Journal of
Instrumentation

PAPER • OPEN ACCESS

Track reconstruction as a service for collider physics

To cite this article: Haoran Zhao *et al* 2025 *JINST* **20** P06002

Published by IOP Publishing for Sissa Medialab

Received: *January 21, 2025*
Revised: *March 12, 2025*
Accepted: *April 2, 2025*
Published: *June 6, 2025*

Track reconstruction as a service for collider physics

Haoran Zhao,[a] **Yuan-Tang Chou**,[a,*] **Yao Yao**,[b] **Xiangyang Ju**,[c] **Yongbin Feng**,[d] **William Patrick McCormack**,[e] **Miles Cochran-Branson**,[a] **Jan-Frederik Schulte**,[b] **Miaoyuan Liu**,[b,*] **Javier Duarte**,[f] **Philip Harris**,[e] **Shih-Chieh Hsu**,[a] **Kevin Pedro**[g] **and Nhan Tran**[g]

[a] *Department of Physics, University of Washington, Seattle, WA 98195, U.S.A.*
[b] *Department of Physics and Astronomy, Purdue University, West Lafayette, IN 47907, U.S.A.*
[c] *Physics Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, U.S.A.*
[d] *Department of Physics and Astronomy, Texas Tech University, Lubbock, TX 79409, U.S.A.*
[e] *Department of Physics, Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.*
[f] *Department of Physics, University of California San Diego, La Jolla, CA 92093, U.S.A.*
[g] *Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.*

E-mail: yuan-tang.chou@cern.ch, liu3173@purdue.edu

Abstract: Optimizing charged-particle track reconstruction algorithms is crucial for efficient event reconstruction in Large Hadron Collider (LHC) experiments due to their significant computational demands. Existing track reconstruction algorithms have been adapted to run on massively parallel coprocessors, such as graphics processing units (GPUs), to reduce processing time. Nevertheless, challenges remain in fully harnessing the computational capacity of coprocessors in a scalable and non-disruptive manner. This paper proposes an inference-as-a-service approach for particle tracking in high energy physics experiments. To evaluate the efficacy of this approach, two distinct tracking algorithms are tested: Patatrack, a rule-based algorithm, and Exa.TrkX, a machine learning-based algorithm. The as-a-service implementations show enhanced GPU utilization and can process requests from multiple CPU cores concurrently without increasing per-request latency. The impact of data transfer is minimal and insignificant compared to running on local coprocessors. This approach greatly improves the computational efficiency of charged particle tracking, providing a solution to the computing challenges anticipated in the High-Luminosity LHC era.

Keywords: Computing (architecture, farms, GRID for recording, storage, archiving, and distribution of data); Data processing methods; Online farms and online filtering; Software architectures (event data models, frameworks and databases)

ArXiv ePrint: 2501.05520

*Corresponding author.

https://doi.org/10.1088/1748-0221/20/06/P06002

Contents

1 Introduction

The computing demands of particle physics experiments at the CERN Large Hadron Collider (LHC) [1], such as ATLAS [2] and CMS [3], are expected to increase dramatically in the era of the High-Luminosity LHC (HL-LHC). This anticipated increase is primarily due to the higher luminosity at the HL-LHC, which will lead to more simultaneous proton-proton interactions — known as pileup — in each collision, resulting in a larger number of particles that need to be processed. Consequently, significant efforts are being made to accelerate existing workflows, including event reconstruction, which aims to deduce the properties of particles produced in collisions based on detector measurements.

In collider and fixed target experiments, tracking detectors placed close to the beam collision area and immersed in a strong magnetic field provide high-precision position measurements from which the trajectories of charged particles can be determined. This task is known as charged particle tracking [4]. Among all event reconstruction algorithms, tracking is typically the most time-consuming component, accounting for 45% or more of the total computing time during data processing [5, 6]. By optimizing this critical component, the entire data processing pipeline can achieve faster throughput, enabling more timely analysis and interpretation of the vast amounts of data recorded by the detectors.

Track reconstruction algorithms primarily involve pattern recognition operations to identify the paths of charged particles as they interact with the detector in three-dimensional space. These operations can be significantly accelerated with coprocessors, such as graphics processing units (GPUs), which are well-suited for parallel computing tasks. Integrating these coprocessors with modern CPUs — a method known as heterogeneous computing — enables the system to handle heavy computational loads more efficiently by distributing tasks between CPUs and coprocessors.

2025 JINST 20 P06002

To address the challenges encountered when implementing heterogeneous computing for data-intensive physics experiments, an *as-a-service* approach has been developed. The current computing infrastructure aggregates tasks from central experimental operations and individual user requests into a global computing grid, distributing tasks to available computing centers worldwide. However, the traditional technique of directly connecting coprocessors to CPUs in each computing center faces several challenges, such as coprocessor underutilization or overutilization, inconsistent availability of coprocessors across sites, and the need to maintain software compatibility between the varying types of coprocessors and CPUs at each computing site.

The as-a-service approach overcomes these challenges by adding an abstraction layer that enables the dynamic allocation of CPU and GPU resources tailored to specific tasks, as illustrated in figure 1. This method provides several key benefits: it helps to achieve optimal GPU utilization, facilitates remote access to GPU resources, eliminates the need for local GPUs, decouples the server from clients, modularizes software support for CPU and GPU. Consequently, server-side coprocessor configuration changes require minimal modification on the client side, simplifying technical support and software compilation processes. This approach has been explored in various technologies including field-programmable gate arrays (FPGAs) [7, 8], GPUs [9, 10], Intelligence Processing Units (IPUs), and CPUs [11], and has undergone extensive testing in numerous experiments such as CMS [11], ProtoDUNE [12, 13], and LIGO [14].

