



GPU-accelerated machine learning inference for offline reconstruction and analysis workflows in neutrino experiments

Tingjun Yang, Maria Acosta, Phil Harris, Ben Hawks, Burt Holzman, Jeff Krupa, Kevin Pedro, Nhan Tran, Mike Wang

HSF-WLCG Virtual Workshop

Nov 24, 2020



v SONIC

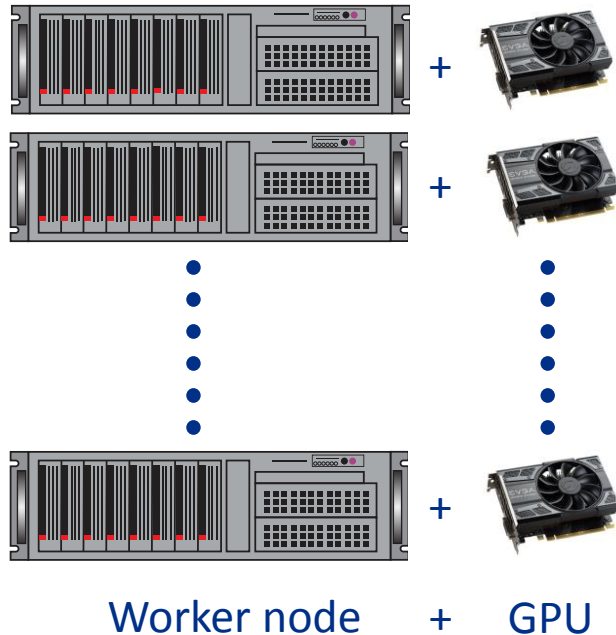


Introduction

- Machine learning applications, especially those employing deep neural nets (DNNs) have proven to be effective analysis tools in LArTPC-based neutrino experiments
- As a consequence, their use has become more common in offline reconstruction chains
- DNN training has its own computing challenges
 - But happens ~once/year and outside of compute infrastructure
- **Inference** happens on billions of events many times a year
 - Massive datasets of statistically independent events
 - Unique challenge across HEP
 - Slow on CPUs

