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

Yuan-Tang Chou[1,*], *Xiangyang* Ju[2,**], *Haoran* Zhao[1], *Attila* Krasznahorkay[3], *Johannes* Elmsheuser[4], *Debottam* Bakshi Gupta[5], *Beojan* Stanislaus[2], *Julien* Esseiva[2], *Vakhtang* Tsulaia[2], *Shih-Chieh* Hsu[1], and *Paolo* Calafiura[2] on behalf of the ATLAS Computing Activity.

[1]University of Washington
[2]Lawrence Berkeley National Lab
[3]University of Massachusetts, Amherst
[4]Brookhaven National Laboratory
[5]University of Texas at Arlington

Abstract. Machine learning (ML)-based algorithms play increasingly important roles in almost all aspects of the data analyses in ATLAS, including detector simulations, event reconstructions, and data analyses. These diverse ML models are being deployed in the ATLAS software framework, Athena. To harmonize the ML inference in both the Athena environment and the ROOT environment, a dual-use ML interface is defined and implemented with two distinct backends: the Open Neural Network Exchange (ONNX) Runtime and Inference as a Service. While ONNX Runtime serves as the primary inference backend in Athena, scalable inference strategies are required to address the growing demands of processing simulation and collision data, including maximizing event throughput and utilizing coprocessors like graphics processing units (GPUs). To meet this challenge, we introduce AthenaTriton, a solution that integrates the NVIDIA Triton Inference Server with Athena. In AthenaTriton, Athena operates as a Triton client that sends requests to a remote or local server that performs the model inference. This scalable approach can be used in both online and offline computing workflows.

ATL-SOFT-PROC-2025-026
10 February 2025

1 Introduction

With the increase in luminosity at the upcoming High-Luminosity LHC (HL-LHC) [1], the growing demand for computing resources has become a critical topic, necessitating aggressive R&D efforts [2]. To cope with the increased computing demand, the use of coprocessors, such as GPU and field-programmable gate array (FPGA), became potential options to explore for the upcoming ATLAS Phase-II Trigger and Data Acquisition (TDAQ) and software upgrades [3, 4]. This requires a software stack to be able to utilize coprocessors efficiently. Besides, more and more machine learning (ML) algorithms are adopted in almost every aspect of the simulations, reconstructions, and data analyses. This calls for a common inference tool

*e-mail: yuan-tang.chou@cern.ch
**e-mail: xju@lbl.gov

within the ATLAS reconstruction software framework, Athena [5], to reduce the duplication of the codebase.

This document presents an overview of existing ML inference implementation in ATLAS and details the design of a new common inference tool, AthInferenceTool, which can handle inference with local or remote GPUs. Section 2 described the detailed design of AthInferenceTool. Secion 2.3 gives an overview of the Inference as a Service (IaaS) computing model and tool development. Section 3 briefly summarized the graph neural network (GNN)-based tracking pipeline deployed with NVIDIA Triton [6]. Finally, the performance studies of the demonstrator are presented in section 4.

2 ML inference in ATLAS

Diverse ML models are being used in ATLAS for detector simulation, event reconstruction, and data analyses. These models are featured with different numbers of input (output) data and input (output) data types. For example, a GNN typically requires two or three tensors as inputs: node features of type `float`, edge lists of type `int64_t`, and, optionally, edge features of type `float`. In addition, data analyses would need to be supported in both the Athena environment and the ROOT [7] environment, necessitating a dual-use design for ML inference tools. [8].

To address these challenges, we designed a common ML interface in Athena: `IAthInferenceTool` [9], as shown in Fig. 1. The input and output data are represented as a dictionary that maps names to pairs of `(std::vector<int64_t>, std::variant)`. The first element describes the tensor shape, while the second stores the data. The dictionary-based representation supports a dynamic number of inputs and outputs, and the use of `std::variant` enables handling multiple data types.