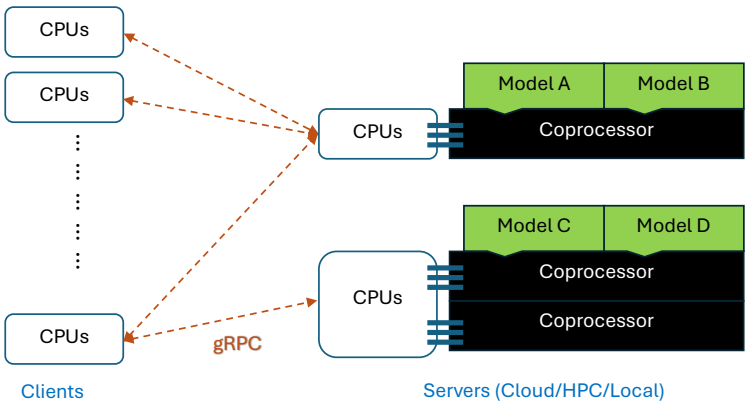

Figure 1. Inference *as-a-service* approach: users send various inference requests from client CPUs, which include details about the type of inference desired, input dimensions and content, and output dimensions and labels. This information is delivered from the clients to the servers through gRPC protocol, a high-performance Remote Procedure Call. The server CPUs receive these tasks, batch them, execute inference on the appropriate coprocessor based on the specific request, and deliver the output back to the client CPUs via gRPC protocol. In this approach, each server can contain a different number of coprocessors and provide different models. Each client can deliver tasks to multiple servers so that the tasks can be processed in parallel. The client-to-server ratio can be scaled based on the demand of client requests.

To demonstrate the effectiveness of the as-a-service approach in LHC experiments, we focus on two track reconstruction algorithms: Patatrack [15, 16], a rule-based algorithm implemented using Compute Unified Device Architecture (CUDA), and Exa.TrkX [17], a machine learning algorithm based on graph neural network (GNN). This paper presents the improvements in throughput achieved

2025 JINST 20 P06002

for both algorithms by implementing customized server backends that adapt them to the as-a-service model, showcasing a scalable solution to the computational challenges anticipated in the HL-LHC era.

In section 2, we review the typical pipeline of track reconstruction algorithms in particle physics, providing a detailed description of both Patatrack and Exa.TrkX with particular emphasis on their input and output structures. Sections 3 and 4 discuss the implementation of the as-a-service approach for each algorithm, related performance metrics, including throughput and GPU utilization. Finally, we discuss the differences in implementation and the challenges encountered during deployment.

2 Background

2.1 HEP Computing: online and offline reconstruction

In typical high-energy physics experiments like ATLAS and CMS at the LHC, the reconstruction processes are divided into online and offline reconstruction. *Online reconstruction* refers to the trigger system that rapidly reconstructs and filters the data. It aims to identify potentially interesting events to record for further analysis [18–22]. The trigger chain is divided into multiple levels. The level-1 trigger (L1T), implemented in hardware like application-specific integrated circuits (ASICs) and FPGAs, provides the first rapid decision-making layer. It uses a simplified reconstruction of signals from a subset of detector systems to reduce the data rate from 40 MHz to around 750–1000 kHz by selecting events with high-energy particles or specific decay products [18, 22].

Following the L1T, the high-level trigger (HLT) system, which operates in software using commercial CPU or GPUs, applies a more refined selection to reduce the event rate to 5–10 kHz. It uses more detailed information and complex algorithms, similar to offline reconstruction, to filter the surviving events by applying additional criteria. Latency and throughput are typically the limiting factors at this stage.

After the online selection, events are written to permanent storage, where *offline reconstruction* occurs. This stage involves using more sophisticated algorithms to fully reconstruct and calibrate the recorded events. Unlike online reconstruction, offline reconstruction is not constrained by real-time requirements, allowing for the application of precise calibration, alignment corrections, and detailed analysis procedures. This includes fitting charged tracks, identifying jets, and accurately reconstructing decay vertices, etc. Nevertheless, the challenge remains in how to process the vast amount of data efficiently and how to seamlessly integrate new coprocessors into the production framework as the hardware landscape rapidly evolves.

2.2 Track reconstruction

When a charged particle passes through a tracking detector, it leaves behind charge deposits in each detector layer as it traverses the materials. The resulting signals are read out by the detector electronics and then, if initially analog, converted to digital. The primary goal of track reconstruction algorithms is to determine the trajectories of charged particles, including their curvature and point of origin (vertex). The magnetic field applied within the detector causes the trajectory of a charged particle to follow a curved path, with the curvature inversely related to the particle's momentum; tighter curves indicate lower momentum. Accurate vertex determination is crucial for identifying the specific collision event that produced the track, especially in environments with high pileup, such as the HL-LHC.

2025 JINST 20 P06002

Typical tracking algorithms involve several stages: spacepoint formation, track seeding, track following, and track fitting. Initially, 2D or 3D measurement positions, known as *spacepoints* or hits, are estimated by combining nearby raw detector measurements into clusters and determining their coordinates. Track seeding then uses these spacepoints to form initial track candidates, providing preliminary estimates of trajectory parameters such as direction, origin, and curvature. Track following refines these seeds by adding more spacepoints along the projected path, ultimately leading to the track fitting stage. Here, an initial trajectory is calculated through the spacepoints, enabling the estimation of the particle's physical and kinematic properties, including charge, momentum, and origin. However, most traditional fitting algorithms, such as the Kalman filter, are generally not optimized to run on GPU, limiting how they can scale to handle increasing computational demands.

With the rapid development of machine learning (ML), another class of algorithms using geometric deep learning methods (GDL) has emerged [23]. Several architectures exist to utilize 3D point clouds for pattern recognition [17, 24–27]. These ML-based algorithms learn to cluster or connect spacepoints from the same tracks based on high-dimensional latent space features, leading to improvements in reconstruction speed and accuracy in complex environments with high particle multiplicity. These algorithms are also more parallelizable by nature.

We demonstrate the feasibility of *tracking as a service* with Patatrack, a GPU-optimized rule-based approach described in section 2.2.1 and the Exa.TrkX pipeline, a ML-based pattern recognition algorithm detailed in section 2.2.2.

2.2.1 Patatrack

Patatrack [15, 16] is a GPU-optimized track reconstruction algorithm designed for the CMS HLT [11] that employs a cellular automaton for pattern recognition. Unlike traditional tracking algorithms, Patatrack is explicitly tailored for execution on GPU, enabling efficient parallel data processing. The inputs to the algorithm are raw data from the pixel detectors and information about the beam spot — the region where the proton beams overlap during collisions. The algorithm comprises five major sub-algorithms: digitization, hit reconstruction, ntuplet creation, pixel tracks, and vertex reconstruction. The outputs of Patatrack are reconstructed pixel tracks, vertices, and other intermediate objects necessary for downstream tasks such as muon reconstruction.

