AthenaTriton: A Tool for running Machine Learning Inference as a Service in Athena

Yuan-Tang Chou[1], Beojan Stanislaus[2], Charles Leggett[2] ,Haoran Zhao[1], Julien Esseiva[2], Paolo Calafiura[2], Shih-Chieh Hsu[1],
Vakho Tsulaia[2], Xiangyang Ju[2]

on behalf of the ATLAS Computing Activity

[1]University of Washington
[2]Lawrence Berkeley National Lab

CHEP 2024
Oct 19, 2024

OAC-2117997

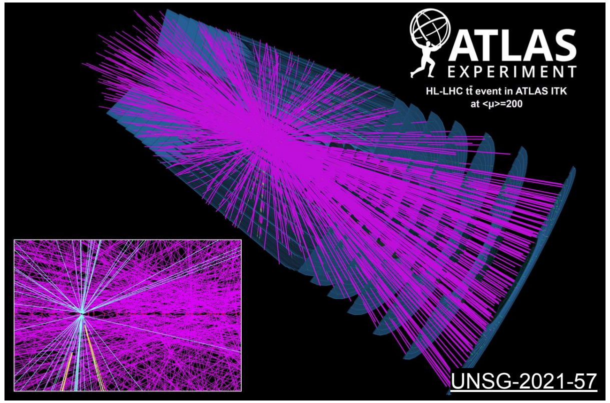
Introduction

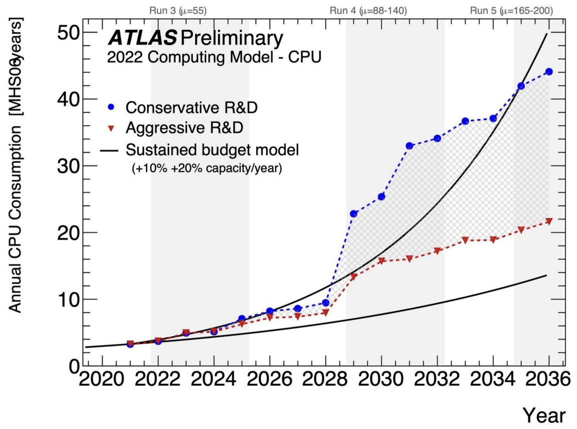
With the increase in luminosity at the upcoming HL-LHC, the growing demand for computing resources has become a critical topic, necessitating aggressive R&D efforts

- Track reconstruction accounted for ~ 64% of the total ATLAS event reconstruction CPU time in Run 2.

Besides, more and more machine learning algorithms are adopted in almost every aspect of the data analysis

- Detector simulation - see FastCaloGAN poster by Federico
- Physics analysis - see Neural Simulation-based Inference talk by Aishik
- GNN-based Tracking - see GNN4ITK Talks from Alina, Daniel, Corentin, Jay, and Heberth





ATLAS Software and Computing HL-LHC Roadmap

ML inference in ATLAS Athena framework

- We started from the in-house tools such as lwtnn and MVAUtils
  - supports scikit-learn, Keras, MVA.
- New development in Athena to create common interface for ONNXRuntime:
  - Creating a common ONNXRuntime environment,
  - Supporting ML models to run on different co-processors,
  - Simplifying / standardize ML inference.
  - See Xiangyang poster later this week

But this only solved part of the issue…

What we need but OnnxRuntime cannot do?

Known issues with OnnxRuntime:

- Certain ML operations are **not supported** by OnnxRuntime 😢
- Users simply **cannot convert** their ML model to Onnx → **show-stopper** 😭.
- OnnxRuntime inference values may vary in the last digits for different runs [issue].