2.1 ONNX Runtime

ONNX Runtime [10] is a high-performance inference engine designed to accelerate machine learning models across various platforms and hardware, supporting the ONNX (Open Neural Network Exchange) format for interoperability. In ONNX Runtime, at least one `ORT::Env` object is required to manage the logging, thread pools, and memory allocations for each `ORT::Sessions`. Each `ORT::Sessions` handle the execution of a ML model. Additionally, `ORT::Sessions` supports different execution providers (EP)—platforms where the model inference tasks are performed—including NVIDIA CUDA [11] and TensorRT [12], Intel oneDNN [13], and AMD Vitis AI [14].

The integration of ONNX Runtime in Athena is shown in Fig. 1. We developed an Athena service `IOnnxRuntimeSvc` [15] that returns a pointer to the `ORT::Env` object. The Athena service is designed to have only one instance within Athena, which is shared across all Athena algorithms. Currently, the service provides a common environment primarily for logging. In the future, it could be extended to support global/shared thread pools and shared memory allocator(s) when numerous ML models are executed within a single Athena job. Additionally, we implemented a lightweight Athena Tool interface `IOnnxRuntimeSessionTool` [16] that returns a pointer to the `ORT::Sessions` object. This interface supports multiple execution providers (EPs), allowing users to switch between EPs via the Athena Python configuration [17] configuration without recompilation. Currently, only the CPU and CUDA EPs are implemented. Note that each `IOnnxRuntimeSessionTool` corresponds to one specific ML model stored in the ONNX file format.

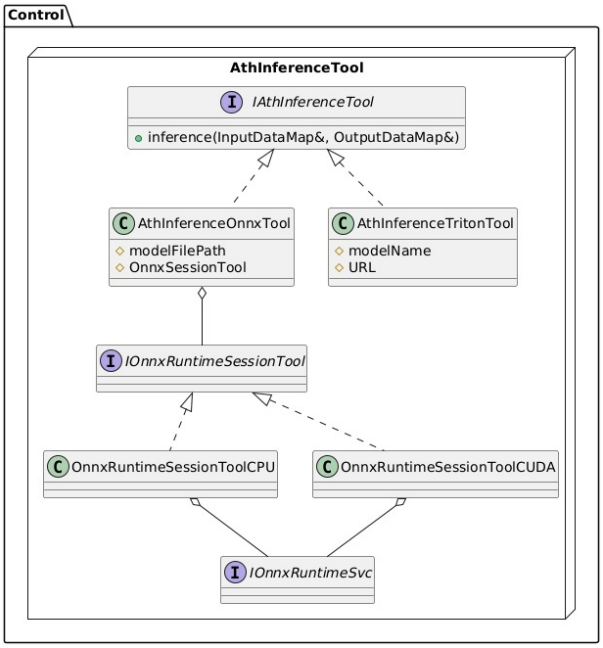

Figure 1. Illustration of the common ML interface for both ONNX Runtime for direct inference and IaaS using NVIDIA Triton.

2.2 AthenaTriton

AthenaTriton refers to Athena jobs that employ the Triton Inference Server [18], i.e., the IaaS approach. In AthenaTriton, Athena acts as a Triton client that sends requests to a remote or local Triton server that performs the ML model inference. To that end, we developed the `AthInferenceTritonTool` that uses Triton to implement the common ML interface `IAthInferenceTool`.

The `AthInferenceTritonTool` is lightweight and requires minimal dependencies. The tool takes only two inputs: `modelName` for requesting the model deployed at the server and Uniform Resource Locator (URL) for connecting to the Triton server. It does the following internally to facilitate the IaaS: 1) check the server status and connect to the server, 2) convert the input data to raw data format of type `uint8_t`, 3) send the input data to the server for ML inference, 4) check the returned result status and convert the result to a user-defined format if no error occurs. Similar to the ONNX Runtime implementation, each Triton tool corresponds to one specific ML model, even though the Triton server might host more than one model.

