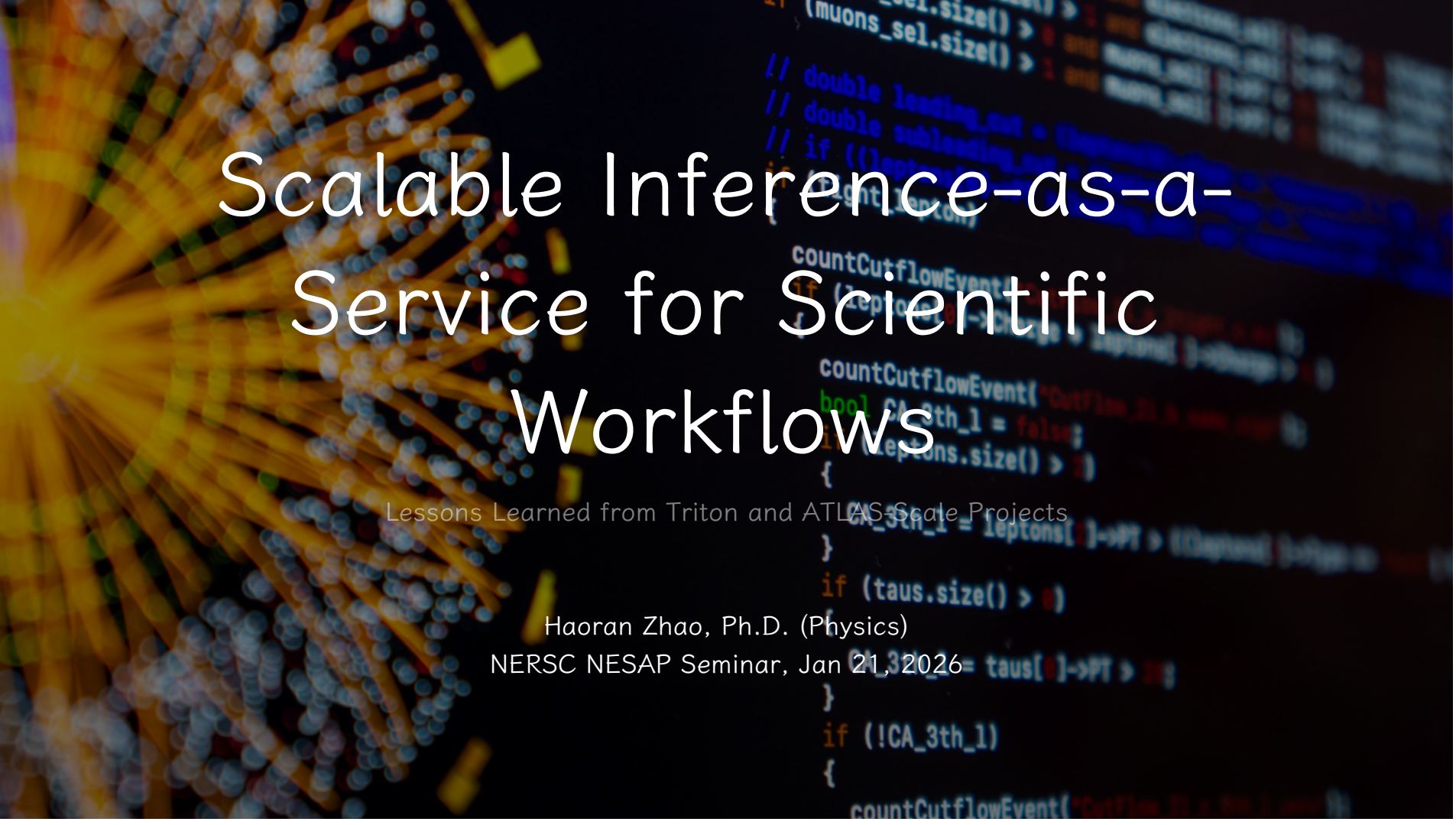


Scalable Inference-as-a-Service for Scientific Workflows

Lessons Learned from Triton and ATLAS-Scale Projects

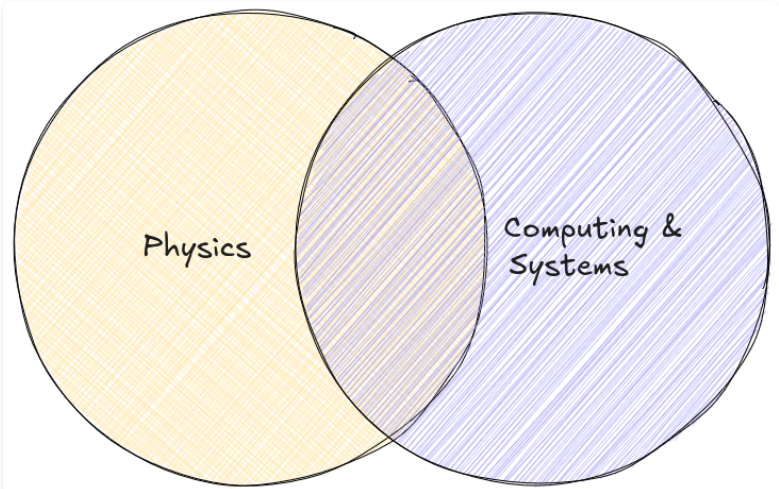
Haoran Zhao, Ph.D. (Physics)

NERSC NESAP Seminar, Jan 21, 2026

Who am I?

How I approach physics, computing, and systems

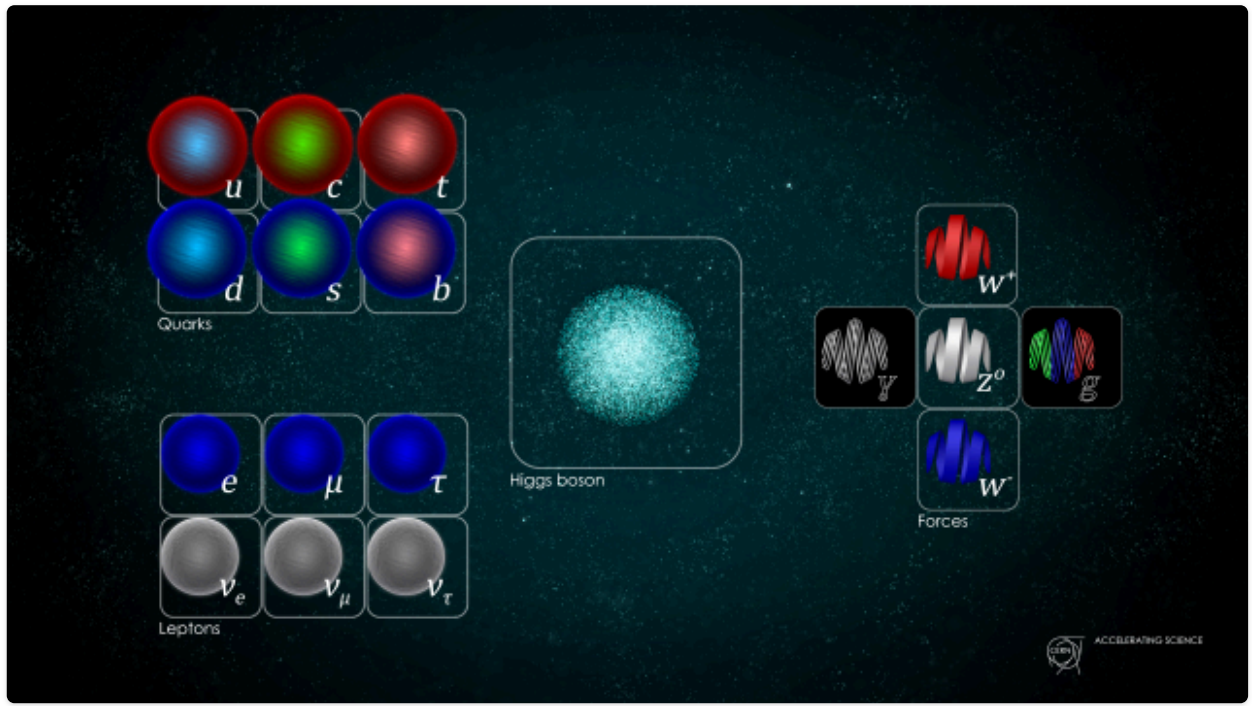
- **Experimental Particle Physicist** — once hunted for charged Higgs bosons
- **A long-time NERSC User** — from Cori to Perlmutter, looking forward to Doudna
- **Tinkerer** — build things to understand how they actually work
- **Curious across domains** — physics, ML, DevOps, agentic AI, whatever breaks next



Intro to LHC and ATLAS

Standard Model of Particle Physics

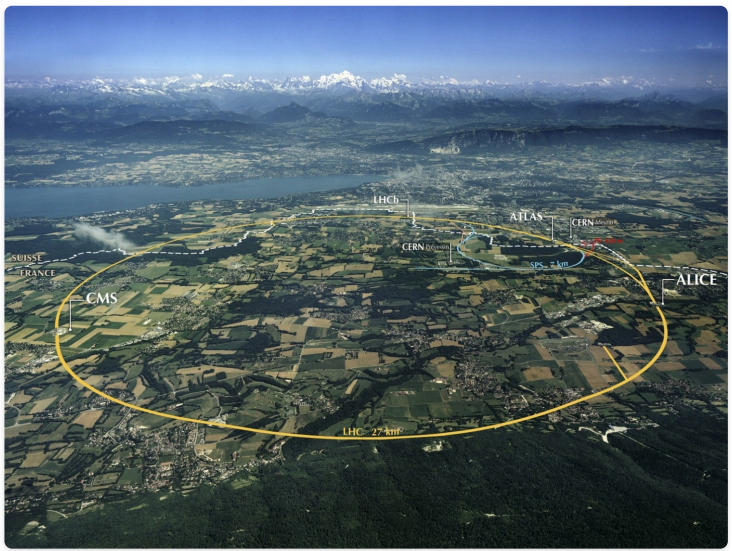
fundamental building blocks + interactions