The workflow of the Patatrack algorithm running on a GPU with the as-a-service approach is illustrated in figure 2. To minimize data movement and conversion, intermediate objects are retained on the device, while objects needed for downstream tasks are transferred back to the host. The input size is approximately 80 kB per event, and the outputs are around 2 MB per event.

2.2.2 Exa.TrkX pipeline

The Exa.TrkX algorithm is based on deep learning models. It uses spacepoints as inputs and produces track candidates as outputs. Each track candidate is a list of spacepoints. The pipeline contains three major modules: graph construction, edge classification, and graph segmentation. The graph construction module uses multilayer perceptrons (MLPs) to encode spacepoint raw features to a new latent space. The hits from the same charged particles are close to each other and away from other hits from different charged particles. This step is called *embedding*. Then, a fixed-radius algorithm builds connections between spacepoints in this latent space (*edges*). The number of edges after the embedding step necessitates another MLP to filter out clearly fake edges — called *filtering*. A fixed

2025 JINST 20 P06002

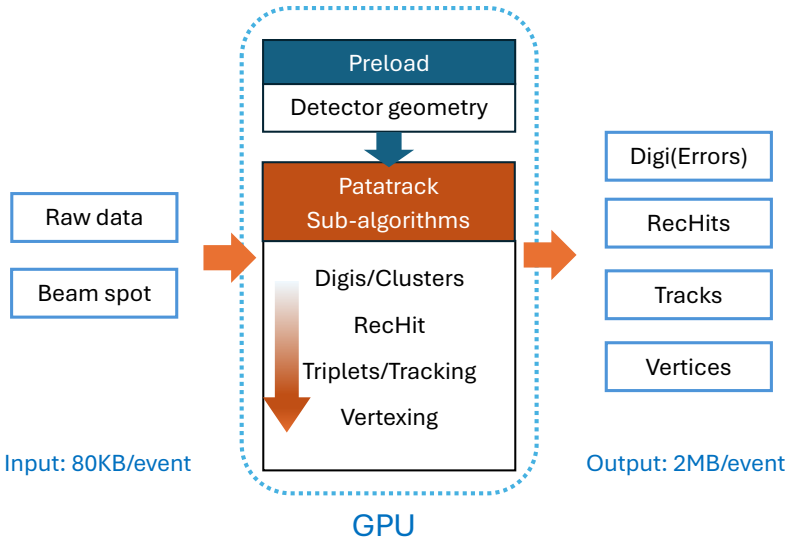

Figure 2. This illustration shows Patatrack running on a GPU using the as-a-service approach. Common detector-related information, such as the location of each detector layer and detector indices, is preloaded onto the GPU to be used later in the processing. For each event, raw data and beam spot information are delivered to the GPU. The raw data is compressed before delivery, and during the digitization step, it is unpacked and converted back into "digis." In the same detector layer, neighboring digis are clustered to determine the location of a "hit," representing a single interaction of a charged particle with that detector layer. From the hits, those in adjacent layers are paired to form doublets, then pair to triplets as track seeds. From a seed, a group of hits that potentially form a track are picked, and then a fit is applied to determine the track parameters and vertex location. Finally, the reconstructed tracks and vertices are delivered back to the host.

threshold is applied to these filtered edge scores to retain true edges while eliminating false ones. The edge classification step uses the interaction network [28]. In the end, the GNN edge scores are passed to the weekly-connected-component (WCC) algorithm to form track candidates.

The Exa.TrkX pipeline is implemented in C++ for CPU- and GPU-only.[1] Trained PyTorch models are executed via the libTorch library. The nearest neighbor search is performed with the FRNN package [29] for GPU and FAISS [30] for CPUs. The connected component algorithm is from the Boost package, with potential future improvements from the CUDA-based version in cuGraph [31].

The Exa.TrkX pipeline has been integrated into the A Common Tracking Software (ACTS) framework [32]—an experiment-independent toolkit for track reconstruction. ACTS serves as a test bed for a range of tracking and vertex reconstruction algorithms. Simulation events are generated using Fatras fast simulation [33], which invokes `PYTHIA 8` [34] to simulate $t\bar{t}$ Monte Carlo (MC) events with an average number of additional proton-proton interactions within the same or nearby bunch crossings (pileup) of 200, replicating the conditions expected at the HL-LHC. The raw measurement from detectors is processed and clustered into spacepoints by the framework and provided as inputs to the Exa.TrkX pipeline. In the model utilized for this study, each spacepoint is characterized by three features, resulting in approximately 350 thousand points per event, corresponding to a data size of average size of 3.4 MB per event. The output from the Exa.TrkX pipeline includes track candidates with the average size of 1.4 MB per event.

[1]The software can be found at https://github.com/The-ExaTrkX-Project/exatrkx-service.

2025 JINST 20 P06002

2.3 Inference as a service using NVIDIA Triton Inference Server

The inference as a service approach in this paper is implemented using the NVIDIA Triton Inference Server [35]. It is an open-source package built on the open-source gRPC server package that standardizes the deployment and execution of ML models across various workloads [36]. It natively supports several machine learning frameworks as backends: PyTorch [37], TensorFlow [38], TensorRT [39], and ONNX Runtime [40]. A backend is a wrapper around an existing ML framework that executes client requests and returns the results to the client via gRPC. The Triton server also supports custom implementations using C/C++ or Python, denoted *"custom backend"* in the following discussion. In this paper, we utilized the flexibility of the custom backend to demonstrate the potential to apply it to any custom workflow.