Clients can communicate with the Triton server using either an HTTP/REST [19] protocol, a gRPC protocol [20], or by an in-process C API or its C++ wrapper. By default, AthenaTriton utilizes the gRPC protocol.

2.3 Triton Inference Server

Triton Inference Server is an open-source package developed by NVIDIA [18]. The framework provides a client and server interface to allow deployment and streamline inference requests. A backend is an implementation that executes a model. Triton natively supports

multiple common ML backends such as TensorRT, ONNX Runtime [10], Tensorflow [21], and PyTorch [22]. It also supports custom backends using Python and C++, enabling greater control over the data allocations and complex workflow. This flexibility promotes rapid testing and development of sophisticated ML models, like the Graph Neural Network for particle tracking [23].

Compared to direct GPU connection, Triton allows concurrent execution of multiple models or instances of a single model on the server, improving scalability and resource utilization while reducing operational costs. The number of model instances can be adjusted through a configuration file, providing flexibility to optimize the workloads. Additionally, Triton eliminates the need for the existing production frameworks to implement complex algorithms or manage their dependencies, thereby improving adaptability and enhancing long-term maintainability.

Additional complexities, such as network latency and overhead, may arise, potentially increasing overall deployment costs and system complexity. Triton provides a convenient tool called `perf_analyzer` [24] tool to measure the performance of models deployed on the server by sending augmented real or synthesized data. The `perf_analyzer` provides detailed computing metrics, including response times, latency, throughput rates, and GPU utilization, helping developers identify potential bottlenecks and to optimize their deployment for better efficiency. The `perf_analyzer` is utilized in the standalone study.

Exploring the deployment of Triton Inference Servers within ATLAS computing resources, including the Event Filter computing farm, involves addressing challenges such as resource allocation, latency optimization, and integration with existing systems. It is an ongoing area of investigation. Despite these concerns, the Triton Inference Server remains an optimal solution for ML pipelines due to its robust scalability, straightforward integration capabilities, and support for diverse backends.

3 Graph Neural Network-based track finding algorithm

As described in Section 2.3, Triton provides a simple way to write a customized backend. We use the GNN-based metric learning track-finding algorithm to demonstrate the capability of AthenaTriton. The pipeline consists of three sequential stages: graph construction, edge labeling using GNN, and finally, edge segmentation with connected components and the walkthrough algorithm. Despite its complex internal logic, the pipeline uses a simple interface. It takes a list of space point features as inputs and outputs a list of proto-track candidates. Unlike classical algorithms such as the Kalman Filter, it does not rely on additional information about the detector geometry, detector material, or magnetic field descriptions. Further details can be found in Ref [25].

In our study, the GNN tracking uses the Metric Learning-based graph construction and was trained with ACORN [26]. The data consists of simulated $t\bar{t}$ events with an average number of 200 interactions per bunch crossing (i.e., pileup (μ) = 200), using the ATLAS upgrade tracker ITk [25]. The GNN-based tracking pipeline is referred to as GNN4ITk, and its service version is GNN4ITk as a Service.

The GNN4ITk as a Service uses the Python backend in Triton to execute the pipeline; see the software implementation in Ref. [27]. Note that the ML models were converted to TorchScript, an intermediate representation of a PyTorch model that can then be run in a high-performance environment [28].

An Athena tool is developed for preparing the input spacepoint features, calling the Triton Tool for ML inference, and returning proto-track candidates, which is a list of spacepoint indices separated by a special indice, -1, indicating the end of a track. The averaged size of the input spacepoint features from a simulated $t\bar{t}$ event with $\mu = 200$ is around 300 Mb.

4 Performance of the GNN4ITk as a Service

A complete particle tracking chain typically consists of several steps: space point formation, track finding, track fitting, and ambiguity resolution. GNN4ITk solely does the track-finding step within this chain.