Naïve solution: deploy GPUs on every worker node

Cloud computing cluster

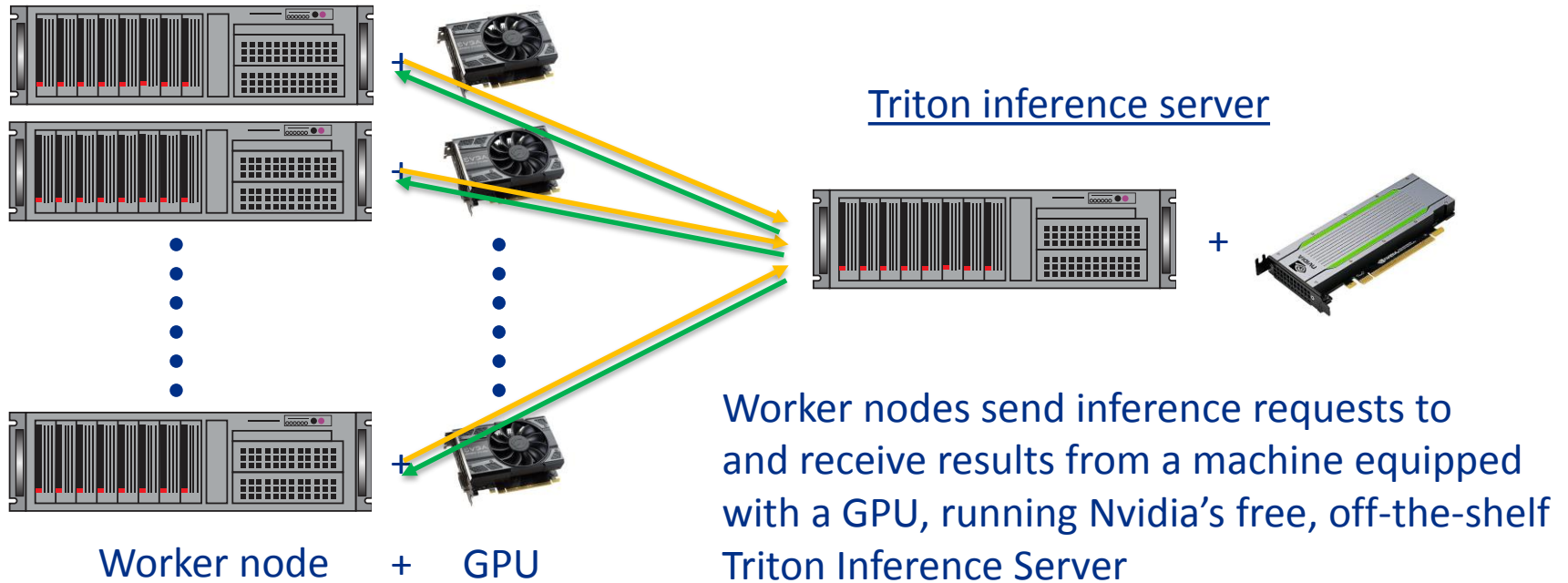


Assuming a moderately sized cluster with 100 nodes:

- Even with low to mid-range “gamer-class” GPUs, easily cost \$20k-30k
- Increased hardware and software maintenance
- Increased power and cooling requirements
- Inefficient use of GPU resources
- Less flexible and more costly to upgrade or replace

Alternative: GPU as a Service (GPUaaS)

Cloud computing cluster




GPUaaS @FNAL

- **SONIC**: Services for Optimized Network Inference on Coprocessors (GPUs, FPGAs, TPUs, ...)
 - C++ code running on CPU to convert data format and send the inference request to Triton.
 - DNN inference is handled completed by the Triton server on GPU.
- **Triton**: Inference server from Nvidia to use GPUs as a service
- Pioneering work by LHC experimentalists with Microsoft Research to demonstrate a proof-of-concept for providing FPGA-accelerated inference as a service for LHC experiments:

Original Article | Published: 14 October 2019

FPGA-Accelerated Machine Learning Inference as a Service for Particle Physics Computing

[Javier Duarte](#), [Philip Harris](#), [Scott Hauck](#), [Burt Holzman](#), [Shih-Chieh Hsu](#), [Sergo Jindariani](#), [Suffian Khan](#), [Benjamin Kreis](#), [Brian Lee](#), [Mia Liu](#), [Vladimir Lončar](#), [Jennifer Ngadiuba](#), [Kevin Pedro](#), [Brandon Perez](#), [Maurizio Pierini](#), [Dylan Rankin](#), [Nhan Tran](#) , [Matthew Trahms](#), [Aristeidis Tsaris](#), [Colin Versteeg](#), [Ted W. Way](#), [Dustin Werran](#) & [Zhenbin Wu](#)

Computing and Software for Big Science **3**, Article number: 13 (2019) | [Cite this article](#)



- First developed for CMS; now expanding to DUNE, ATLAS, astro...

GPUaaS for CMS

- The CMS triton client is now officially integrated: [SonicTriton](#)
- Inference server “ailab01.fnal.gov” located at Fermilab FCC2:
 - server class machine (2 x 10-core Cascade Lake Xeon Silver CPUs)
 - running Nvidia Triton inference server (r19.10)
 - “Turing” architecture Tesla T4 GPU (2,560 CUDA + 320 Tensor cores, 16GB GDDR6) and FPGA-based accelerators
 - T4 is a lower power (and cost) GPU for inference. More powerful GPUs (e.g. V100) are available.
- [arXiv:2007.10359](#): **GPU coprocessors as a service for deep learning inference in high energy physics**
- We have recently started applying GPUaaS to the LArTPC reconstruction using **ProtoDUNE** as an example.

