Inference-as-a-Service Prototype at NERSC

Johannes Blaschke[1], Bruno Coimbra[2], Pengfei Ding[1], **Po-Han Huang**[3], Xiangyang Ju[1], Andrew Naylor[1], **Colin Thomas**[4], **Hilary Utaegbulam**[5], Michael Wang[2]

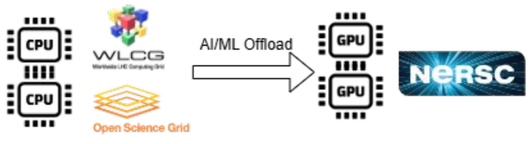
[1]Lawrence Berkeley National Laboratory, [2]Fermi National Accelerator Laboratory, [3]Georgia Institute of Technology, [4]University of Notre Dame, [5]University of Rochester

Introduction

We present a system providing inference-serving capabilities and metrics collection for multiple scientific ML models. The system is deployed across slurm-based compute nodes and a container-based platform. Metrics and traces are collected on both client and server via OpenTelemetry, with Prometheus and Jaeger used for time-series metrics and trace visualization. We demonstrate the ability to serve two GNN-based particle-tracking and track reconstruction models for ATLAS[1] and DUNE[2].

Accelerating Time-to-Science

Scientific workflows, such as SONIC[3] from the high energy physics (HEP) community, increasingly use AI/ML algorithms for data processing. Utilizing GPUs for these algorithms provides greater performance versus CPUs, therefore accelerating time-to-science.



Inference-as-a-Service (IaaS) connects GPU-rich HPC resources with CPU-rich systems, enabling:

1. **Faster AI/ML inference** by leveraging GPUs.
2. **Better resource utilization** by avoiding idle GPUs.
3. **Improved scalability** by combining HTC and HPC.

This model delivers a **flexible, efficient, and scalable** solution for accelerating scientific discovery.

Acknowledgement

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. DEAC02-05CH11231

Architecture

Both HPC users and external clients can submit requests to the ingress endpoint, where Envoy automatically load-balances them to the appropriate server hosting the requested model. The system scales based on Envoy load and resource saturation, and nodes self-register with Envoy via the registration service. Service discovery processes monitor heartbeats and retire inactive instances. CPU-bound components such as Envoy, Prometheus, Grafana, and Jaeger are managed by Spin, a containerized service management platform, while GPU-bound workloads are executed within the HPC cluster.

Both the client and server sides send trace timestamps to Jaeger's endpoint. To ensure that traces from both sides are linked into a single view, we include additional context in the request headers so the client and server can associate themselves. For developers seeking to optimize models, Jaeger provides detailed execution traces. For system administrators monitoring the entire system, we continuously send metrics such as inference time, queue time, and GPU utilization to a time series database, and visualize them in Grafana. Triton and Ray can also be configured based on model size and GPU utilization to run multiple parallel instances of the same model, enabling higher throughput.