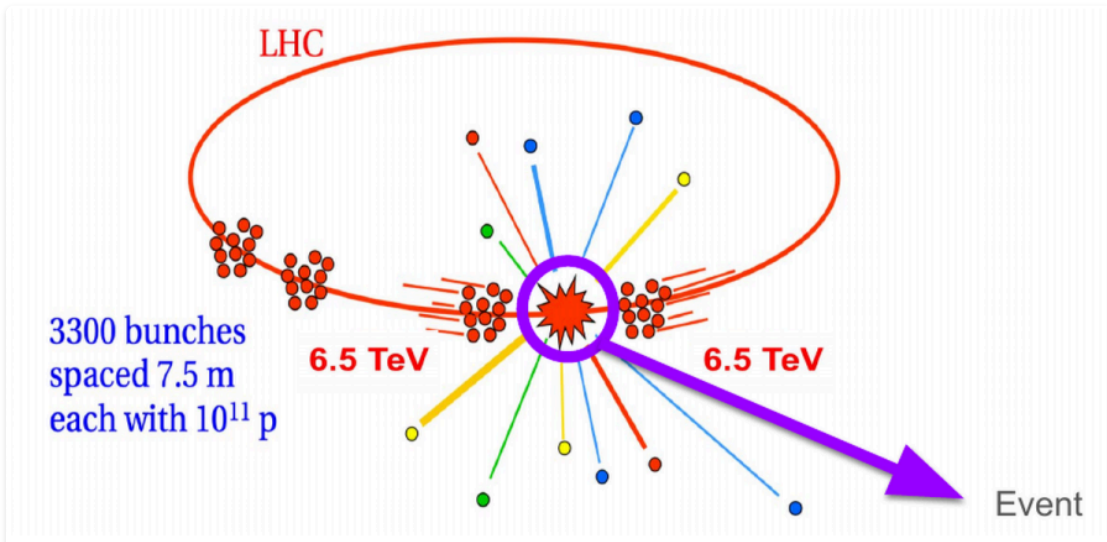


Credit

Large Hadron Collider (LHC)

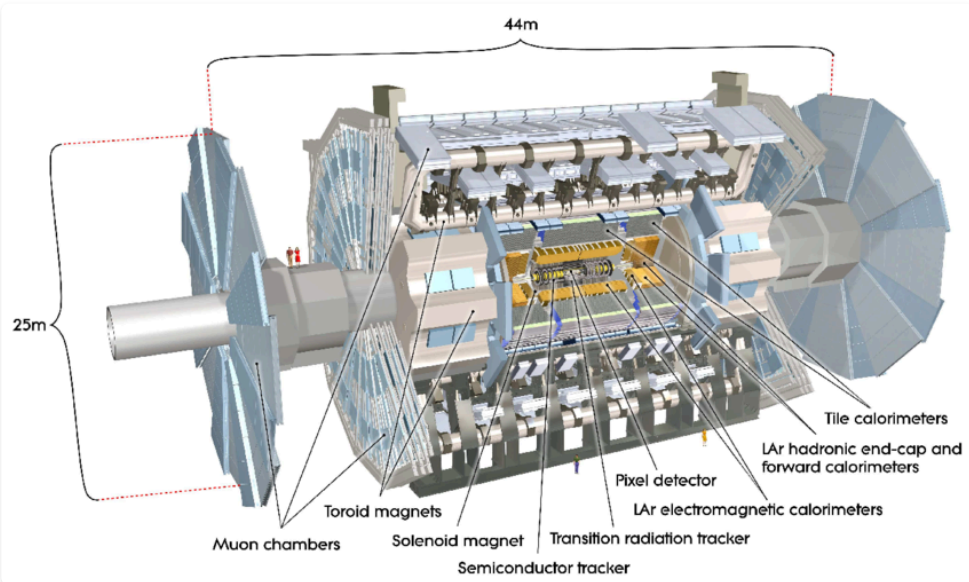
World's largest and most powerful particle collider