What we need but ONNX cannot do

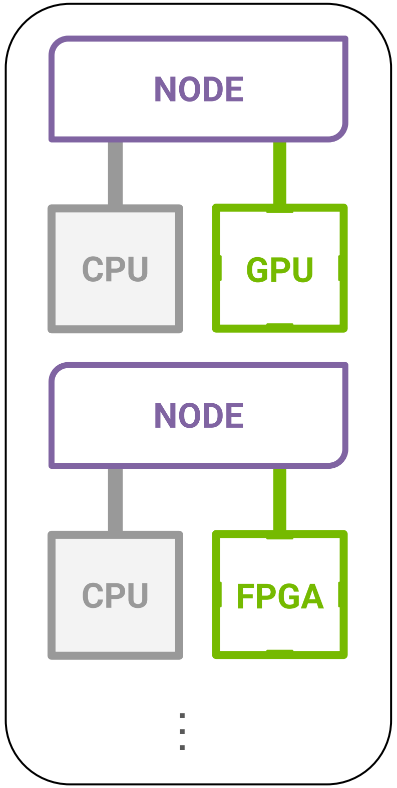
- Access to **remote co-processors** (GPUs, FPGAs, and so on)
- Support non-ML algorithms that can enjoy speedups by running in co-processors

What if we need to run new fancy state of the art algorithm not in Athena? 🤗

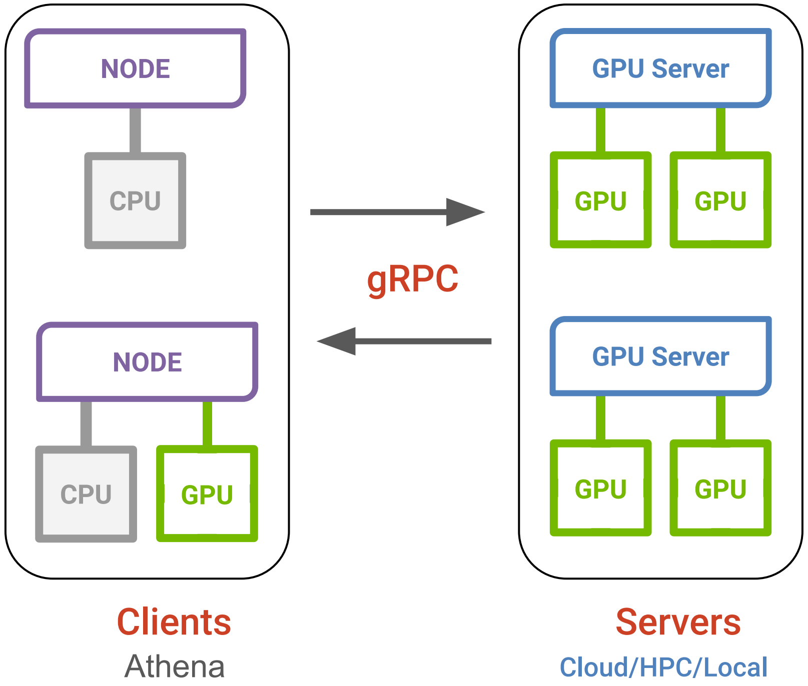
Heterogeneous Computing

- The most straightforward way to deploy algorithms on coprocessors is to run on machines with coprocessors
- However, **direct connection** can be inefficient and expensive at scale
- Need to matched the CPU-coprocessor ratio perfectly to fully utilized all the resource

“Direct connection”



Inference as a Service



Inference as-a-service provides a way to free Athena from:

- Implementing the algorithms
- Installing their dependencies
- Adapting to algorithm updates
- Supporting them to run on different platforms

But Athena has to:

- Install client dependencies (gRPC and Nvida Triton Client)
- Check if the requested model matches the expectation

“Direct Connection”

Pros:

- Many have working example

Cons:

- Can be an inefficient use of resources
- Expensive
- Machines without GPUs/FPGAs can’t benefit from coprocessors

“Inference as a Service”

Pros:

- **Factorized** out the underlying backend implementation
- More straightforward to integrate with the **production framework**
- Independent of the underlying **technology choices** and algorithms
- Better scalability and resource utilization (**Reduce cost**)

Cons:

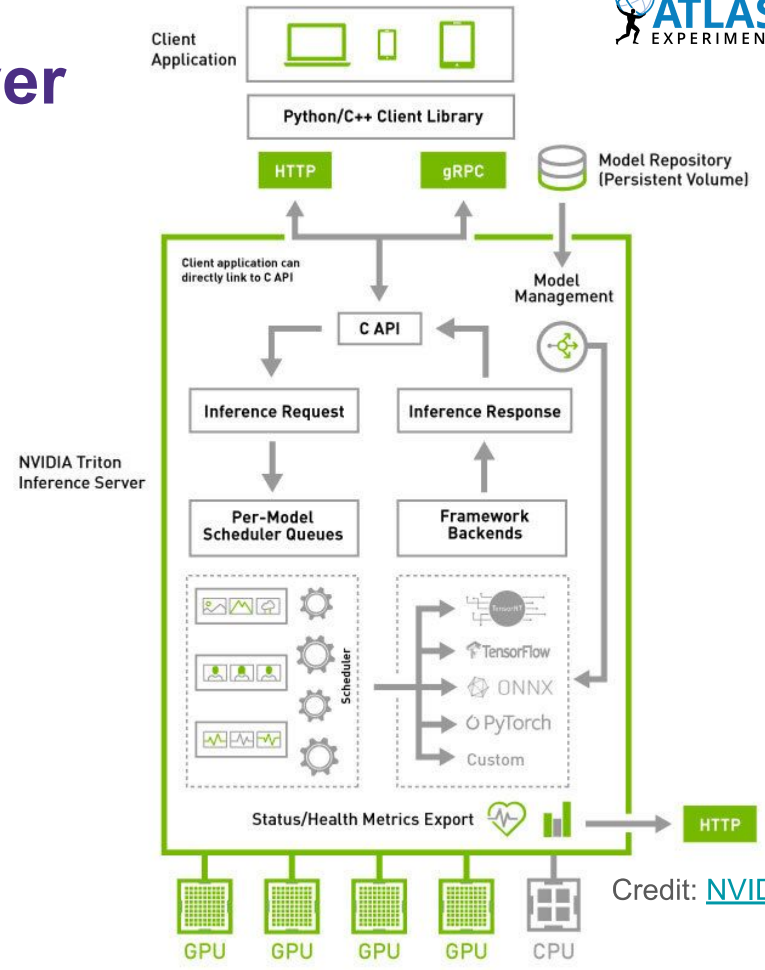
- Latency and network transmission overhead
- Adds complexity

Brief introduction to Triton Server

- It supports different backends.
- It provides backend API that allows adding custom backend.
- It supports Python-based backend.

  → attractive to testing ML development

- It features concurrent model execution, allowing multiple models and/or multiple instances of the same model to execute in parallel on the same system.

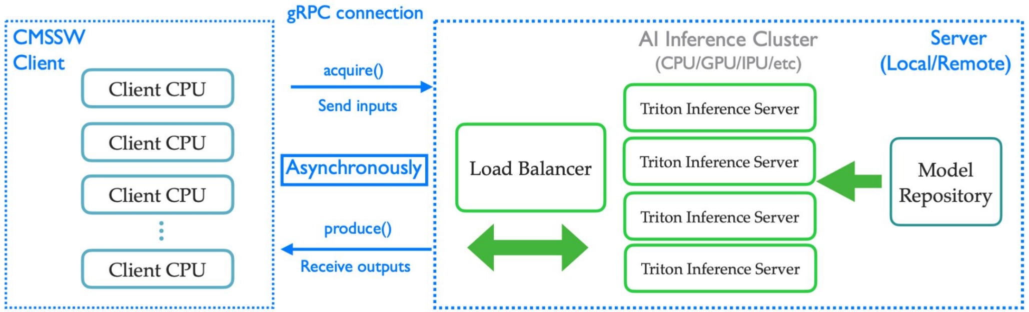

Client Application
Python/C++ Client Library
HTTP
gRPC
Model Repository (Persistent Volume)
Client application can directly link to C API
Model Management
C API
NVIDIA Triton Inference Server
Inference Request
Inference Response
Per-Model Scheduler Queues
Framework Backends
Scheduler
TensorFlow
ONNX
PyTorch
Custom
Status/Health Metrics Export
HTTP
GPU
GPU
GPU
GPU
CPU