To evaluate the main characteristics of the IaaS approach, two studies are conducted: (1) the standalone performance of the GNN4ITk as a Service, and (2) the end-to-end AthenaTriton inference with multiple client threads sending requests to a remote GPU. The standalone test isolates the GNN4ITk from the full chain, while the end-to-end study evaluates its integration within the complete chain.

4.1 Standalone Scaling

The standalone study uses the tool `perf_analyzer` to measure the throughput (events per second) and the GPU utilization by sending the same $t\bar{t}$ event with $\mu = 200$ to the Triton server repeatedly. The results are summarized in Figure 2. The throughput scaling efficiency is defined as $\epsilon = \frac{T_i/i}{T_1}$ where T_i is the throughput with i model instance(s). The GNN4ITk service runs on a GPU server hosted at the Perlmutter GPU node with NVIDIA A100 GPU with 80 GB memory. The scaling efficiency stays above 98% with more concurrent Triton instances running on the same GPU. The GPU utilization increases up to around 45% before reaching the memory limit when the model instance increases. Further studies are ongoing to improve the GNN4ITk pipeline and reduce its memory consumption.

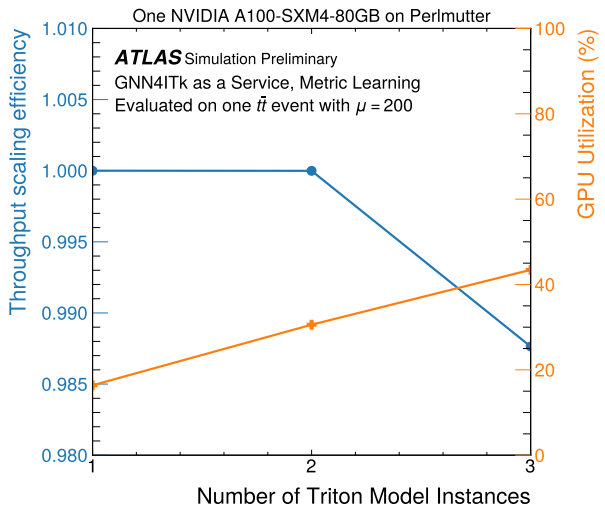

Figure 2. The event processing throughput scaling efficiency (blue line, left y-axis) and the GPU utilization (yellow line, right y-axis) as a function of number of Triton model instances running on a single GPU.

4.2 End-to-End scaling with AthenaTriton

The full end-to-end performance gain is evaluated by measuring the execution time of processing a fixed number of events with Athena. This takes into account the additional overhead

between client-to-server communication. In this study, the server is at a Perlmuter GPU computing node with A100, while the Athena client is at a Perlmuter CPU node. The overhead from data transmission between the GPU and CPU node is small.

The performance is evaluated by the strong scaling efficiency, ϵ, defined as $\epsilon = \frac{t_1/n}{t_n}$, where t_1 is the execution time for a single Athena thread, t_n for the execution time for n Athena threads, and n ranges from 1 to 3. The execution time t_n is measured as the wall time for running the tracking reconstructions on $t\bar{t}$ events with a pileup of $\mu = 200$ in Athena on a Perlmutter CPU node. The GNN4ITk service is the same as in the standalone scaling test, with the number of model instances on a single GPU adjusted to match the number of Athena threads. Beyond three model instances, the server encountered an out-of-memory error. This configuration fully utilizes the A100 GPU's computing capabilities to handle multiple concurrent Athena requests, enabling AthenaTriton to achieve a 2.4 speedup compared with the baseline direct inference.

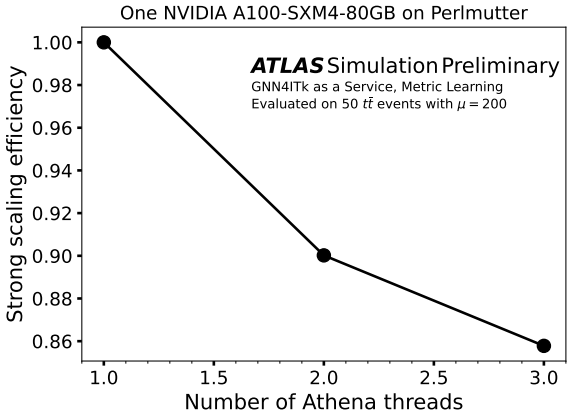

Figure 3. The strong scaling efficiency as a function of the number of Athena threads for GNN-based tracking as a Service.