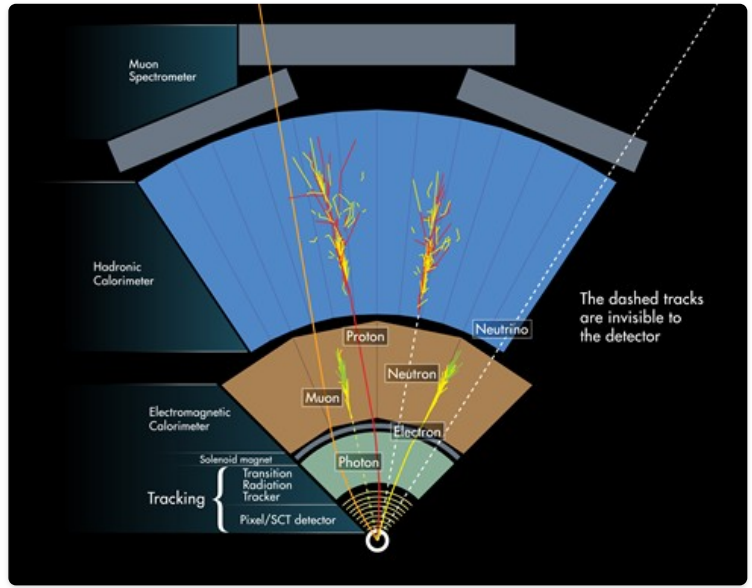




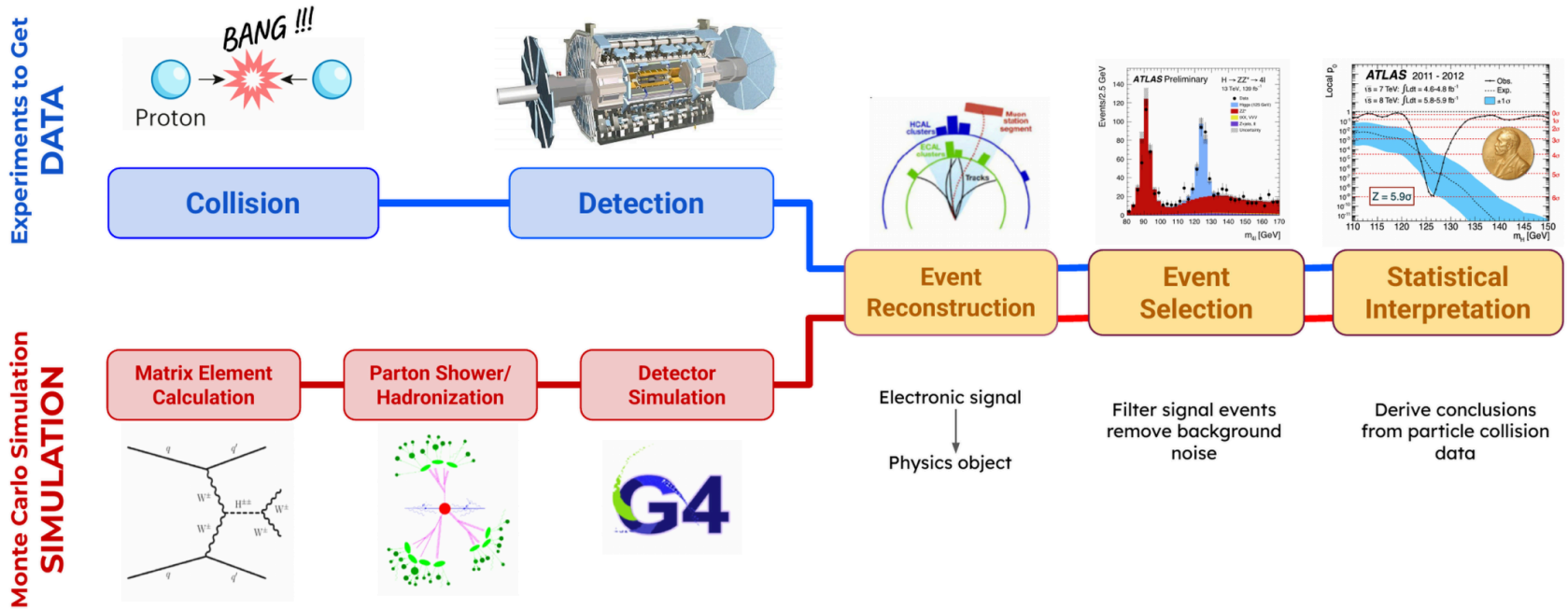
ATLAS detector

the largest general-purpose particle detector at LHC

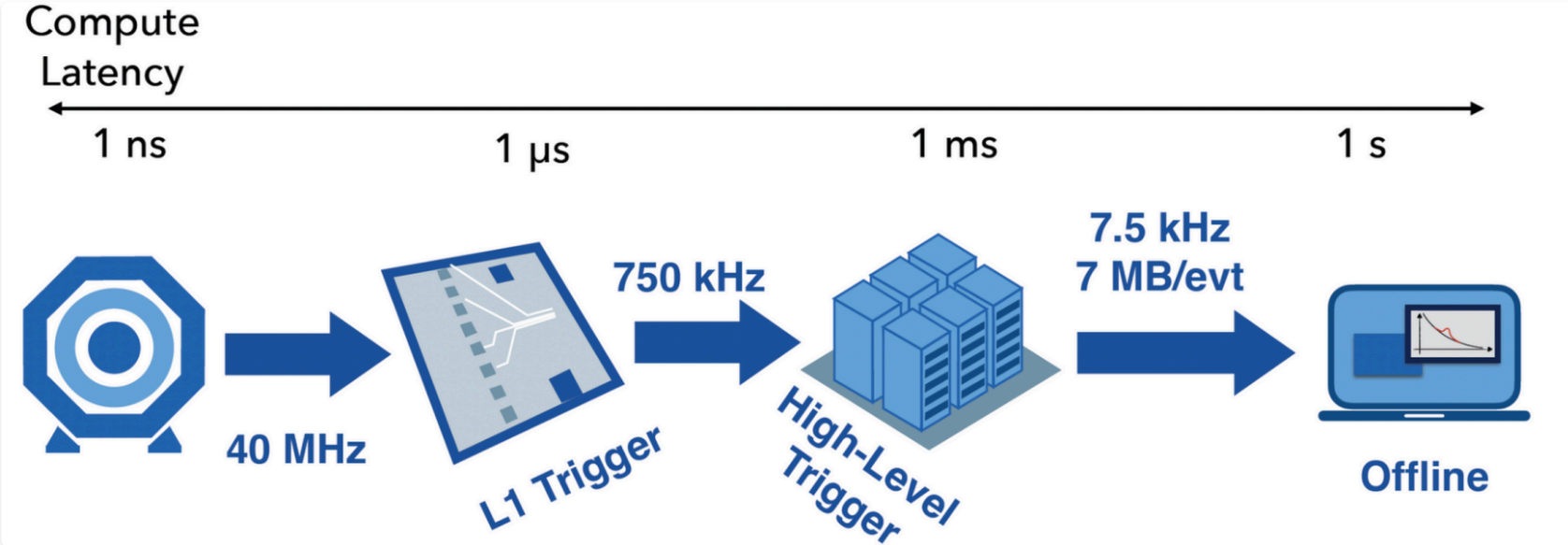




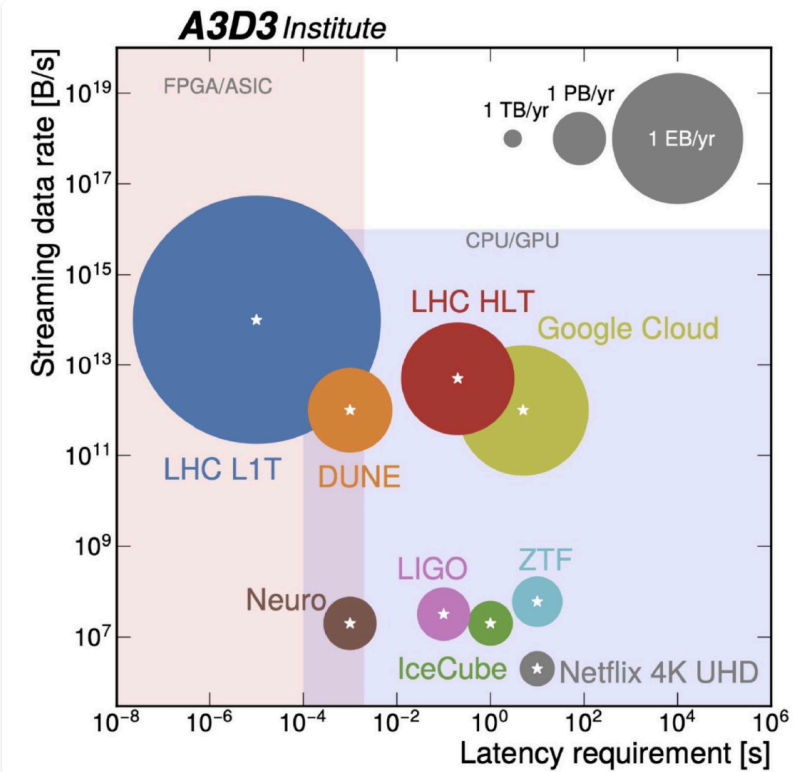
Data Analysis Workflow



Trigger & Latency



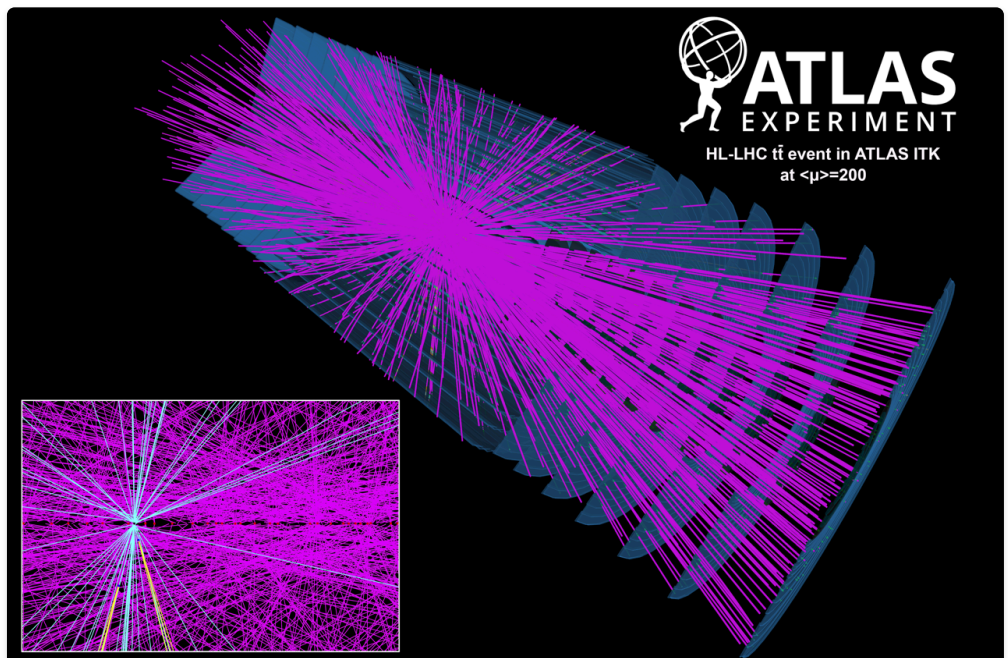
Latency vs Data Rate



Inference as a service

The compute needs for High-Luminosity LHC

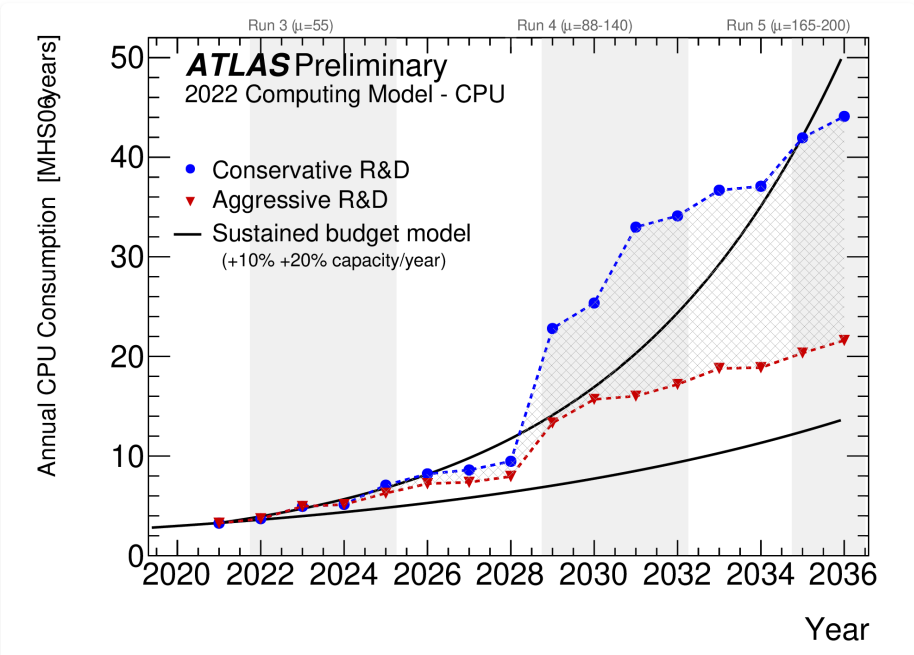
- HL-LHC dramatically increases event complexity
  - extreme pile-up
  - dense tracking environment



Credit

The compute needs for High-Luminosity LHC

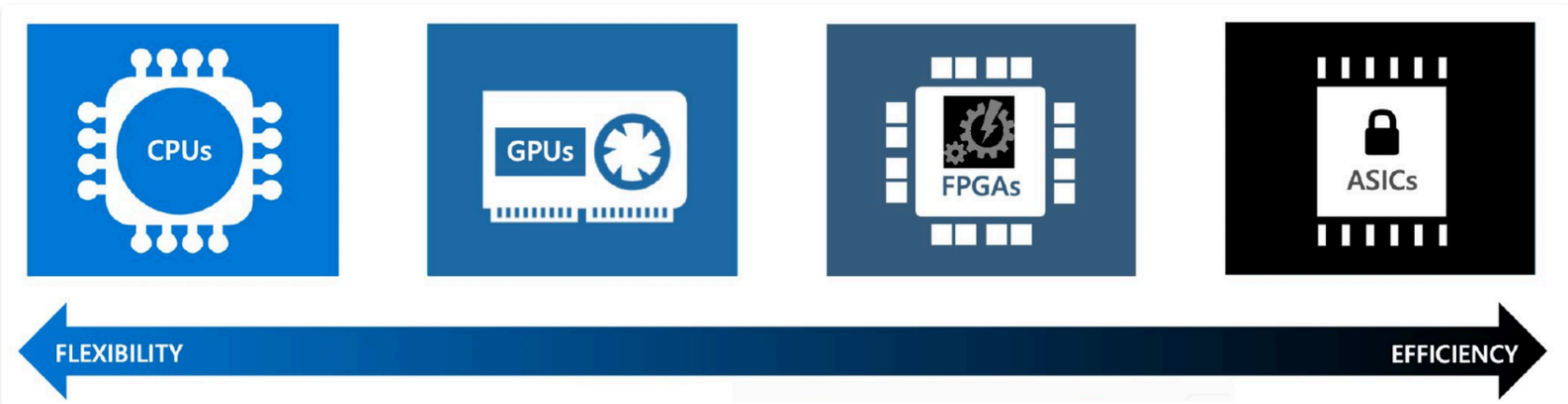
- Reconstruction cost grows super-linearly
- CPU scaling hits power and cost limits