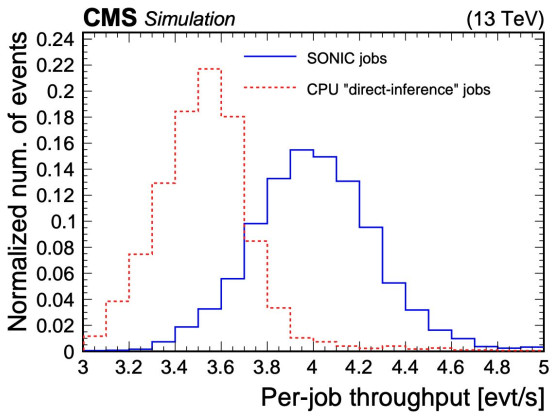
Credit: NVIDIA

Inference as a Service in other experiments

CMS-MLG-23-001

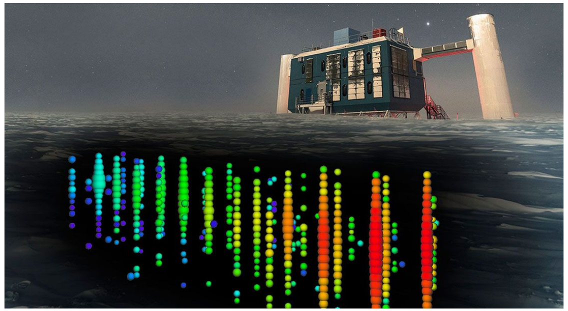
SONIC@CMS: Already implemented in CMSSW for Mini-AOD production



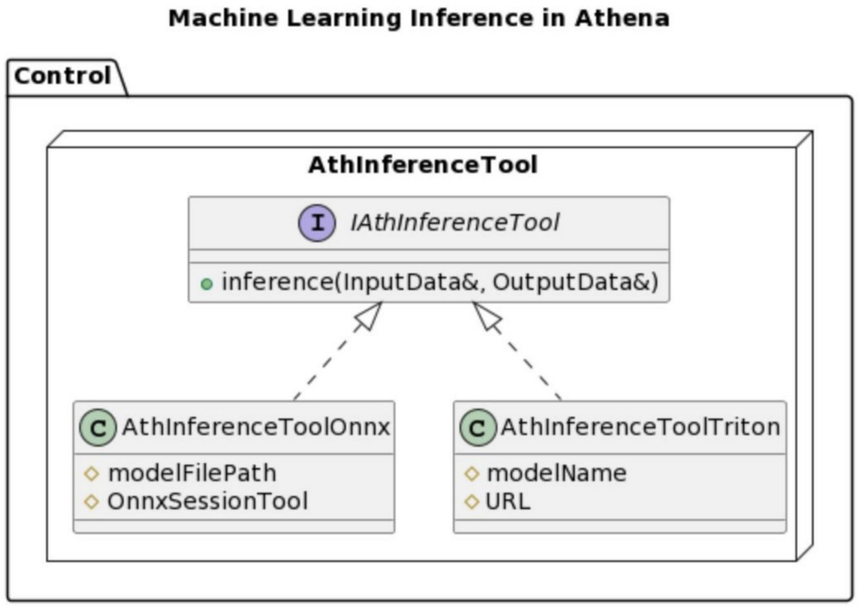


Off-loading **9%** of the total processing time on CPU to GPUs via SONIC, which gives a **13% increase in overall throughput** for CMS data processing- see SOINC poster by Kevin

Triton is also tested in **DUNE, LIGO, and IceCube (**IceSONIC Poster by Benedikt)