2.3.1 Custom backend

A custom backend may be necessary depending on the complexity of the algorithm pipeline. For example, a complex ML pipeline may consist of multiple modules that do not efficiently align with the ensemble model architecture provided by Triton. [41]. Data transfer between models on different GPUs introduces significant overheads. In this scenario, the backend can be meticulously designed to integrate seamlessly with Triton, utilizing its official API to manage heterogeneous computing tasks efficiently. The custom backend also provides a convenient way to scale traditional rule-based algorithms. It serves as a wrapper function around existing algorithms and provides an easy pathway to scaling them with Triton. The use of a custom backend has a few additional benefits:

- *Fine-grained control*: custom backends allow developers to optimize performance at a low level to avoid unnecessary type conversion and copying in memory, which is preferable for complex pipelines with many models chained together.
- *Low latency*: lower latency can be achieved by bypassing the high-level ML framework and implementing custom operations to avoid redundant memory copying, which is crucial for real-time applications.
- *Custom logic and operations*: developers can implement specialized operations or algorithms not supported by standard deep learning frameworks.
- *Integration with legacy code*: custom backends can be built from existing C/C++ codebases to avoid unnecessary rewriting and validation.
- *Support for non-standard data types:* custom backends can be designed to handle unique data types or formats not natively supported by the server.

2.3.2 Model performance measurement

The performance analyzer, `perf_analyzer`, is a performance measurement tool provided by the NVIDIA Triton. It is designed to evaluate the throughput and latency of model instances deployed on the server. This analyzer generates and sends a configurable number of inference requests to the Triton server, allowing developers to simulate different load conditions and assess how their models perform under various scenarios. `perf_analyzer` provides detailed metrics, including response times, latency, and throughput rates, helping developers identify potential bottlenecks and optimize their deployment for better efficiency. The `perf_analyzer` has been utilized in the following standalone studies, along with full LHC workflows, when specified.

2025 JINST 20 P06002

3 Patatrack as a Service

The Patatrack algorithm is implemented as a custom backend that runs on the NVIDIA Triton Inference Server. As shown in figure 2, the modules in the dark orange block are contained in the backend and are called during the inference stage. When launching the servers, environment data and configurations, such as the cabling map from the front ends to the detector, gain calibrations, and cluster calibration parameters, are preloaded into the server to be used later during inference. Since these values remain consistent during data-taking, they are configured beforehand rather than sent to the server at inference time.

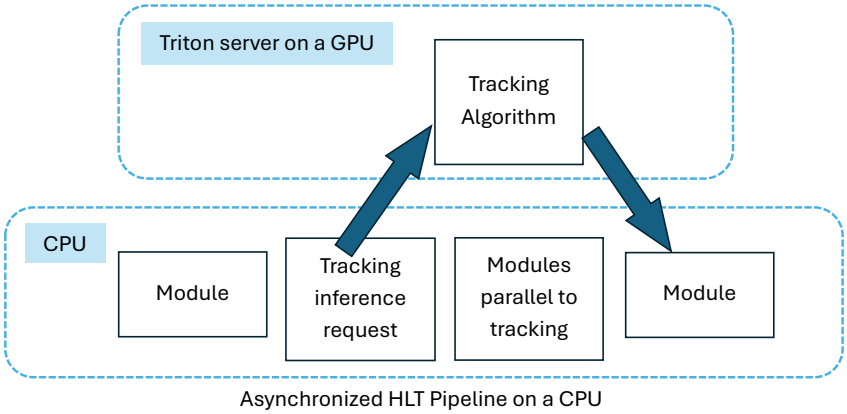

Figure 3. Illustration of the HLT running with tracking as a service. In the HLT workflow, the tracking algorithm inference runs on a server, while other modules in the workflow run on the client CPU in parallel to the tracking algorithms, ensuring no time is wasted waiting for track reconstruction results. This asynchronous pipeline has been implemented in production.

3.1 Standalone algorithm throughput tests

The throughput of running the algorithm as a service is tested and compared to running the algorithm directly on GPUs. In the as-a-service setup, the inference is triggered using dummy client inference calls. The results show that the throughput is found to be 400 events/s with a single-threaded client and 820 events/s when 10 threads are used to communicate with the server. For both tests, the server used is a NVIDIA T4 GPU. The resulting output data rate is found to be approximately 1 GB/s. For a single-threaded client and a direct connect setup, no significant difference in throughput is found when comparing direct GPU with inference as a service. For comparison, the CPU-only rate of Patatrack is found to be 25 events/s on a single 4-threaded HLT job.

3.2 HLT workflow throughput scanning

A GPU is considered saturated when increasing the number of CPU clients interacting with the GPU server no longer results in increased throughput. To measure the number of CPU clients that can interact with a GPU before it becomes saturated, a server with an NVIDIA Tesla T4 GPU is used, equipped with the Patatrack backend for inference tasks. On the client side, the CMS HLT workflow processes a prepared dataset to simulate actual collisions, with each job using one CPU thread per physical CPU core. The number of CPU clients is increased from 20 to 120, and the resulting throughput improvement is shown in figure 4.

2025 JINST 20 P06002

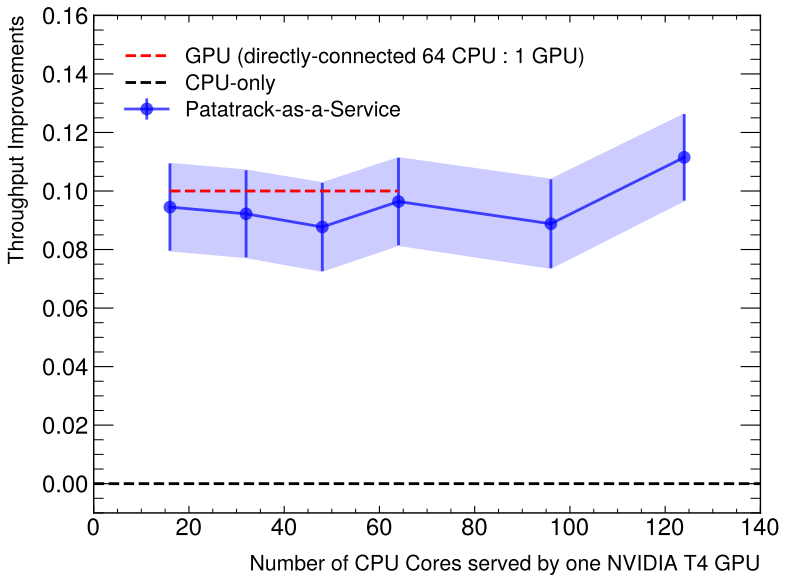

Figure 4. A scan of throughput improvement for the HLT workflow, varying the number of CPU clients communicating with a single GPU server. Direct inference on a GPU is limited to 64 CPUs requesting Patatrack inference. This results in about a 10% throughput gain compared to running it on the CPU alone (black dotted line). Using Patatrack as a service, the system can handle in excess of 120 CPU cores while maintaining the same level of throughput improvement, showing no GPU saturation.