Credit

Trending in Industry: Heterogeneous Computing

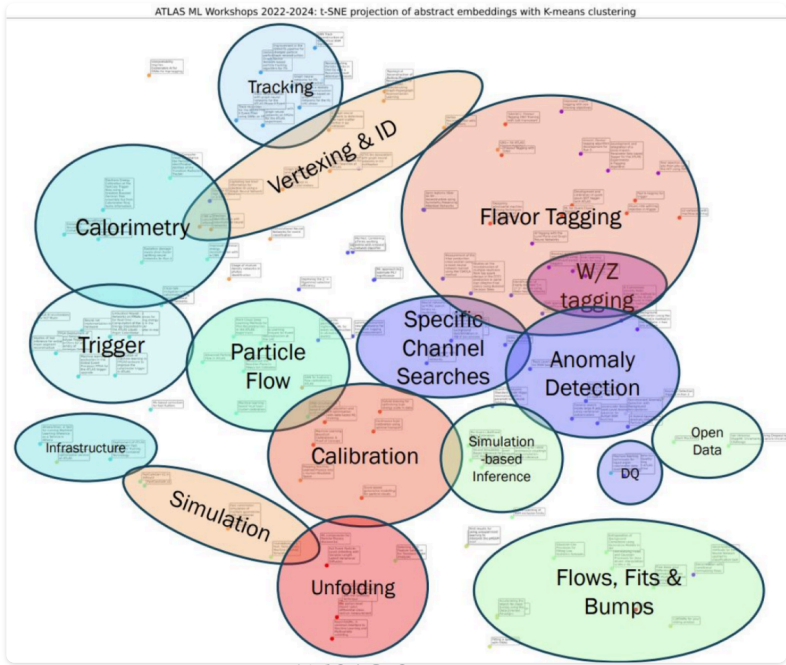
- Rise of specialized accelerators (GPU, TPU, NPU, FPGA)



Credit

ML Trends in ATLAS

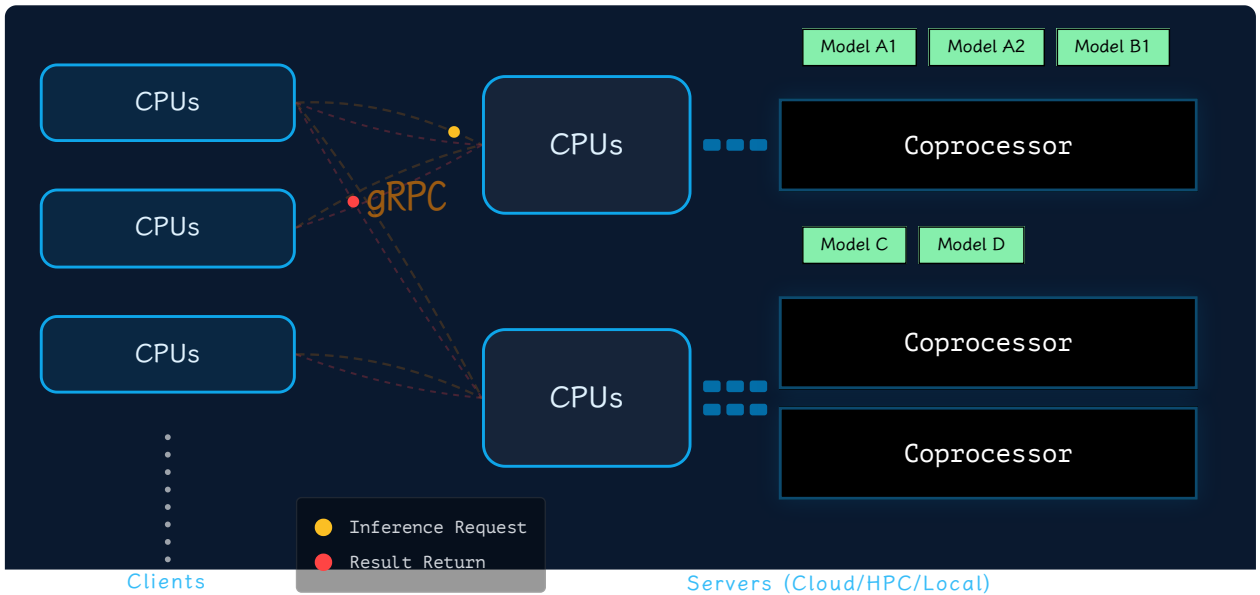
- Many ML models are deployed into Athena
- Moving to a production phase
- Need an efficient, scalable, maintainable way to run inference



Credit

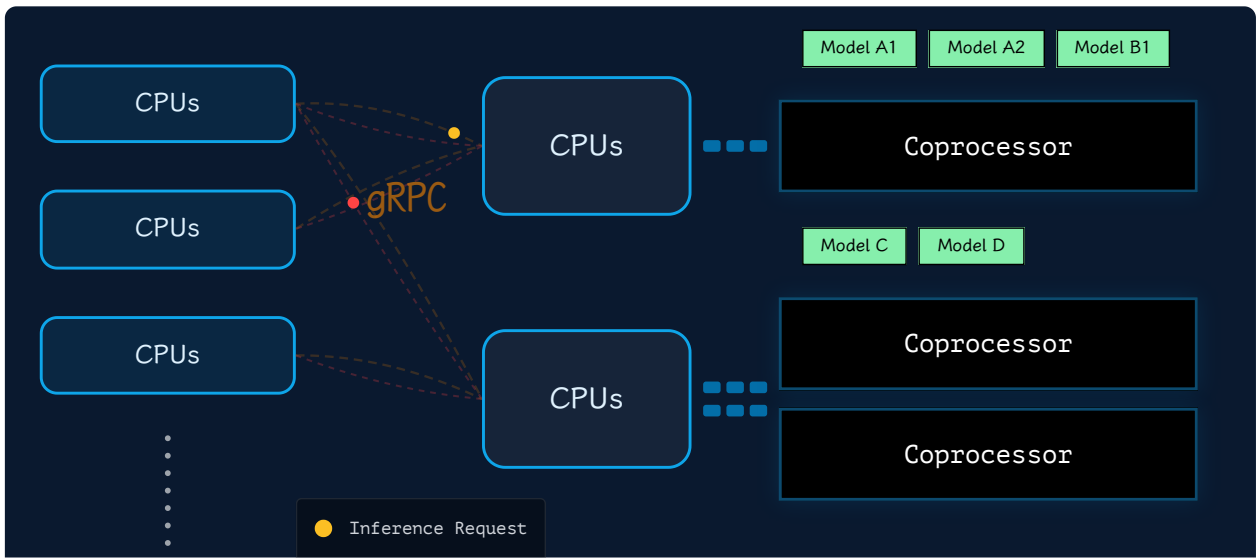
Inference as a Service

- Inference runs as a shared service
- Jobs send requests instead of managing hardware



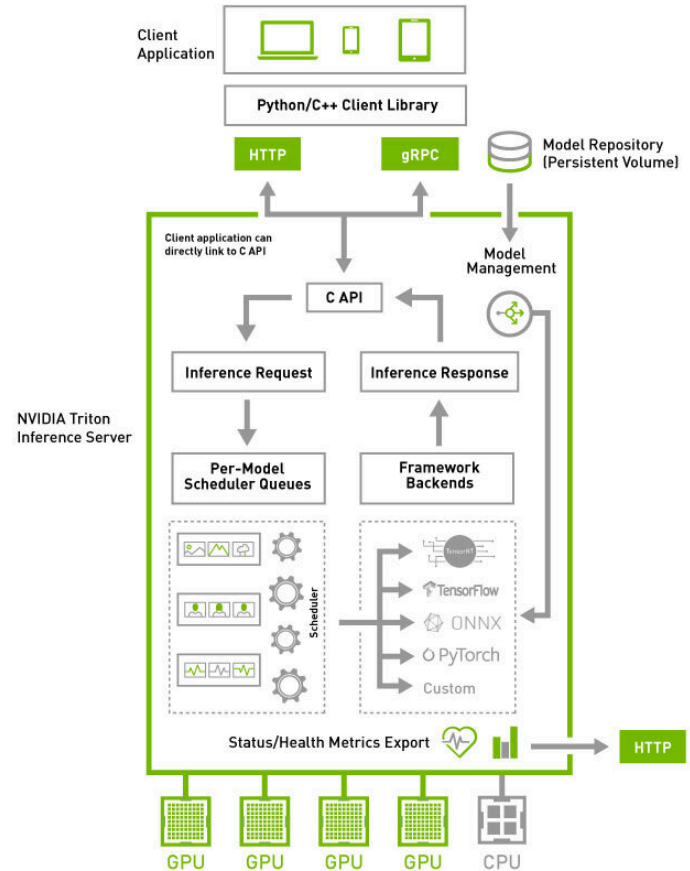
Inference as a Service Benefits

- Tuning GPU utilizations
- No heavy software stacks in clients
- Improves scalability and maintainability at production scale



Nvidia Triton Inference Server

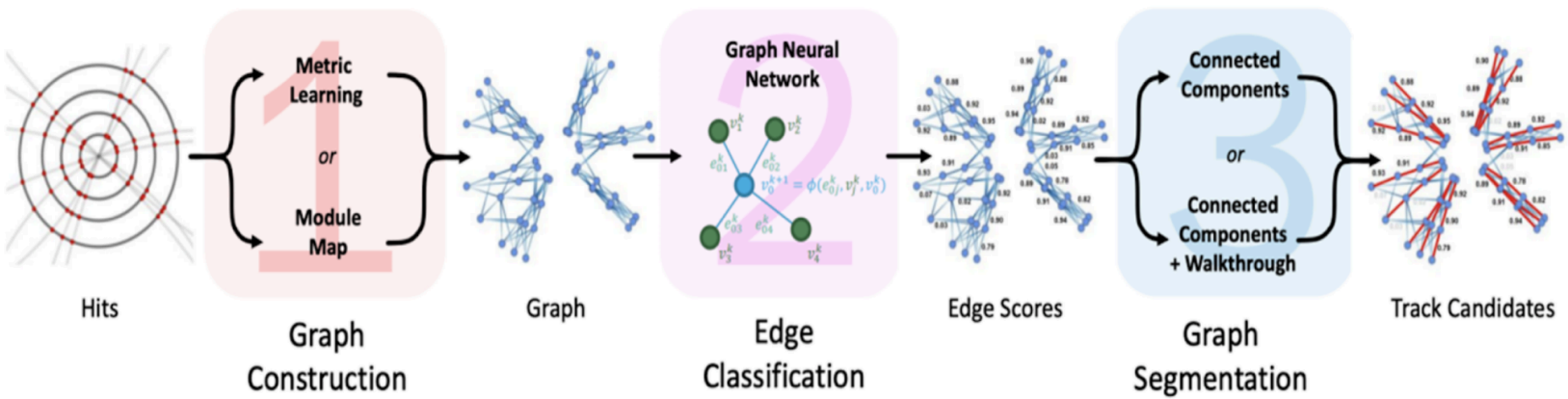
- Production-ready inference serving
- Designed for scalable inference deployment
- Schedules inference requests across shared resources
- Supports custom backends