GPUaaS for DUNE

ORIGINAL RESEARCH ARTICLE

Front. Big Data | doi: 10.3389/fdata.2020.604083

GPU-accelerated machine learning inference as a service for computing in neutrino experiments

Provisionally accepted The final, formatted version of the

article will be published soon.  [Notify me](#)

 [Michael Wang](#)^{1*},  [Tingjun Yang](#)^{1*}, [Maria Acosta Flechas](#)¹, [Philip Harris](#)²,  [Benjamin Hawks](#)¹,  [Burt Holzman](#)¹, [Kyle Knoepfel](#)¹, [Jeffrey Krupa](#)²,  [Kevin Pedro](#)¹ and  [Nhan Tran](#)^{1, 3*}

¹Fermilab (DOE), United States

²Massachusetts Institute of Technology, United States

³Northwestern University, United States

- Accepted for publication in the section Big Data and AI in High Energy Physics of the journal *Frontiers in Big Data*!
- Use GPUaaS to accelerate ProtoDUNE reconstruction.

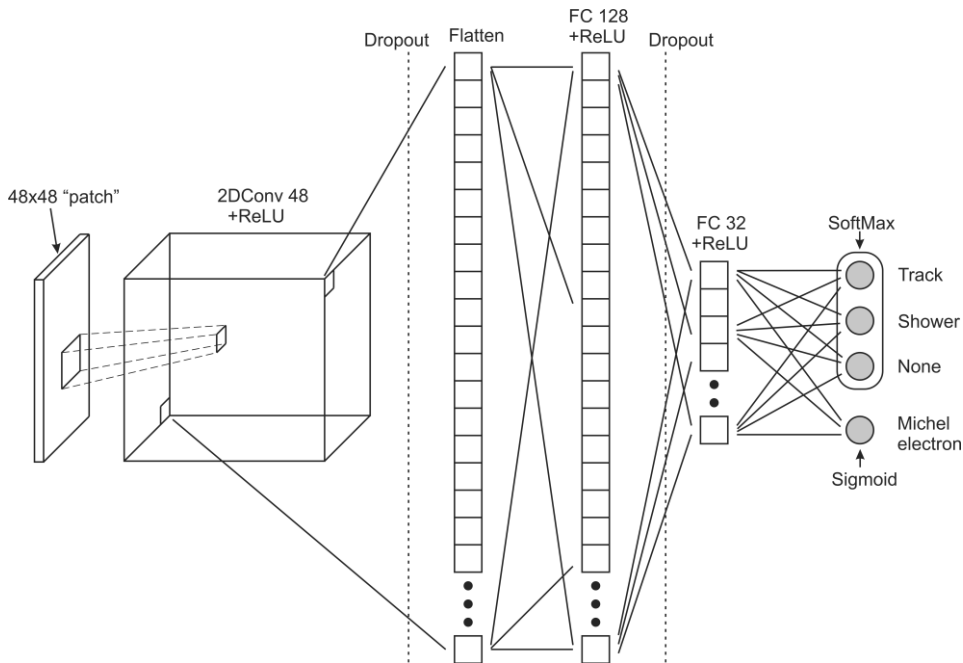


ProtoDUNE SP ~1kt LAr-TPC at CERN

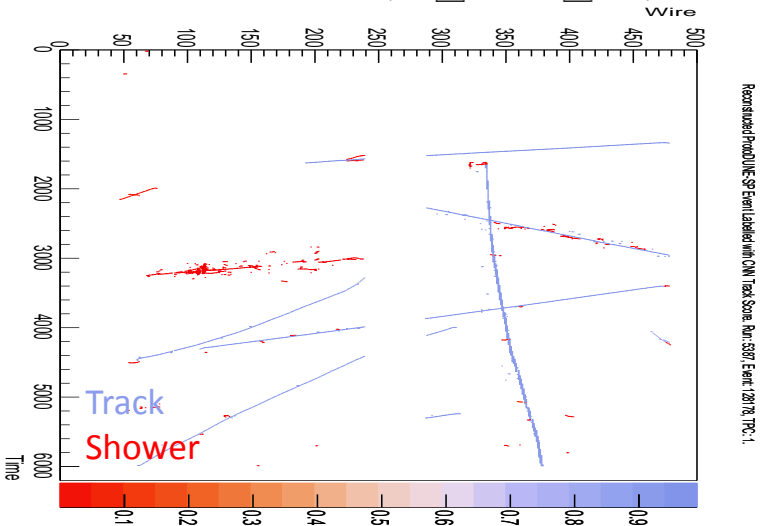
One of the two prototypes for DUNE far detector

