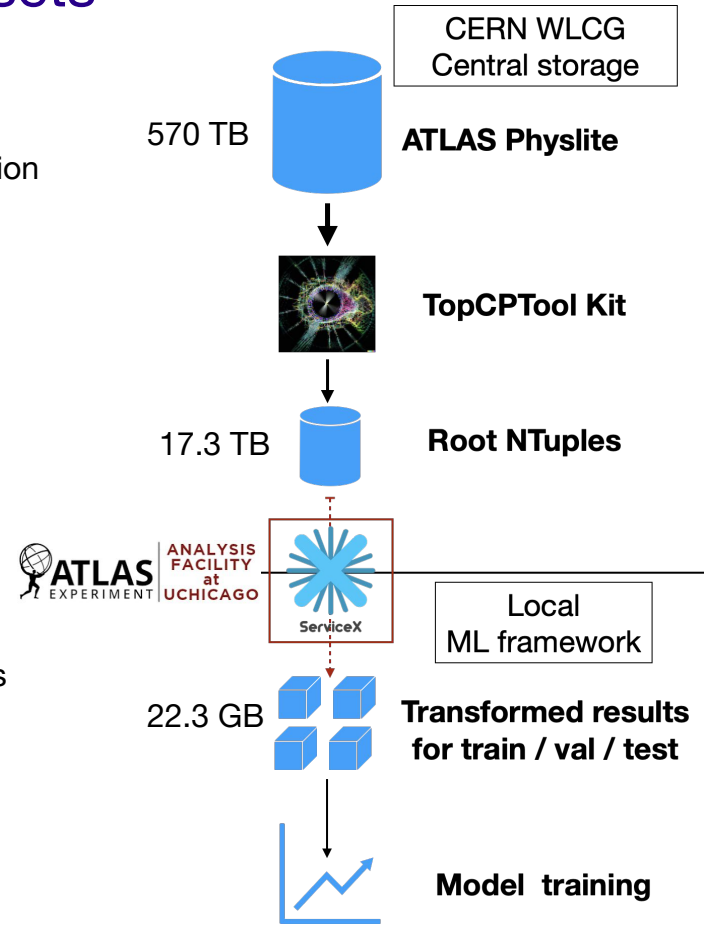
    Workers deployed with py HEP libraries, ATLAS/CMS software
  - Fast & Scalable: deployed @ UChicago Analysis Facility

    Kubernetes pod autoscaling, XCache, CPU resources, high bandwidth to WLCG
  - Simple interface to build data requests

    Functional programing & declarative syntax; *awkward* expressions…

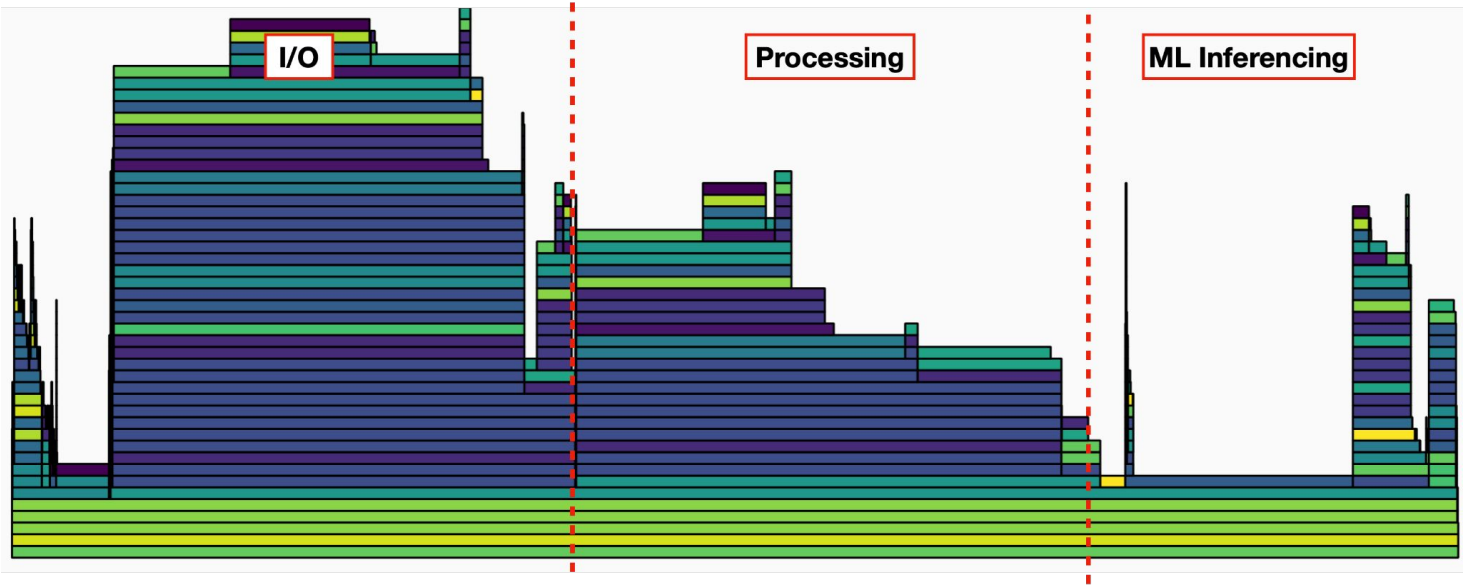
- Building ML datasets in the integration challenge
  - Request specific columns with event selection from remote NTuples
  - Good performance: **build local ML datasets in 7 mins**