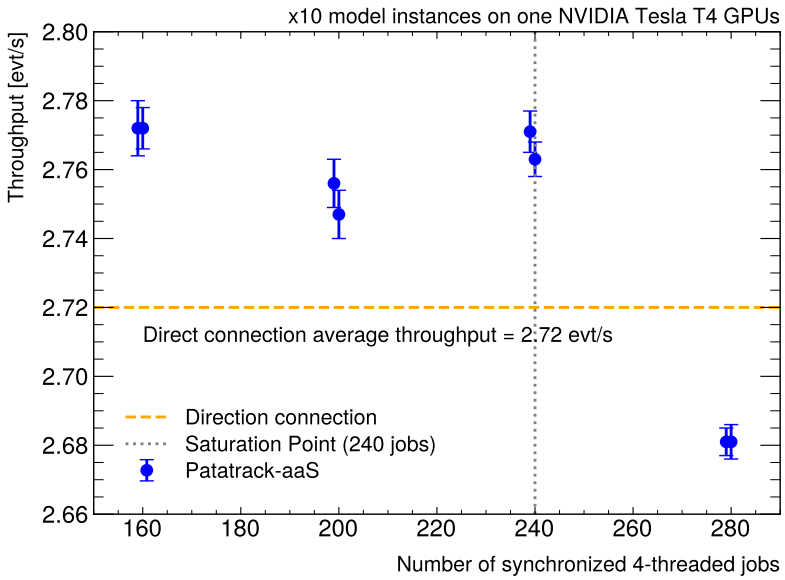

Figure 5. A throughput saturation scan is performed by launching a server with ten model instances loaded on one NVIDIA Tesla T4 GPU. The remote Triton server loads the Patatrack model, receiving inference requests from multiple synchronized 4-thread CPU client jobs. The throughput is expected to stay at the same level before the GPU computing resources are fully saturated. The server becomes fully saturated when around 240 synchronized 4-thread CPU client jobs send requests simultaneously. The throughput starts to drop beyond the saturation point.

2025 JINST 20 P06002

Before the GPU saturates, the throughput improvement remains flat as the number of CPU clients increases, stabilizing around 10%. This flat gain is approximately equal to the processing time of the Patatrack tasks offloaded to GPUs directly connected to CPUs. Compared to the current HLT workflow, which uses 64 CPU cores to serve one GPU, the as-a-service approach demonstrates significant potential to serve a larger amount of CPUs simultaneously and increase GPU utilization. This improved utilization can reduce the number of GPUs needed to serve the same number of clients.

A further test of the throughput saturation scan was conducted with a fully realistic setup of the HLT workflow using an emulated HLT system constructed using Google Cloud. By deploying a server with ten model instances running on a single NVIDIA Tesla T4 GPU, the Triton server hosted the Patatrack models and received inference requests from multiple synchronized 4-thread CPU client jobs. Each scan was performed twice per data point to ensure the stability of the results. The throughput remained stable until the GPU computing resources reached full saturation. Saturation occurred when 240 synchronized 4-thread CPU client jobs were sent to the server, beyond which the throughput began to decrease.

Throughput measurements were repeated multiple times at each data point, with the uncertainties calculated as the standard deviation of the measured throughput values for a given test. As shown in figure 5, the results show that the Patatrack-as-a-Service framework can process up to 240 simultaneous inference requests without experiencing a drop in throughput. A 2% increase in throughput was observed compared to the average throughput under direct connection, highlighting the server's capability to optimize and handle a higher number of simultaneous inference requests. Moreover, when comparing our approach with the existing CMS HLT GPU model, represented by the orange line, we find that we can more than double the number of threads that a GPU can service.

4 Exa.TrkX as a service in ACTS

The Exa.TrkX pipeline benefits from the flexibility of a custom backend. Custom backend provides fine-grained control when executing on the NVIDIA Triton Inference Server. This is especially important for sequential pipelines like Exa.TrkX that connect multiple ML models. The client communications with the remote Triton Inference Server are implemented in ACTS.

The standalone implementation includes a configurable class with interfaces for loading the full pipeline, performing inference, and setting the GPU device ID. Assigning models and data to a specific GPU minimizes data transfers and optimizes performance in multi-GPU environments. After this step, the Triton server manages inputs (inference requests) and outputs (responses), streamlining the process for server-side deployment.

4.1 Exa.TrkX backend lifecycle

The Exa.TrkX backend demonstrates how to use a custom backend as a wrapper around a complicated ML workflow, which is common in high energy physics. The backend architecture follows a modular life cycle divided into three phases:

- *Initialization*: configurations are loaded, and the model is prepared. During this phase, the server setup begins with the creation of a model object, which is configured according to a predefined input shape, backend library, and model path, as detailed in the configuration. The

2025 JINST 20 P06002

model instance fetches the device ID from the model object to ensure that all computations are executed on the same GPU where the data resides, enhancing processing efficiency.

- *Execution*: data is transferred to the GPU memory for processing. Model instances are created based on the instance_group settings in the model configuration. The server opens specific ports to accept inputs, such as a vector of 3D spacepoints position, via HTTP or gRPC protocols. These inputs are processed to generate track candidates.

- *Termination*: resources are cleaned up after processing, and responses are sent back to the client. This phase ensures that all resources are efficiently released and clients receive accurate output from the Exa.TrkX pipeline.

4.2 Standalone algorithm throughput tests

A standalone pipeline is used to inspect the performance of the Triton server, including throughput and latency. The tests use the simulated $t\bar{t}$ events as described in section 2.2.2 as input data, and the output track candidates are validated to be identical whether inference occurs on a local GPU or on a remote GPU accessed via the Triton server. The tests are performed on Perlmutter, a heterogeneous computing system at the National Energy Research Scientific Computing Center (NERSC). For direct inference, the tests are performed on computing nodes directly connected with GPUs. For inference using the Triton server, the client is set up on a CPU node while the server is launched on another node with access to a GPU. The data is transferred between the client and the server through the gRPC protocol. NVIDIA A100-SXM4 GPUs with 40 GB and 80 GB of memory are tested.