[arXiv:2007.06722](https://arxiv.org/abs/2007.06722): First results on ProtoDUNE-SP liquid argon time projection chamber performance from a beam test at the CERN Neutrino Platform

EmTrkMichelId CNN



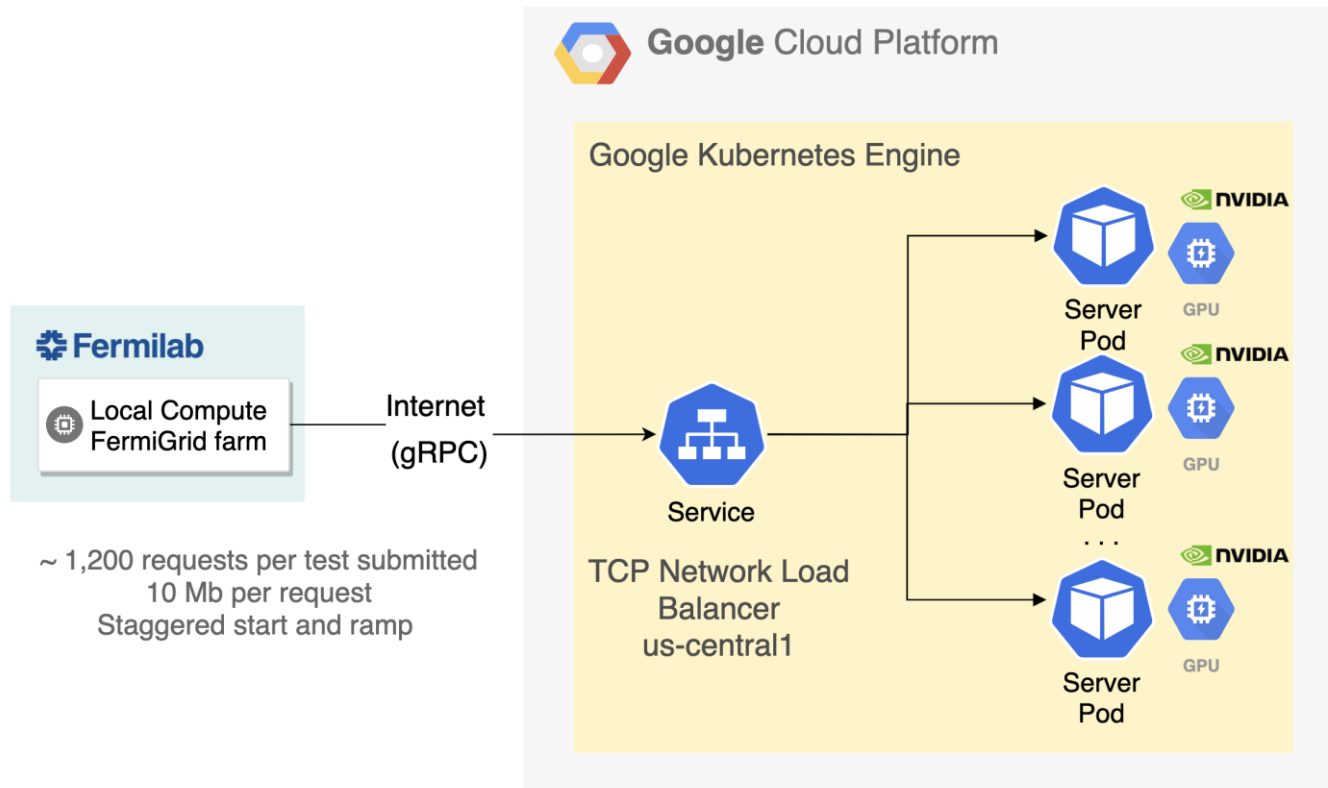
- Most time-consuming module in the reco chain.
 - Input of 48x48 pixel patch
 - A single convolutional layer containing 48 5x5 pixel filters
 - Two dense layers and two dropout layers
 - Output split into two branches
 - Track/Shower/None
 - Michel electron
- ~11.9M parameters
- Each event has ~55k patches