    17.3 TB across 8k files - > 22.31 Gb across 3k files
  - Train / val / test splitted, converted and ready to use in [salt-ml]



Inference during analysis

- Fairly rough divisions on CPU preliminary tests:
  - I/O (decompress, get, basket_to_array) is about 40%
  - Processing, histograming (broadcasting, materialization, ufuncs) takes about 40%
  - **ONNX Inferencing** (preparing input, loading model, running inference) is lower than **20%**



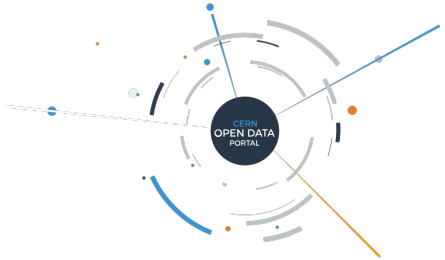
Summary

- **Integration Challenge** uses **Python** (especially Scikit-HEP)-based stack to define **analysis solutions** for HL-LHC
  - The stack is working well overall, UX and performance feel good!
  - Distributed analysis challenges have lead to the adoption of new workflow patterns (with `histserv`)
  - The inclusion of modern tools like ServiceX allowed exploring new ways to integrate common ATLAS frameworks
- The ML framework shows **good performance** in preparing training samples & the inferencing (while being reasonable) is being optimized to improve walltime
- Come chat with us this week about any details you might be interested diving into!

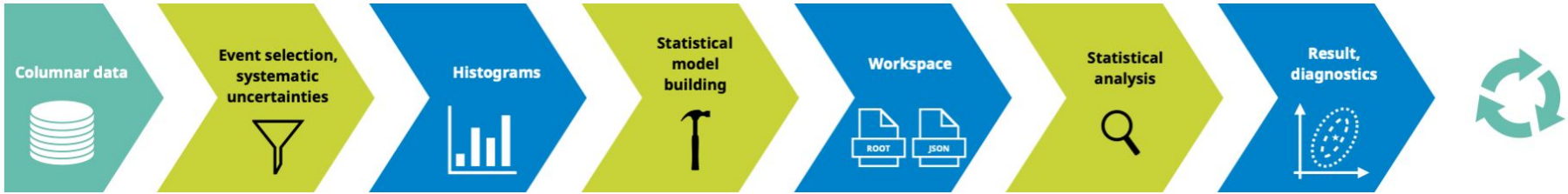
Backup

Analysis Grand Challenge (AGC)

- The **Analysis Grand Challenge (AGC)** defines a **physics analysis task** with **Open Data** to test **HL-LHC workflows**
  - columnar data extraction from large datasets & data processing into histograms
  - statistical model construction and statistical inference, relevant visualizations
  - ML training & inference



- Focused on **workflow & interfaces**, not computational cost: **1.8 TB inputs, 1 B events**



The 200 Gbps Challenge (2024)

200 Gbps →

180 TB dataset

25% of file content

30 minutes

We identified a **200 Gbps data rate** as relevant target in 2024 and pursued this goal:

see https://arxiv.org/abs/2503.15464

CMS NanoAOD example

With **2 kB / event**, this means

- **90 B events,**
- **50 MHz event rate,**

or 1k cores with 50 kHz and 25 MB/s each.