4.2.1 Multiple model instance scaling

The throughput and GPU utilization are measured with the performance analyzer as described in section 2.3.2. The client makes asynchronous calls to the server so that the client does not block the thread while waiting for the inference results from the server. The maximum throughput for a single GPU is measured to be about 1.75 events per second for both types of GPUs, shown in figure 6 for the 40 GB GPU. Saturation is reached when there are more than 2 model instances on the A100-SXM4 GPUs and when the number of concurrent requests is larger than the number of model instances hosted on the Triton server. The throughput increases with the number of model instances because the GPU utilization is improved. This point that multiple model instances better utilize the GPU is shown in figure 7. The latency in the measurements is dominated by computing inference time and queue time. Other components are negligible, including the time that the client sends/receives data, the network sends/receives data, and the server computes input/output. This is expected due to the simple network topology between the client and server in the current setup.

4.2.2 Multiple GPU scaling

The throughput for Triton servers with one GPU and four GPUs are compared. The throughput is measured with one model instance, and all GPUs are occupied with requests. We see an increase from 1.6 to 4.6 events per second in figure 8. The default load balancer in the Triton server is used to distribute requests among all the connected GPUs. Further optimization of load balancing is beyond the scope of this study; it may be examined in the future when testing the server deployment at high-performance computing (HPC) centers.

2025 JINST 20 P06002

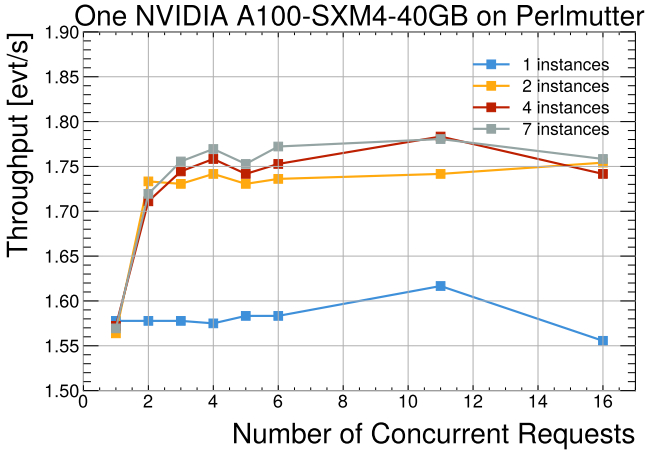

Figure 6. Inference throughput with the Exa.TrkX model on an A100-SXM4-40GB GPU. The throughput is measured with a fixed number of Triton instances while increasing the number of concurrent requests until the GPU saturates and reaches the maximum throughput. The throughput is compared between different numbers of model instances running on a single GPU. It is observed that with two or more model instances on the A100-SXM4 GPU all receiving requests, the maximum throughput of about 1.75 events per second can be reached.

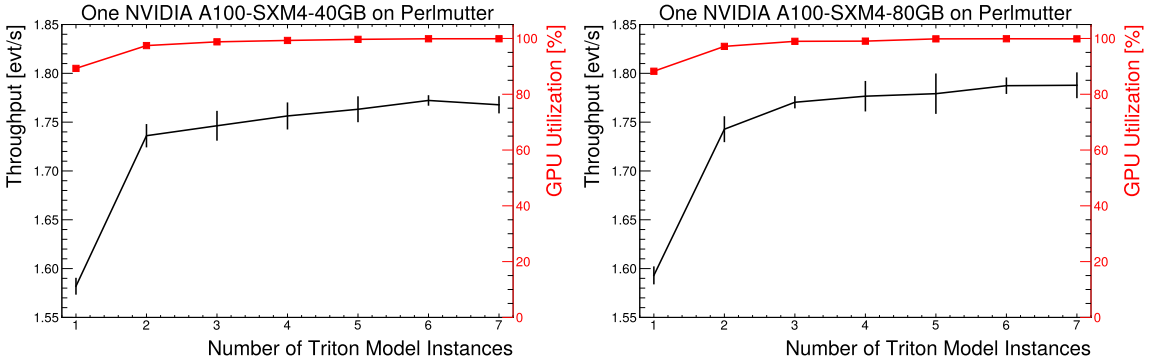

Figure 7. Maximum throughput and GPU utilization measured on an A100-SXM4-40GB (left) GPU and an A100-SXM4-80GB (right) GPU with 1 to 7 model instances. No significant throughput difference is observed between the two GPUs with different amounts of memory. Each data point represents the average throughput of 10 measurements when the GPU reaches the maximum throughput, with the number of concurrent requests ranging from the number of model instances to 10 more concurrent requests beyond the saturation point. The throughput (black line) and GPU utilization (red line) reach their maxima simultaneously when there are four or more model instances.

4.3 Integrated throughput tests with ACTS

In the ACTS reconstruction workflow, the Exa.TrkX pipeline is the dominant component, consuming a significant fraction of the total processing time. The track reconstruction process can be divided into several steps. The raw measurements from the pixel and strip detectors are converted into three-dimensional space points for pattern recognition, executed in less than 8 ms on an AMD EPYC 7763 CPU. Subsequently, these space points serve as inputs to the Exa.TrkX pipeline, which identifies a collection of proto-track candidates. A proto-track collection is simply a list of space points that the Exa.TrkX pipeline predicts will form a track. The final step of this sequence is the track-fitting process using the Kalman filter.

2025 JINST 20 P06002

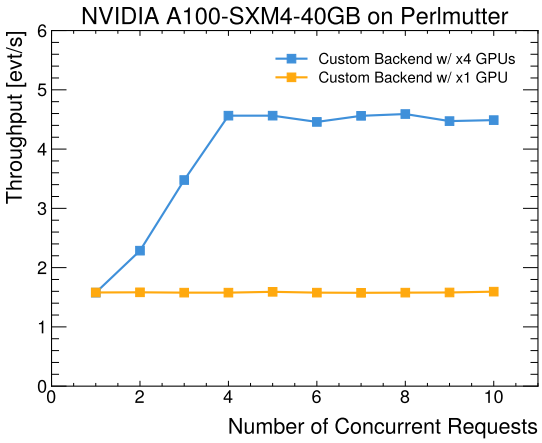

Figure 8. Throughput of Triton servers with 1 GPU and 4 GPUs. The maximum throughput increases from 1.6 events to 4.6 events per second.