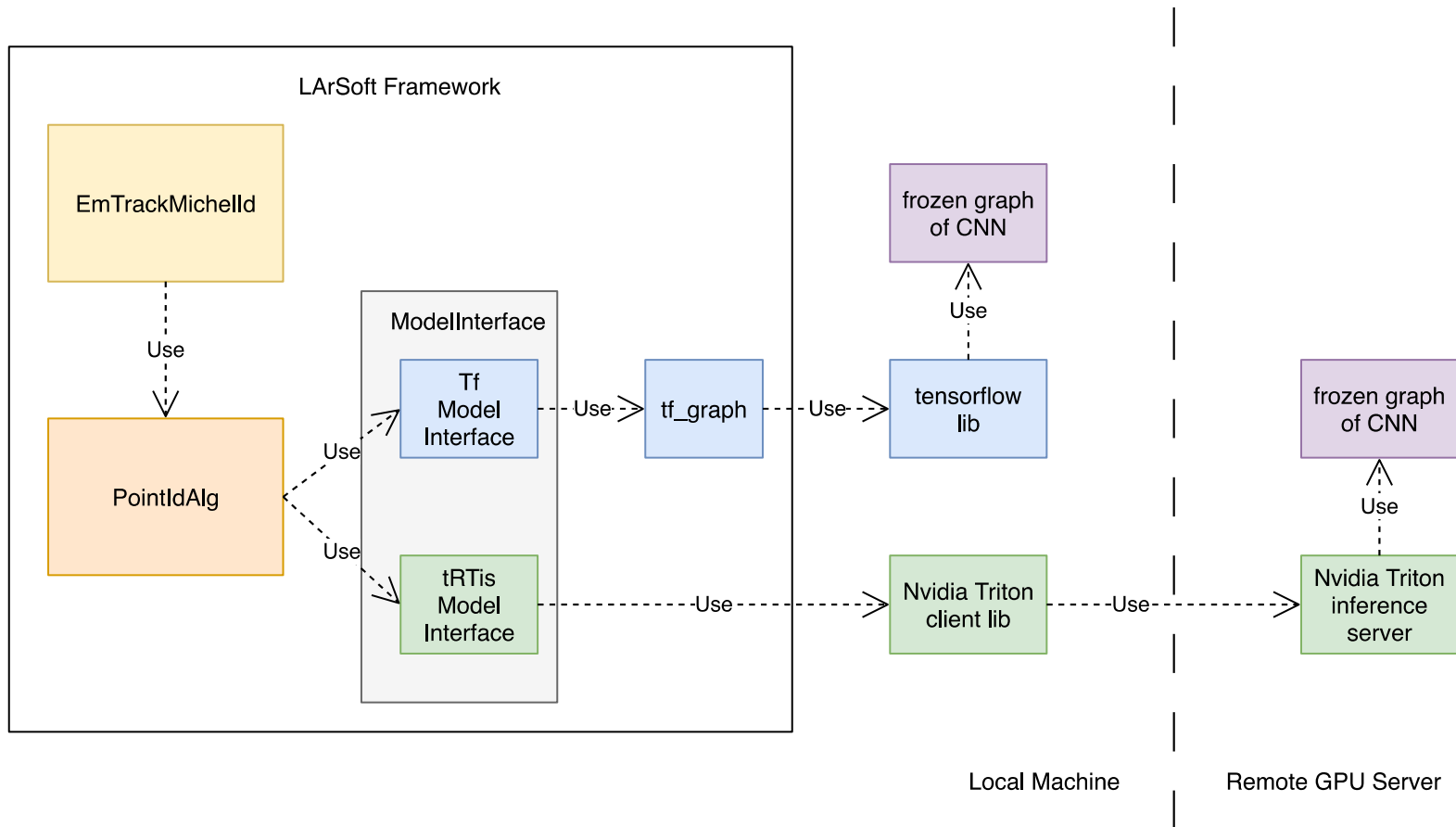
ProtoDUNE reco chain	CPU time/event
Non-ML module	110s
ML module (EmTrkMichelId)	220s
Total	330s