5 Conclusions

The development of a common ML inference interface in Athena represents a crucial step in core software development for the future HL-LHC. The `IAthInferenceTool` provides a unified interface for all ML model inferences in Athena, enabling consistent and flexible integration. Two Athena tools are developed: one with ONNX Runtime and another with the Inference as a service computing model. The AthenaTriton extends this capability by supporting remote coprocessors via Triton.

We demonstrated that the IaaS can be a viable solution to improving GPU utilization, as exemplified by the GNN-based Metric learning tracking pipeline. This remains an active area of development within ATLAS, with ongoing efforts to migrate the ML inference infrastructure to the common interface. For AthenaTriton, future work will focus on optimizing the server deployments on high-performance computing (HPC) environments and Analysis User Facilities, and designing load-balancing mechanisms to support multiple GPUs efficiently.

Acknowledgments

Y. Chou, S. Hsu, and H. Zhao are supported by National Science Foundation (NSF) grant No. PHY-2117997. This research used resources of the National Energy Research Scientific Computing Center (NERSC), a U.S. Department of Energy Office of Science User Facility located at Lawrence Berkeley National Laboratory, operated under Contract No. DE-AC02-05CH11231.

References

[1] O. Aberle, I. Béjar Alonso, O. Brüning, P. Fessia, L. Rossi, L. Tavian, M. Zerlauth, C. Adorisio, A. Adraktas, M. Ady et al., High-Luminosity Large Hadron Collider (HL-LHC): Technical design report, CERN Yellow Reports: Monographs (CERN, Geneva, 2020), https://cds.cern.ch/record/2749422

[2] ATLAS Collaboration, ATLAS Software and Computing HL-LHC Roadmap (2022), https://cds.cern.ch/record/2802918

[3] ATLAS Collaboration, ATLAS TDAQ Phase-II Upgrade: Technical Design Report (2017), ATLAS-TDR-029; CERN-LHCC-2017-020, https://cds.cern.ch/record/2285584

[4] ATLAS Collaboration, The Phase-II Upgrade of the ATLAS Trigger and Data Acquisition System - Event Filter Tracking Amendment: Technical Design Report (2022), CERN-LHCC-2022-004, ATLAS-TDR-029-ADD-1, https://cds.cern.ch/record/2802799

[5] ATLAS Collaboration, Software and computing for Run 3 of the ATLAS experiment at the LHC (2024), 2404.06335.

[6] NVIDIA Corporation, Triton Inference Server: An Optimized Cloud and Edge Inferencing Solution., https://github.com/triton-inference-server/server, accessed: 2025-01-23

[7] R. Brun, F. Rademakers, P. Canal, A. Naumann, O. Couet, L. Moneta, V. Vassilev, S. Linev, D. Piparo, G. GANIS et al., root-project/root: v6.18/02 (2019), https://doi.org/10.5281/zenodo.3895860

[8] D. Adams, P. Calafiura, P.A. Delsart, M. Elsing, S. Farrell, K. Koeneke, A. Kraszna-223 horkay, N. Krumnack, E. Lancon, W. Lavrijsen et al. on behalf of the ATLAS Collaboration, Dual-use tools and systematics-aware analysis workflows in the atlas run-2 analysis model, Journal of Physics: Conference Series **664**, 032007 (2015). 10.1088/1742-6596/664/3/032007

[9] ATLAS Collaboration, IAthInferenceTool, https://gitlab.cern.ch/atlas/athena/-/blob/main/Control/AthOnnx/AthOnnxInterfaces/AthOnnxInterfaces/IAthInferenceTool.h, accessed: 2025-01-24