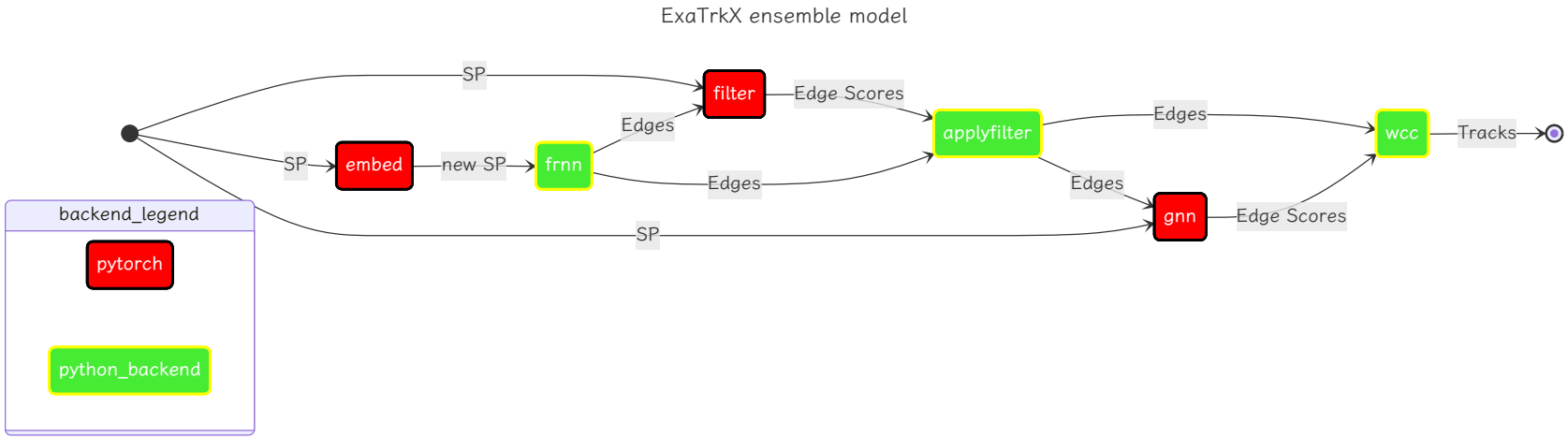
Case: ExaTrkX Pipeline as-a-Service

- Tracking is the most time-consuming part of event reconstruction
- Modern tracking relies heavily on GNN-based inference
- A natural first attempt: ExaTrkX as a Service
- ExaTrkX projects led by LBNL scientists
- Track Finding Algorithm:

  Input: Space points, Output: track candidates



A first try: Triton ensemble models



- ExaTrkX is a sequential and stateful pipeline
- Triton ensemble introduces hard model boundaries
- Python backends amplify latency
- GPU affinity becomes difficult to guarantee

Why C++ Custom Backend?

- Stateful, tightly-coupled pipeline doesn't fit the ensemble models

What Custom Backend enables:

- Fine-grained control
  - Custom logic
- Low latency
- Legacy integration w/ C++ codebase
  - ACTS, Traccc

Custom Backend Design (1)

Standalone-first development

- Modular: develop a fully working C++ standalone GPU pipeline first
- Only after the pipeline is stable: wrap it as a Triton custom backend
  - lifecycle

```
#include "ExaTrkXTrackFinding.hpp"

void ExaTrkXTrackFinding::getTracks(
    // ************
    // Embedding
    at::Tensor eOutput = e_model.forward(eInputTensorJit).toTensor();
    // Filtering
     at::Tensor fOutput = f_model.forward(fInputTensorJit).toTensor();
    // GNN
    auto gOutput = g_model.forward(gInputTensorJit).toTensor();
    //....
)
```

Custom Backend Design (2)

Interface design for I/O

- Inputs are event-based, variable length -> per event request inference request
- Only primative data type is supported -> flattened inputs and outputs
- Potential improvements:
  - padding -> dynamic batching
  - stream of `bytes` instead of 1d `int`

```
Event 1: [  • • • • •  ]      ➡ inference request 1
Event 2: [  • • •  ]          ➡ inference request 2
Event 3: [  • • • • • • •  ]  ➡ inference request 3


# response, original design, vector of vector, spacepoint id
3 tracks: [  • • • • •  ], [  • • •  ], [  • • • • • • •  ]

# response, flatten 1d, track cluster id
[  • • • • • • • • • • • • • • • • ]
```

Custom Backend Design (3)

Explicit GPU affinity and device guarding

- Obtain GPU device ID from Triton model instance
- Bind one inference pipeline instance to one specific GPU
- Use explicit device guarding to avoid implicit data movement
- Enables predictable scale-up across multiple GPUs

```
#include "ExaTrkXTrackFinding.hpp"
// Called once per Triton model instance
TRITONBACKEND_ModelInstanceInitialize(instance)
{
  int32_t device_id;
  TRITONBACKEND_ModelInstanceDeviceId(instance, &device_id);

  config.device_id = device_id;
  engine = ExaTrkXTrackFinding(config);  // bind engine to GPU
}
  { // inside ExaTrkXTrackFinding
    at::cuda::CUDAGuard device_guard(m_cfg.device_id);
    torch::Device device(torch::kCUDA, m_cfg.device_id);
  }
```

Benchmark Setup

Experimental design (high level)

- Perlmutter GPU compute node, Containerized inference service
- Client and server setup on the same node
- Scalibilty setup by `instance_group` in model `config.txt`

Data

- Small events (PU 0, O(100) spacepoints) for functional validation
- Large events (PU 200, O(10^6) spacepoints) for stress testing

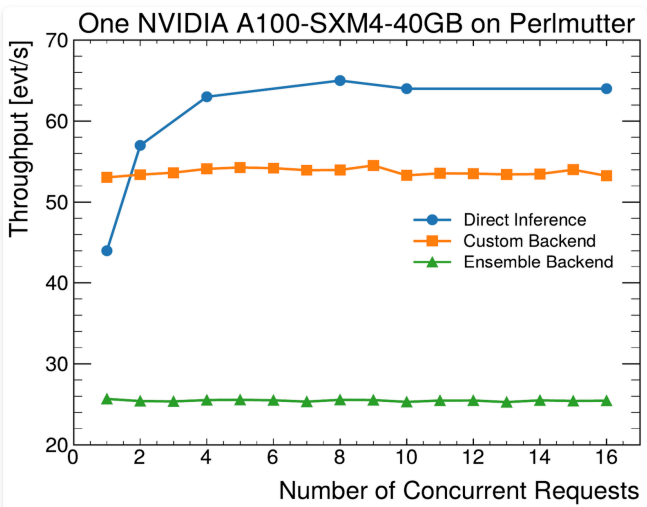
Metrics & observations

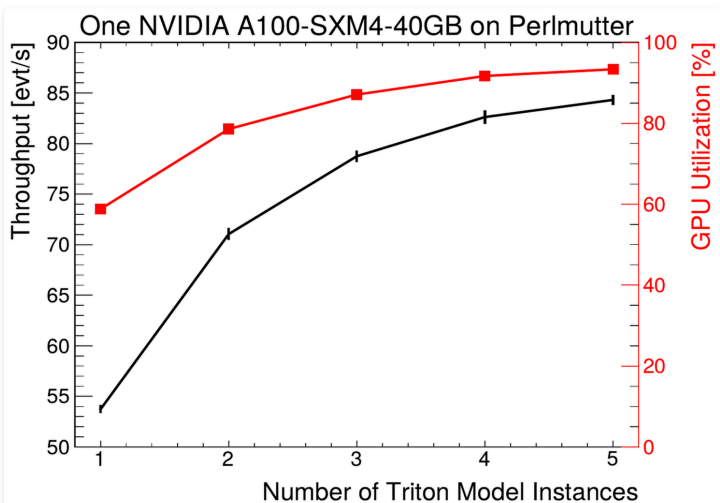
- Raw measurement from `perf_analyzer`
- Throughput under over-saturated request conditions