Three implementations of the Exa.TrkX pipeline are considered and compared: direct CPU, direct GPU, and GPU inference as a service. In the first two cases, the Exa.TrkX pipeline is executed using the local device without interfacing with Triton. It is straightforward to implement an execution sequence that can switch between direct and as-a-service implementations. The mean processing duration was computed from ten simulated $t\bar{t}$ events, with inference times summarized in table 1. Under baseline conditions, the Exa.TrkX model operates on a dedicated CPU node at Perlmutter, accounting for approximately 95% of the timeline from raw measurement to track fitting. A significant speedup is achieved using the GPU to run inference for Exa.TrkX pipeline, completed in 2.4 seconds on a directly connected GPU. This investigation used an NVIDIA A100 GPU for both the direct and as-a-service approaches.

Furthermore, deploying the Exa.TrkX pipeline on a Triton server introduced no detectable overhead. The throughput is almost identical for inference via direct GPU and remote Triton server. This highlights the robustness of the Triton server implementation for the Exa.TrkX model and the significant gain in the event processing rate achievable for the HL-LHC.

The inference as a service measurement is performed with the client and server located at Perlmutter, so the network latency is expected to be negligible. This is relevant to demonstrate the potential application of this technology in the HLT farm, where the GPU cluster is close to the CPU processors, which minimizes the network latency. An alternative scenario could be to utilize a remote GPU farm from a powerful HPC, such as the National Research Platform at UCSD. In such a scenario, additional latency is expected, which strongly depends on the distance between the client and server that provides the inference calculation. However, using asynchronous communication between the client and server minimizes the impact of this latency on the event processing time and throughput, often to a level where it is negligible [11].

Table 1. Average inference time for the Exa.TrkX model using simulated $t\bar{t}$ events with an average pileup of 200.

Implementation	Exa.TrkX model inference time (s)
Direct CPU	9.65
Direct GPU	2.42
Exa.TrkX-aaS GPU	2.24

2025 JINST 20 P06002

5 Summary

Track reconstruction is a critical and computationally intensive step in data processing at the HL-LHC. This paper explores the potential of leveraging modern GPUs to address the growing computational challenges through an innovative *inference as a service* approach. Two representative track reconstruction algorithms were evaluated: the ruled-based Patatrack algorithm, utilized for online pixel track and vertex reconstruction, and the ML-based Exa.TrkX pipeline, which aims to be used in online and offline pattern recognition. Both algorithms were successfully adapted to run on the NVIDIA Triton Inference Server, enabling efficient use of GPU computing resources and increased throughput. Using custom backend opens up new possibilities, as it not only facilitates the efficient scaling of complex ML pipelines with Triton but also extends these capabilities to non-ML algorithms. It enables the seamless scaling of rule-based, non-ML workflows on GPUs, harnessing their substantial parallel processing potential.

When considering the benefits of using GPU as a service compared to direct GPU or CPU, we note that the GPU-as-a-service paradigm achieves near full GPU utilization; this allows for the maximum observed throughputs to be attained. When comparing directly with a full 64-core CPU, we find an approximate factor of 2 (Patatrack on an NVIDIA T4) to a factor of 4 (Exa.TrkX on an NVIDIA A100) increase in overall throughput. This leads to an approximate 4-8x reduction in power consumption and, ultimately, operational cost. The equivalent reductions without as-a-service are limited due to the underutilization of the GPU.

The tracking-as-a-service approach can achieve better throughput with minimal additional overhead compared to the traditional direct-connection approach while significantly improving GPU utilization. This method provides flexibility to dynamically scale resources based on demand, potentially reducing the total number of GPUs required. The tracking as a service approach enhances the adaptability and sustainability of computing resources, laying the groundwork for more efficient data processing pipelines in the era of the HL-LHC. Several future developments are still actively under investigation to bring this portability to other GPU vendors and other coprocessor types, ARM processors, tensor processing unit (TPU)s, and FPGA. There is also ongoing work to optimize the load balancing across multiple GPUs and to reduce the overhead further.

Acknowledgments

K. Pedro and N. Tran are supported by Fermi Forward Discovery Group, LLC under Contract No. 89243024CSC000002 with the U.S. Department of Energy, Office of Science, Office of High Energy Physics. Y. Feng was supported by Fermi Research Alliance, LLC under Contract No. DE-AC02-07CH11359 with the Department of Energy (DOE), Office of Science, Office of High Energy Physics and the DOE Early Career Research Program under Award No. DE-0000247070. Y. Chou, M. Cochran-Branson, J. Duarte, P. Harris, S. Hsu, M. Liu, P. McCormack, J.-F. Schulte, Y. Yao, and H. Zhao are supported by National Science Foundation (NSF) grant No. PHY-2117997. M. Liu, P. McCormack, J.-F. Schulte, and Y. Yao were supported by the U.S. CMS Software and Computing Operations Program under the U.S. CMS HL-LHC R&D Initiative. Additional support for cloud credits was obtained through the Internet2 Exploring Clouds to accelerate science grant NSF Award #1904444. 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.

2025 JINST 20 P06002

References

[1] L. Evans and P. Bryant, *LHC Machine*, 2008 *JINST* **3** S08001.

[2] ATLAS collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, 2008 *JINST* **3** S08003.

[3] CMS collaboration, *The CMS Experiment at the CERN LHC*, 2008 *JINST* **3** S08004.

[4] A. Strandlie and R. Fruhwirth, *Track and vertex reconstruction: From classical to adaptive methods*, *Rev. Mod. Phys.* **82** (2010) 1419.

[5] ATLAS collaboration, *ATLAS HL-LHC Computing Conceptual Design Report*, CERN-LHCC-2020-015, CERN, Geneva (2020).

[6] CMS Offline Software and Computing, *CMS Phase-2 Computing Model: Update Document*, CMS-NOTE-2022-008, CERN, Geneva (2022).

[7] J. Duarte et al., *FPGA-accelerated machine learning inference as a service for particle physics computing*, *Comput. Softw. Big Sci.* **3** (2019) 13 [arXiv:1904.08986].