[10] ONNX, Open Neural Network Exchange (ONNX), https://github.com/onnx/onnx, accessed: 2023-11-08

[11] J. Nickolls, I. Buck, M. Garland, K. Skadron, Scalable parallel programming with CUDA, in *ACM SIGGRAPH 2008 Classes* (Association for Computing Machinery, New York, NY, USA, 2008), SIGGRAPH '08, ISBN 9781450378451, https://doi.org/10.1145/1401132.1401152

[12] NVIDIA, NVIDIA TensorRT, https://developer.nvidia.com/tensorrt, accessed: 2023-11-08

[13] oneDNN Contributors, oneAPI Deep Neural Network Library (oneDNN), https://github.com/oneapi-src/oneDNN, accessed: 2024-12-15

[14] Xilinx, AMD Vitis™ AI, https://github.com/Xilinx/Vitis-AI, accessed: 2025-01-23

[15] ATLAS Collaboration, IOnnxRuntimeSvc, https://gitlab.cern.ch/atlas/athena/-/blob/main/Control/AthOnnx/AthOnnxInterfaces/AthOnnxInterfaces/IOnnxRuntimeSvc.h, accessed: 2025-01-24

[16] ATLAS Collaboration, IOnnxRuntimeSessionTool, https://gitlab.cern.ch/atlas/athena/-/blob/main/Control/AthOnnx/AthOnnxInterfaces/AthOnnxInterfaces/IOnnxRuntimeSessionTool.h, accessed: 2025-01-24

[17] W. Lampl on behalf of the ATLAS Collaboration, A new approach for atlas athena job configuration, EPJ Web Conf. **214**, 05015 (2019). 10.1051/epjconf/201921405015

[18] NVIDIA, NVIDIA Triton Inference Server, https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/index.html, accessed: 2024-05-15

[19] R.T. Fielding, Ph.D. thesis (2000), copyright - Database copyright ProQuest LLC; ProQuest does not claim copyright in the individual underlying works; Last updated - 2023-02-24, https://www.proquest.com/dissertations-theses/architectural-styles-design-network-based/docview/304591392/se-2

[20] gRPC Authors, Remote procedure call, https://grpc.io/, accessed: 2024-12-15

[21] M. Abadi et al., TensorFlow: A system for large-scale machine learning (2016), 1605.08695

[22] A. Paszke, S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga et al., Pytorch: An imperative style, high-performance deep learning library (2019), 1912.01703, https://arxiv.org/abs/1912.01703

[23] J.D. Burleson, S. Caillou, P. Calafiura, J. Chan, C. Collard, X. Ju, D.T. Murnane, M. Neubauer, M.T. Pham, C. Rougier et al. on behalf of the ATLAS Collaboration, Physics Performance of the ATLAS GNN4ITk Track Reconstruction Chain, ATL-SOFT-PROC-2023-047, Geneva (2023), https://cds.cern.ch/record/2882507

[24] NVIDIA, Perf analyzer, https://docs.nvidia.com/deeplearning/triton-inference-server/archives/triton-inference-server-2310/user-guide/docs/user_guide/perf_analyzer.html, accessed: 2025-01-24

[25] ATLAS Collaboration, Expected Tracking Performance of the ATLAS Inner Tracker at the HL-LHC (2019), ATL-PHYS-PUB-2019-014, https://cds.cern.ch/record/2669540

[26] GNN4ITk Team, A geometrical-based tracking reconstruction network, https://gitlab.cern.ch/gnn4itkteam/acorn, accessed: 2025-01-24

[27] X. Ju, Tracking as a service, https://github.com/xju2/tracking-as-a-service/releases/tag/v1.0, accessed: 2024-12-15

[28] The PyTorch Foundation, Torchscript, https://pytorch.org/tutorials/beginner/Intro_to_TorchScript_tutorial.html, accessed: 2024-12-15