Results: Ensemble v.s. Custom Backend

Preliminary Tests on PU=0 *events*

- Efficient custom backend
- Good scalibility observed
- Task compute bound

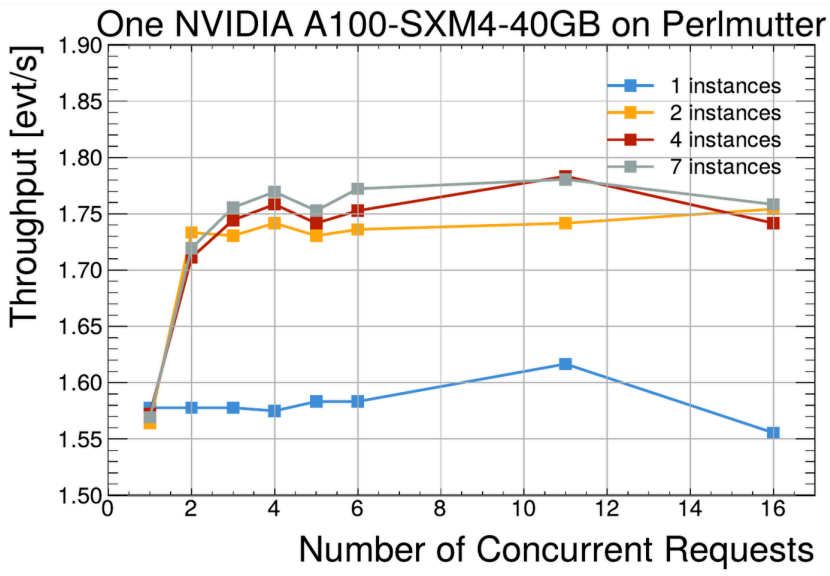


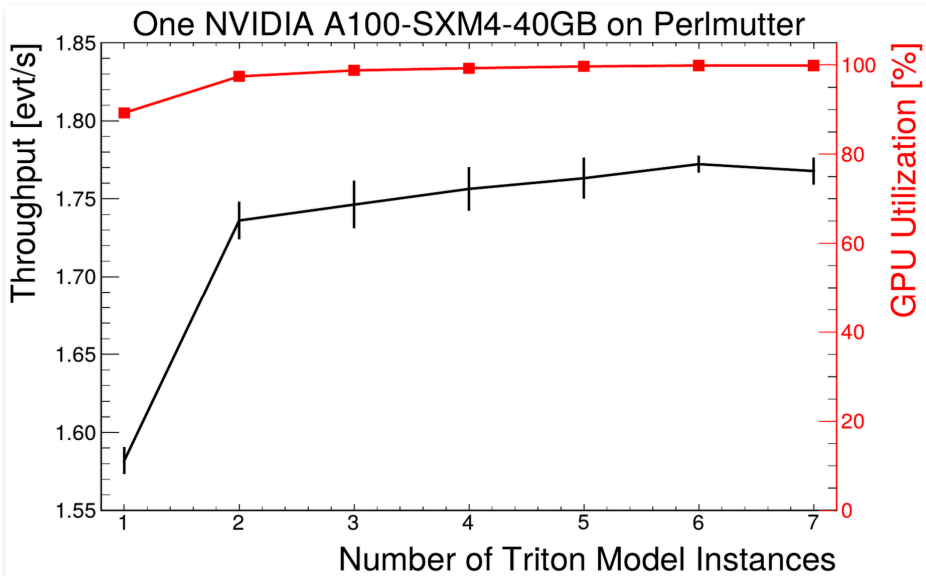


Results: Exa.TrkX as a Service (1)

a more realistic set-up, w/ PU=200

- 10% improvement in throughput
- Task compute bound

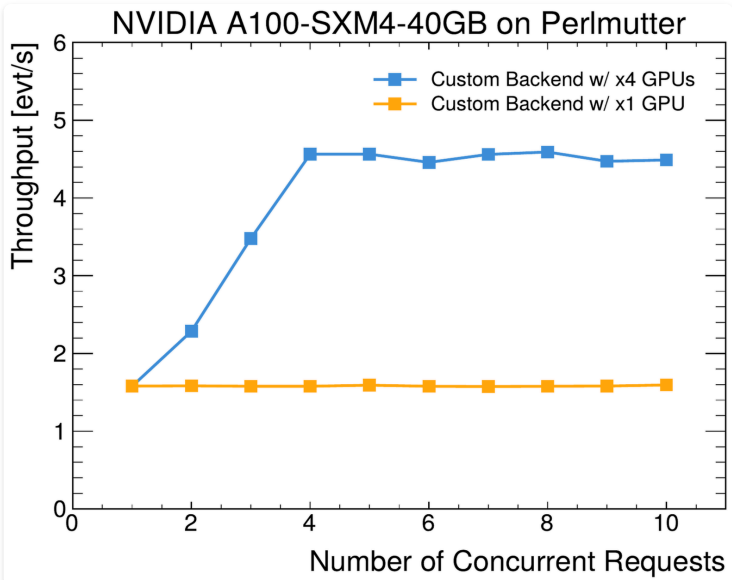




Results: Exa.TrkX as a Service (2)

a more realistic set-up, w/ PU=200

- Multi-GPU scalibility, one instance per GPU



Reusability & Documentation

- Tutorials on different HPC
- Adaptions and extenstion to other software suite
- Development guide
- Now maintained by my mentee Ph.D. student

Welcome to Athena Triton

Contents

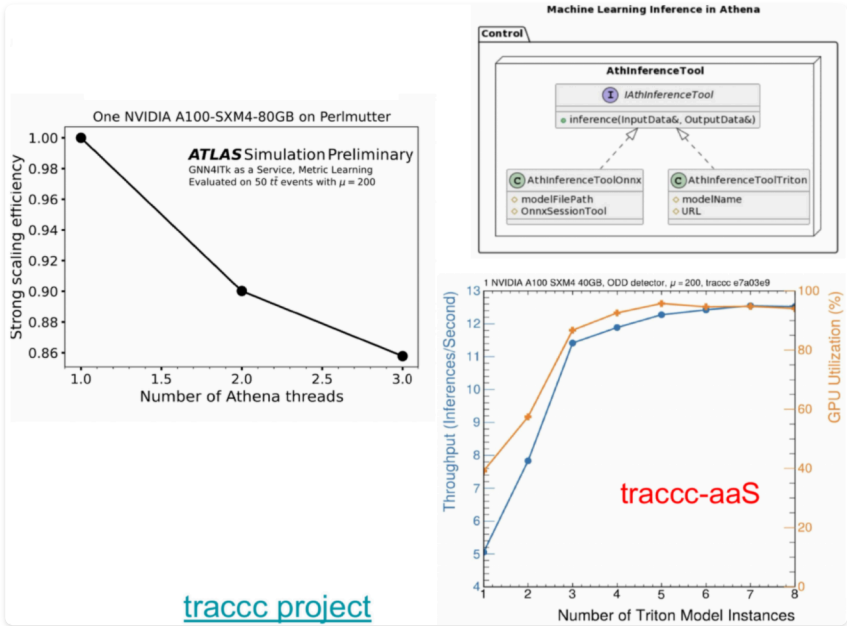
Introduction

Welcome to Athena Triton, the innovative fusion of ATLAS's powerful Athena software package with the cutting-edge capabilities of the Triton Inference Server. ATLAS, one of the largest particle physics

AthTriton in Athena

custom backend aaS in large reconstruction workflows

- Ongoing client integrations in Athena
- Enables ML inference as a service across:
  - Online trigger pipelines
  - Offline reconstruction and R&D
- Demonstrates portability beyond a single algorithm or environment



-->

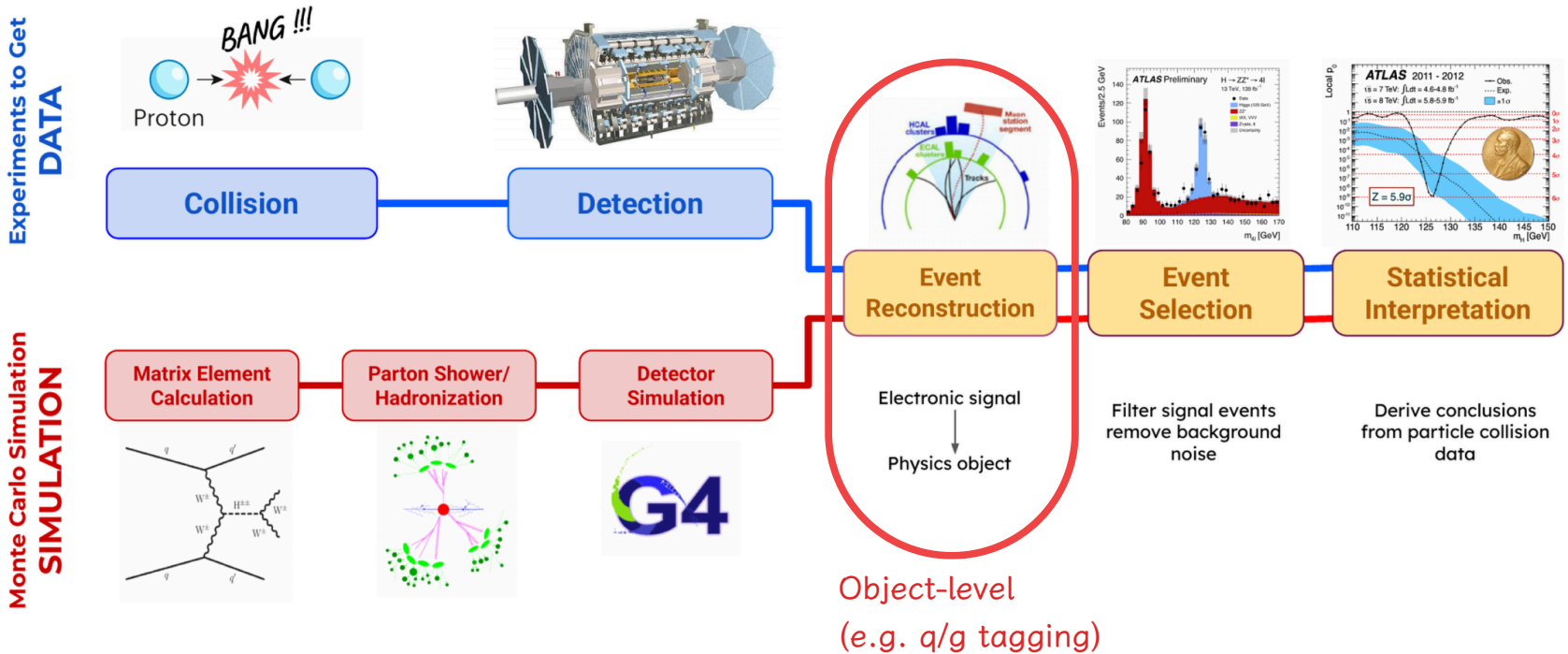
Inference as-a-Service Full Picture

- Can we run it on the Doudna system?