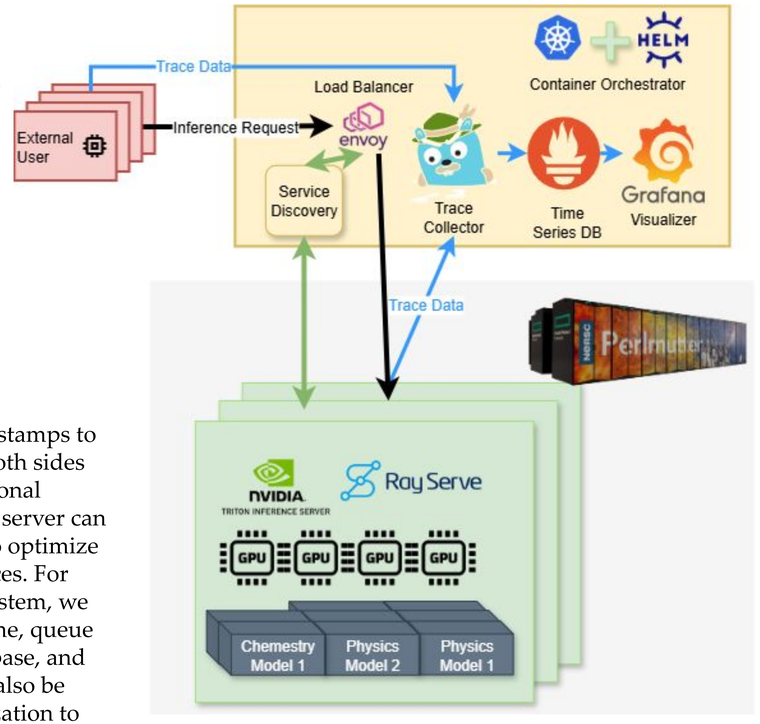


Figure 1: System architecture illustrating interactions between clients, edge services and HPC resources.

Observability

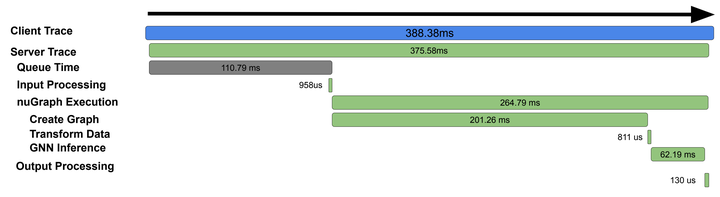
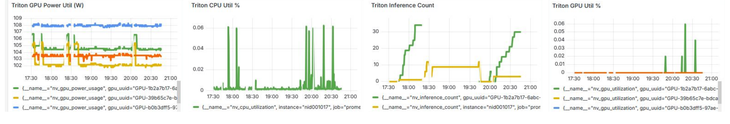


Figure 2: Time-series visualization from Jaeger (above) and resource utilization Grafana dashboard (right).



Our setup enables precise measurement of the time consumption at each processing stage, revealing a substantial slowdown in the graph edge-building phase during scaling. This insight allows us to trace the underlying cause of the observed plateau in overall throughput.

Performance Study

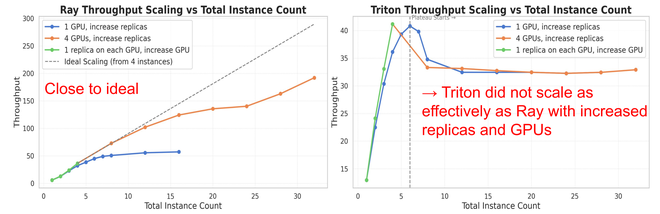


Figure 4: Inference throughput vs. number of model instances for Ray (left) and Triton (right).

We demonstrate the functionality of our system using a Graph Neural Network (GNN)-based particle tracking application for DUNE. The client component is implemented as a Python HTTP client with a 1000-thread pool. We conducted a comprehensive performance study of NVIDIA Triton and Ray Serve across a variety of server configurations and client scenarios. Triton outperformed Ray Serve in GPU-only workloads such as ResNet-50 when combined with client-side network optimizations like perf_analyzer. However, the Python-based nuGraph model used in our study includes multiple stages, several of which are CPU-intensive. In this scenario, Ray Serve achieved superior performance compared to Triton.

Future Work

To transition towards a **scalable, production-ready service** for the scientific community we aim to:

1. Conduct **large-scale testing** to improve scalability across HPC and HTC resources.
2. Explore **broader scientific applications.**
3. Integrate **authentication** with existing systems for secure, seamless access.
4. Streamline **deployment** via Helm charts and containerized packaging.

[1] Zhao, H. et al. 2024. Graph neural network-based tracking as a service. arXiv:2402.09633 [physics.comp-ph]
[2] V Hewe et.al. 2024. Graph Neural Network for Neutrino Physics Event Reconstruction. arXiv:2403.11872 [physics.data-an]
[3] The CMS Collaboration. 2024. Portable acceleration of CMS computing workflows with coprocessors as a service. Computing and Software for Big Science 8 (2024), Article 17. DOI: 10.1007/s41781-024-00124-1.