AthenaTriton

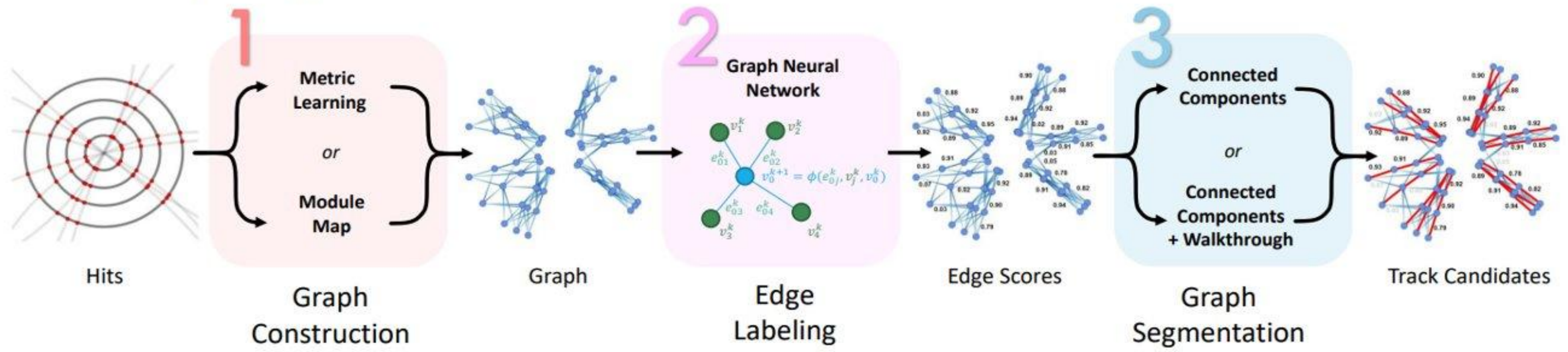


Powered by NVIDIA TRITON INFERENCE SERVER

- Input and output data are presented as a dictionary to enable dynamic number of inputs / outputs.
- Data type is std::variant, allowing different data types.
- Two concrete implementations: one with **ONNXruntime (for direct)** and another with **Nivida Triton (for as-a-service)**.
- Design to be able swapped through configuration easily.

Run GNN4ITk in Athena through AthenaTriton

We implemented the GNN-based track finding algorithm in Athena [1]

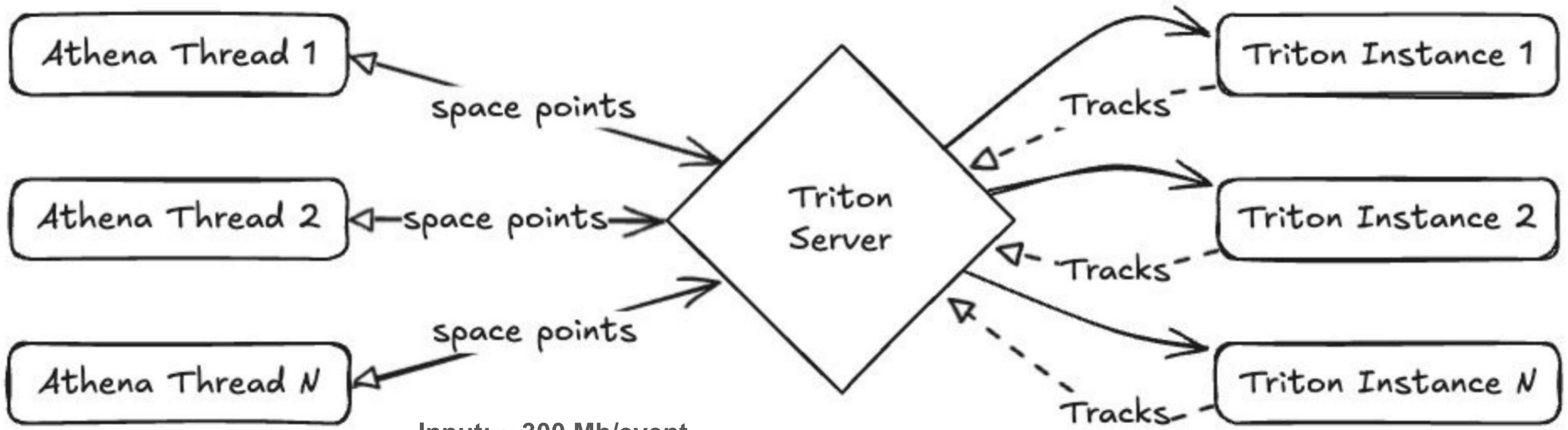


For more details, see talks from Alina and Daniel later this week!