Takeaway

- Ensemble model introduces avoidable overhead
- Custom backend removes pipeline boundaries
- Tracking-as-a-Service matches direct GPU performance
- Enables efficient GPU sharing across many CPU clients

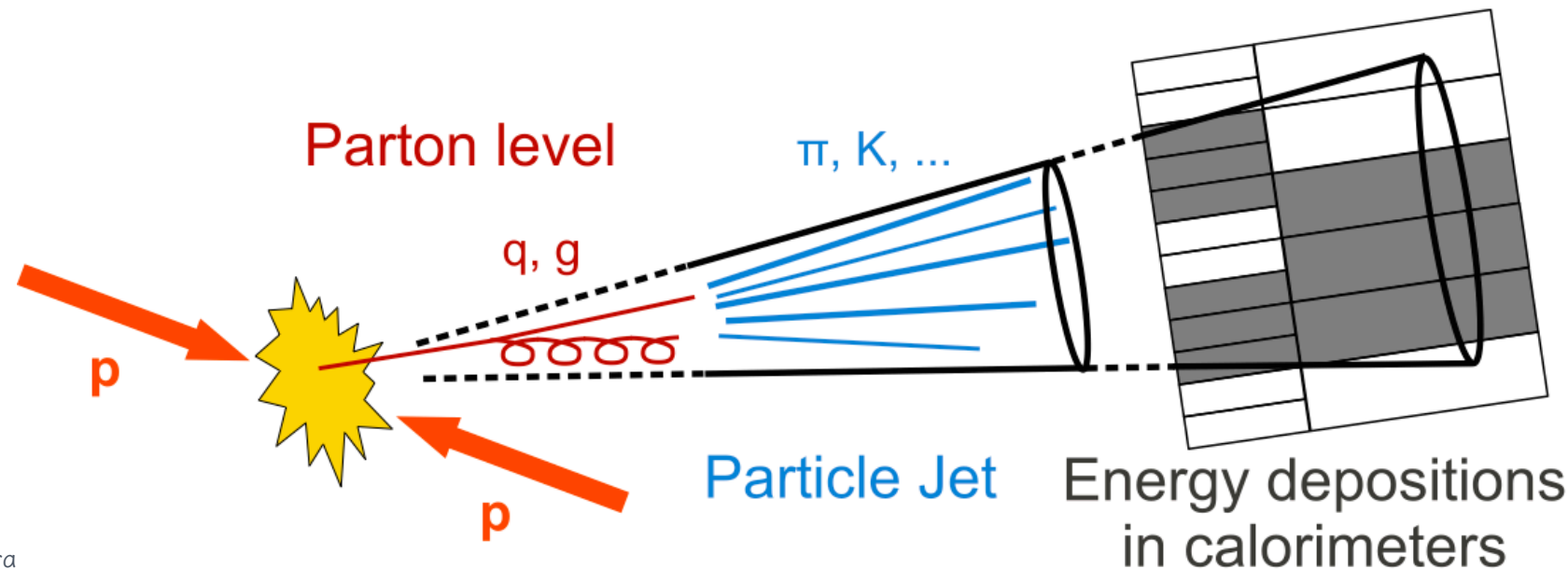
Physics object improvement



Quark / Gluon Tagging and Calibration

Quark / Gluon Jets

- Distinguishes quark- and gluon-initiated jets
- Exploits differences in radiation and substructure
- Implemented as a per-jet tagging decision or score



From tagging to calibration

- Tagging introduces efficiencies
- Taggers are developed and validated in simulation
- Efficiencies differ between simulation and data

Why calibration is unavoidable

- Analyses rely on tagger efficiencies
- Differences are corrected with **scale factors**
- These corrections depend on:
  - working points
  - kinematic regions
  - systematic variations

From calibration to workflow

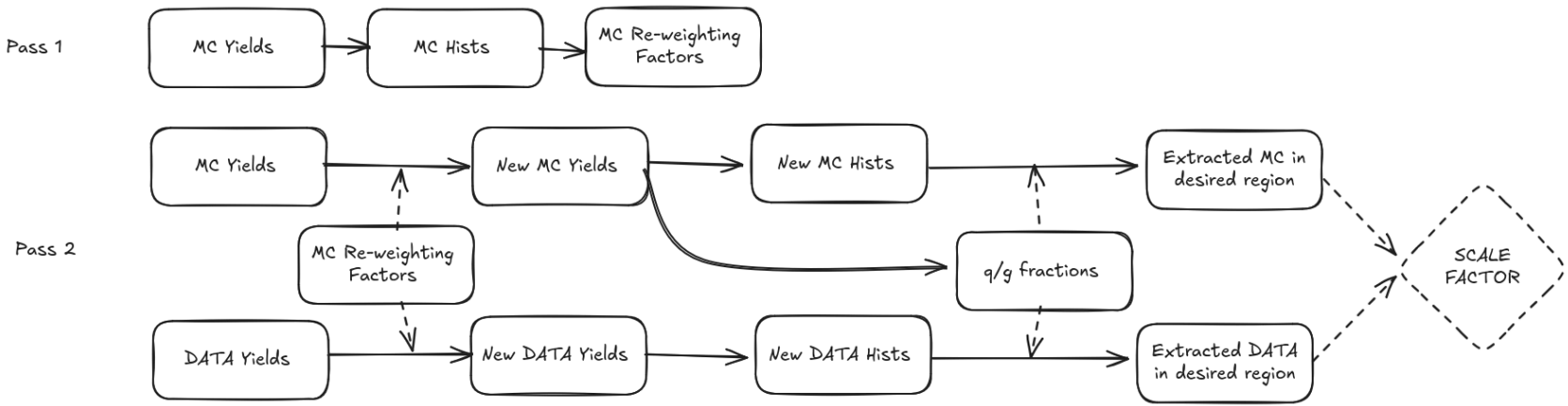
- Large simulated samples
  (O(1000) ROOT files, ~ 3GB each, total TB-scale data)
- Many systematic variations
  (JES/JER, parton shower, hadronization, PDF, ⋯)
- CPU-bound event- and jet-level computation
- Repeated passes over the same datasets

Naive approaches do not scale.

–> This motivates a structured, parallel workflow

Calibration workflow overview

For each systematic variation, do the following:



Workflow design principles

Staged dataflow

- Analysis decomposed into well-defined passes
- Each stage produces explicit, reusable artifacts

```
root2pkl(...)
pkl2predpkl(...)
predpkl2hist(...)
final_reweighting(...)
```

Workflow design principles

Task-level parallelism

- Parallelism across independent files
- No shared state between workers

```
with ProcessPoolExecutor(max_workers=n_workers) as executor:
    executor.map(stage_function, inputs)
```

Workflow design principles

Deterministic merging

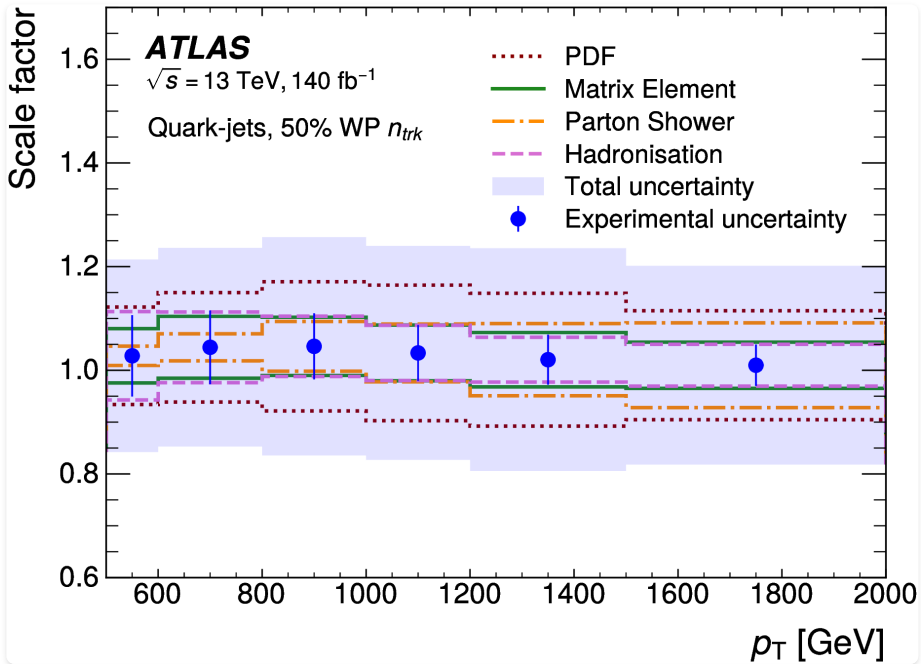
- Parallel outputs combined in an order-independent way
- Final results reproducible by construction

```
for h in hists_list:
    merged_hist += hist
```

What we intentionally did

- Naturally scalable
  - Parallelism ove files, no coordination overhead
- Stable and reproducible
  - Corrections frozen before application
  - Results independent of execution order