ATLAS PHYSLITE example

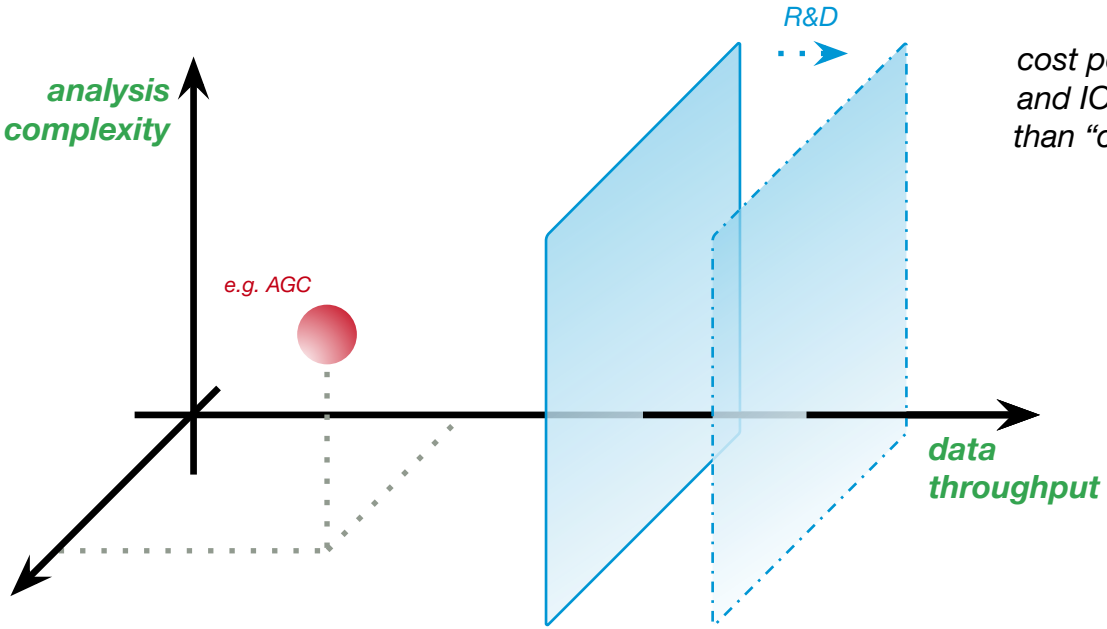
With **10 kB / event**, this means

- **18 B events,**
- **10 MHz event rate,**

or 2k cores with 5 kHz and 12.5 MB/s each.

Breaking down physics analyses

Which of this space do we populate now, which in the future?



cost per event / throughput and IOPS easier to quantify than "complexity" or special requirements

Salt integration

The IC's ML Framework takes advantage of Salt-ML

- Heavily tested and documented
- Familiar tool in ATLAS presenting ONNX export
- Simple CLI-based interface with config files

  (model task, architecutre, training parameters, file paths…)

A custom dataloader was built into the ML framework

- Convertor between ServiceX's output and Salt's HDF5 input
- Making the IC's ML data builder tool an efficient option to build Salt training datasets

```
from ml_framework.data_preprocess import to_hdf5, split_h5
salt_file = "./ml_framework/data/signals_backgrounds_final.h5"
to_hdf5(samples_dict = mc_dict, output_file = salt_file, max_events_per_sample= 500_000)
split_h5(salt_file, train_frac=0.7, val_frac=0.10)
```

Abstract

The "Integration Challenge" is the latest IRIS-HEP project exploring physics analysis pipelines for the HL-LHC. Compared to earlier iterations, this one focuses on capturing the detailed needs of experiments. The contribution presents the analysis pipeline built for the ATLAS version of the project, which is based on a variety of Scikit-HEP and IRIS-HEP projects (including uproot, awkward, coffea, hist, ServiceX). We provide an overview of the components involved and the optimizations that have led us to the current implementation. While still relying on Dask for distributed execution, a number of changes compared to traditional coffea-based approaches have both expanded functionality and increased robustness.