[1] ATL-SOFT-PROC-2023-047

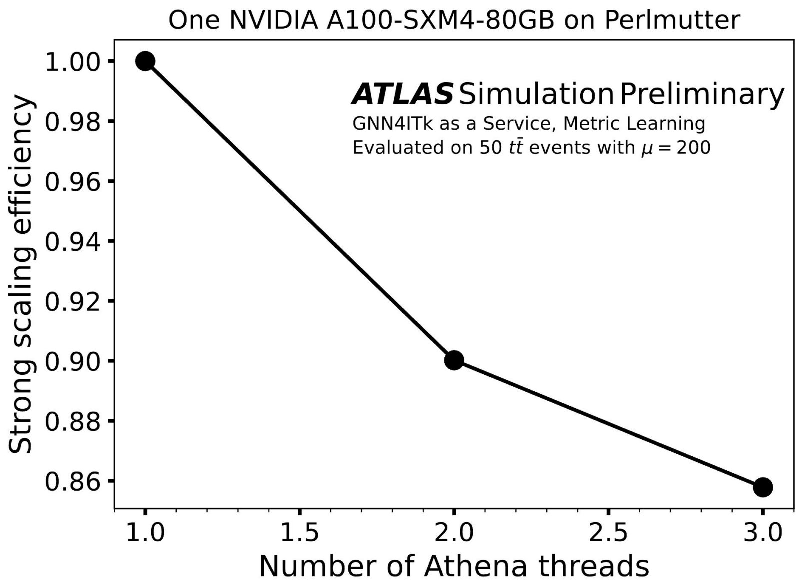
How does everything work together?

AthenaTriton will send space point information to the Triton Server and return a list of space point indexes, which connected and form reconstructed track candidates.



Input: ~ 300 Mb/event

Scaling performance



AthenaTrition offers a free speedup of **2.4x**

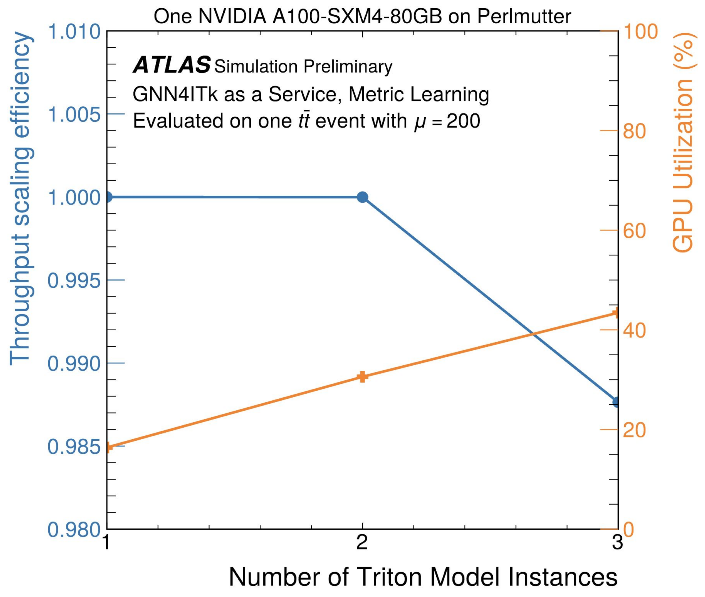
The number of Athena threads increases while the number of events remains constant (50 events).

Triton server runs on Nvidia A100 GPU with 80 GB memory. Create the number of model instances as the same as the Athena threads.

Good scaling efficiency up to 3 threads

The strong scaling efficiency $= \frac{t_1/n}{t_n}$, where t_n is the execution time for n Athena threads.

Standalone Throughput Scaling and GPU memory utilzation



Almost perfect throughput scaling efficiency and GPU utilization scales linearly with more instances.

It take ~18% of GPU memory per model instance on one NIVIDA A100

The throughput scaling efficiency is defined as $\epsilon = \frac{T_i/i}{T_1}$ where T_i is the throughput with i model instance.

Summary

- We are building infrastructure to integrate **Triton client** into Athena
- **AthenaTriton** will facilitate the usage of remote & local coprocessors.
- The GNN-based tracking Metric learning pipline are use as the first client to use the inference as a service in ATLAS.
- This could potential benefit **online reconstruction (High-level trigger)** and aslo **offline simulation and reconstruction.**

There are many developments that still need to be done, deployment, setting up service backend, load balancing…etc.