Representative results

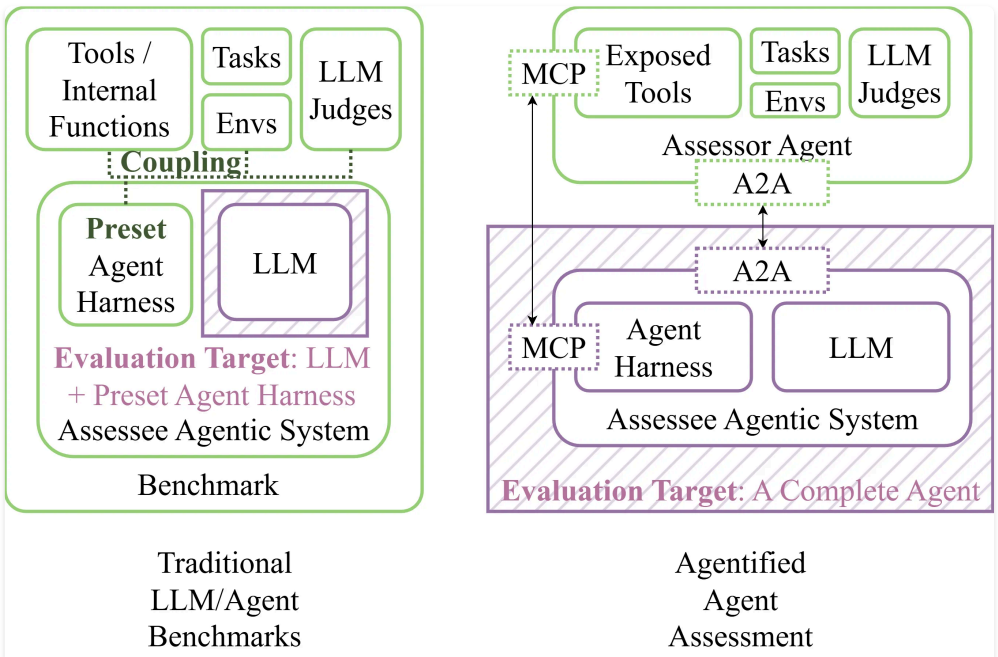


Side Projects

Disclaimer: For Fun

Berkeley Agentic AI AgentBeats Competition

Agentified Agent Assessment



Credit

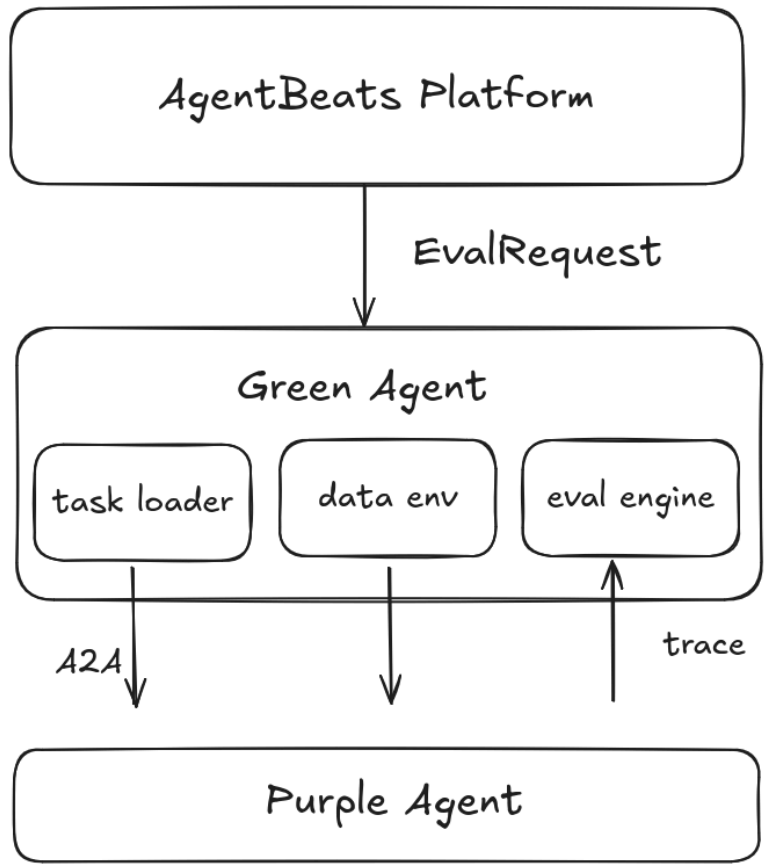
Why an HEP-*ex* AnalysisOps Benchmark?

- HEP analysis is **fixed-pattern** and **operations-intensive**
- Core steps map to standard ML problem types
- The real bottleneck is orchestration and integration
  - HEP analyses span heterogeneous tools, formats, and computing environments
  - Human effort is dominated by *glue* work: code, configs, validation, documentation
- Can agentic AI execute this flow end to end?

HEP-ex AnalysisOps Benchmark

As a starting point,

- Green agent as an analysis host and evaluator
  - task orchestration
  - data provisioning
  - evaluation & scoring
- Purple/White agent
  - explores data & executes analysis
  - produces physics results
  - MCP tools in phase 2



Evaluation Design

Evaluation Engine

```
package_loader.py
rule_engine.py
aggregator.py
```

Spec-driven evaluation - Easy to add tasks

- `task_spec.yaml`

  Execution & environment contract
- `rubric.yaml`

  Three-layer scoring definition
- `eval_ref.yaml`

  Reference values / expectations
- `white_prompt.md`
- `judge_prompt.md`

Three-Layer Rubric

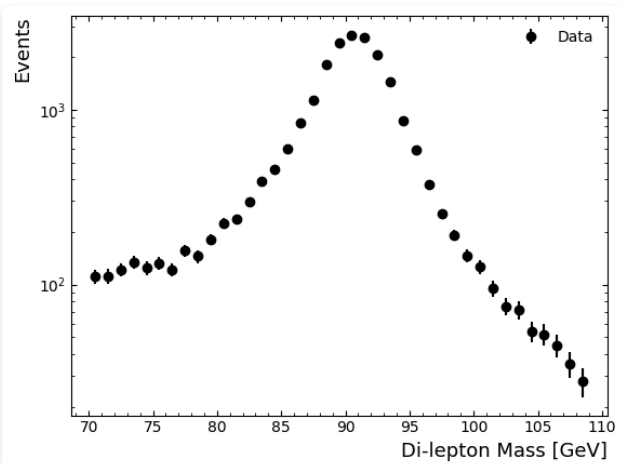
```yaml
# specs/task/rubric.yaml
# Hard correctness first
gates:
  - id: trace_present
    required_fields: ["status", "fit_result", "fit_method"]
    fail_total_score: 0

# Deterministic scoring second
rule_checks:
  - id: mu_closeness
    points: 40
    value_path: "fit_result.mu"
    ...
# Flexible scientific judgment last
llm_checks:
  - id: method_reasoning
    type: llm_reasoning
    ...
```

Example Task: Z → μμ Reconstruction (Z peak fit)

Target observable

- Fitted peak mean μ extracted from di-muon invariant mass
- μ used as a deterministic signal in rule-based evaluation



Tools calling

Only GENERIC funcs are provided.
NO specific workflow is passed in prompt.

```
# data tools
def download_atlas_data_tool()
def list_local_root_files_tool()

# root tools
def inspect_root_schema_tool()
def load_kinematics_tool()

# physics tools
def calc_dilepton_mass_tool()

# fitting tools
def fit_peak_tool()
```

Example Task: $Z \to \mu\mu$ Reconstruction (Z peak fit)

- Scores aggregated from hard, rule-based, and LLM checks
- Structured diagnostics enable partial credit and feedback


```
{
  "fit_result": {
    "mu": 90.75,
    "sigma": 2.28,
    ...
  }
}
```



```
{
  "status": "ok",
  "hard_checks_passed": true,
  "final": {
    "normalized_score": 1.0
  },
  "rule": {
    "score": 85.0
  },
  "llm": {
    "score": 17.0,
    "comment": "Reasonable fit strategy with minor missing
  },

  "issues": [
    { "severity": "warn", "message": "Missing initial fit p
    ...
  ]
}
```

Takeaway

- Evaluates **physics analysis workflows**
- Combines **hard checks**, **deterministic rules**, and **LLM judgment**
- Designed as an **extensible AnalysisOps benchmark**

Looking Forward: Phase 2

- Extend to **more complex physics analyses**
- Enable **more capable agents** to demonstrate advanced workflows
- Explore **agent teams** for collaborative analysis tasks

Self-Hosted Services: Owning the System

What running real services taught me

- Full ownership
  - VPS + Linux administration
  - Firewall (ufw), ports, TLS, domain routing
- Service deployment & isolation
  - Docker, docker-compose, private Docker networks
  - NGINX as reverse proxy
- Operate real, persistent services
  - Mattermost • CodiMD • Calibre-Web • n8n • NextCloud
- TROUBLESHOOTING
  - Limited resources, failures, upgrades, maintenance
- Same mindset as facility-scale workflows
  - Deployment, reproducibility, isolation, observability

Summary

- Presented a **scalable Inference-as-a-Service** design
  - Triton-based, C++ custom backend, stateful ML pipelines
- Grounded in **real scientific workflows**
  - ATLAS jet calibration
- Emphasized **systems thinking**
  - design, deployment, reproducibility, performance

- **Looking ahead**
  - Applying this mindset to **facility-scale computing**
  - Exploring how such services could integrate with **Doudna-era workflows**

Thank you for your attention!

Triton Model Config

input and output shape

```
{
backend: "exatrkxgpu"

input [
  {
    name: "FEATURES"
    data_type: TYPE_FP32
    dims: [ -1, 3 ]
  }
]
output [
  {
    name: "LABELS"
    data_type: TYPE_INT64
    dims: [ -1 ]
  }
]
}
```

scalibility

```
max_batch_size: 0

instance_group [
  {
    count: 1
    kind: KIND_GPU
  }
]

parameters: {
  key: "EXECUTION_MODEL_PATH",
  value: {string_value: "/workspace/exatrkx_pipeline/datanm
}
```

ExaTrkX CPU vs GPU

CPU

```
Input file: ../../exatrkx_pipeline/datanmodels/in_e1000.csv
Models loaded successfully
Running Inference with local CPUs
Embedding model run successfully
is_trained = true
Total 39 tracks in 1 events.
1) embedding: 0.4282
2) building:  0.0166
3) filtering: 3.4804
4) gnn:       0.2146
5) labeling:  0.0023
6) total:     4.1421
```

GPU

```
Input file: ../../exatrkx_pipeline/datanmodels/in_e1000.csv
Running Inference with local GPUs
Total 37 tracks in 1 events.
1) embedding: 0.0398
2) building:  0.0015
3) filtering: 0.0088
4) gnn:       0.0141
5) labeling:  0.0016
6) total:     0.0659
```

```
# PU = 200 Events
Implementation    Exa.TrkX model inference time (s)
Direct CPU        9.65
Direct GPU        2.42
Exa.TrkX-aaS GPU  2.24
```