[8] D.S. Rankin et al., *FPGAs-as-a-Service Toolkit (FaaST)*, in the proceedings of the *IEEE/ACM International Workshop on Heterogeneous High-performance Reconfigurable Computing*, Virtual, November 13 (2020), [DOI:10.1109/H2RC51942.2020.00010] [arXiv:2010.08556].

[9] J. Krupa et al., *GPU coprocessors as a service for deep learning inference in high energy physics*, *Mach. Learn. Sci. Tech.* **2** (2021) 035005 [arXiv:2007.10359].

[10] C. Savard et al., *Optimizing High-Throughput Inference on Graph Neural Networks at Shared Computing Facilities with the NVIDIA Triton Inference Server*, *Comput. Softw. Big Sci.* **8** (2024) 14 [arXiv:2312.06838].

[11] CMS collaboration, *Portable Acceleration of CMS Computing Workflows with Coprocessors as a Service*, *Comput. Softw. Big Sci.* **8** (2024) 17 [arXiv:2402.15366].

[12] M. Wang et al., *GPU-Accelerated Machine Learning Inference as a Service for Computing in Neutrino Experiments*, *Front. Big Data* **3** (2021) 604083 [arXiv:2009.04509].

[13] T. Cai et al., *Accelerating Machine Learning Inference with GPUs in ProtoDUNE Data Processing*, *Comput. Softw. Big Sci.* **7** (2023) 11 [arXiv:2301.04633].

[14] A. Gunny et al., *Hardware-accelerated Inference for Real-Time Gravitational-Wave Astronomy*, *Nature Astron.* **6** (2022) 529 [arXiv:2108.12430].

[15] A. Bocci et al., *Heterogeneous Reconstruction of Tracks and Primary Vertices With the CMS Pixel Tracker*, *Front. Big Data* **3** (2020) 601728 [arXiv:2008.13461].

[16] A. Bocci et al., *Bringing heterogeneity to the CMS software framework*, *EPJ Web Conf.* **245** (2020) 05009 [arXiv:2004.04334].

[17] Exa.TrkX collaboration, *Performance of a geometric deep learning pipeline for HL-LHC particle tracking*, *Eur. Phys. J. C* **81** (2021) 876 [arXiv:2103.06995].

[18] ATLAS collaboration, *Operation of the ATLAS trigger system in Run 2*, 2020 *JINST* **15** P10004 [arXiv:2007.12539].

[19] CMS collaboration, *Performance of the CMS Level-1 trigger in proton-proton collisions at* $\sqrt{s} = 13$ *TeV*, 2020 *JINST* **15** P10017 [arXiv:2006.10165].

[20] CMS collaboration, *Performance of the CMS high-level trigger during LHC Run 2*, 2024 *JINST* **19** P11021 [arXiv:2410.17038].

2025 JINST 20 P06002

[21] CMS collaboration, *The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger*, CERN-LHCC-2021-007, CERN, Geneva (2021).

[22] CMS collaboration, *The Phase-2 Upgrade of the CMS Level-1 Trigger*, CERN-LHCC-2020-004, CERN, Geneva (2020).

[23] M.M. Bronstein et al., *Geometric Deep Learning: Going beyond Euclidean data*, *IEEE Sig. Proc. Mag.* **34** (2017) 18 [arXiv:1611.08097].

[24] G. DeZoort et al., *Charged Particle Tracking via Edge-Classifying Interaction Networks*, *Comput. Softw. Big Sci.* **5** (2021) 26 [arXiv:2103.16701].

[25] K. Lieret et al., *High Pileup Particle Tracking with Object Condensation*, in the proceedings of the *8th International Connecting The Dots Workshop*, Toulouse, France, October 10–13 (2023) [arXiv:2312.03823].

[26] R. Liu et al., *Hierarchical Graph Neural Networks for Particle Track Reconstruction*, in the proceedings of the *21th International Workshop on Advanced Computing and Analysis Techniques in Physics Research: AI meets Reality*, Bari, Italy, October 24–28 (2023) [arXiv:2303.01640].

[27] S. Miao et al., *Locality-Sensitive Hashing-Based Efficient Point Transformer with Applications in High-Energy Physics*, in the proceedings of the *41st International Conference on Machine Learning*, Vienna, Austria, July 21–27 (2024) [arXiv:2402.12535].

[28] P.W. Battaglia et al., *Relational inductive biases, deep learning, and graph networks*, arXiv:1806.01261.

[29] L. Xue, *FRNN*, https://github.com/lxxue/FRNN.

[30] M. Douze et al., *The Faiss library*, arXiv:2401.08281.

[31] cuGRAPH, *RAPIDS Graph documentation*, https://docs.rapids.ai/api/cugraph/stable/.

[32] X. Ai et al., *A Common Tracking Software Project*, *Comput. Softw. Big Sci.* **6** (2022) 8 [arXiv:2106.13593].

[33] K. Edmonds et al., *The fast ATLAS track simulation (FATRAS)*, ATL-SOFT-PUB-2008-001 (2008).

[34] C. Bierlich et al., *A comprehensive guide to the physics and usage of PYTHIA 8.3*, *SciPost Phys. Codeb.* **2022** (2022) 8 [arXiv:2203.11601].

[35] NVIDIA, *NVIDIA Triton Inference Server*, https://docs.nvidia.com/deeplearning/triton-inference-server/user-guide/docs/index.html.

[36] Google, *gRPC: A high performance, open source universal RPC framework*, https://grpc.io/.

[37] A. Paszke et al., *PyTorch: An Imperative Style, High-Performance Deep Learning Library*, arXiv:1912.01703.

[38] M. Abadi et al., *TensorFlow: A system for large-scale machine learning*, arXiv:1605.08695.

[39] NVIDIA, *NVIDIA TensorRT*, https://developer.nvidia.com/tensorrt.

[40] ONNX, *Open Neural Network Exchange (ONNX*, https://github.com/onnx/onnx.

[41] H. Zhao et al., *Graph Neural Network-based Tracking as a Service*, in the proceedings of the *8th International Connecting The Dots Workshop*, Toulouse, France, October 10–13 (2023) [arXiv:2402.09633].

2025 JINST 20 P06002