GPUaaS for ProtoDUNE

- Use the K8s Triton inference server attached with 4 Nvidia T4 GPUs on the Google Cloud.
- gRPC: open-source remote procedure call (RPC) system developed by Google.
- Goal is to demonstrate the improvement in inference time when using GPU as a service.



GPUaaS for LArSoft



- LArSoft: common framework for LArTPC reconstruction
- Interface added to communicate with the remote Triton inference server.

Processing time using SONIC

	CPU time/event	SONIC
Non-ML module	110s	110s
ML module (EmTrkMichellId)	220s	12s
Total	330s	122s

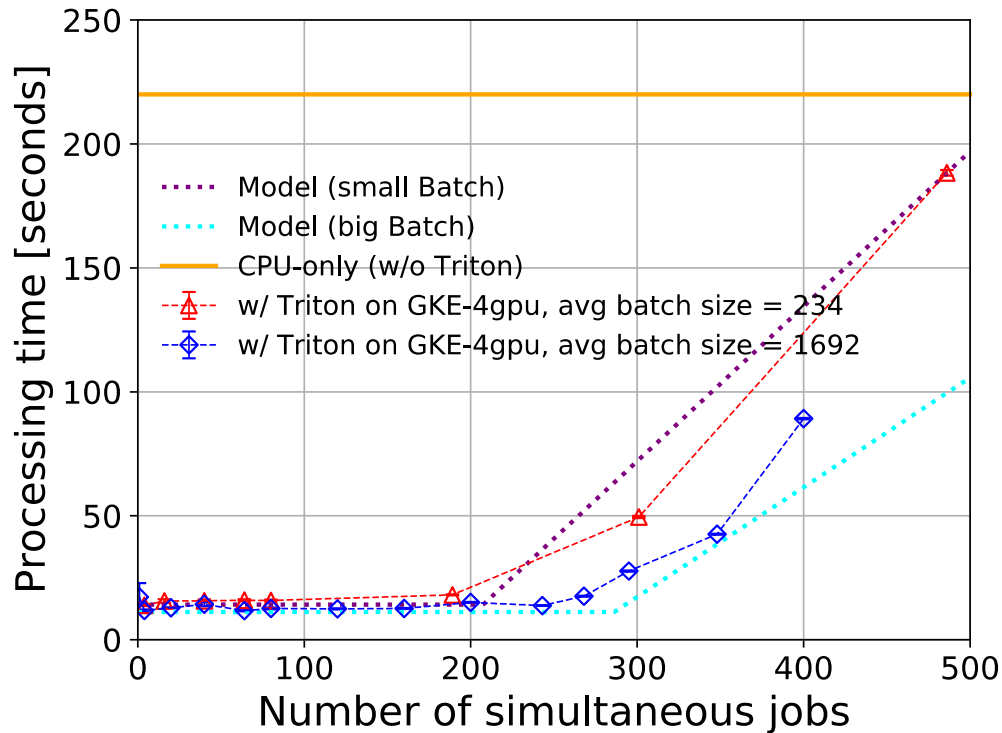


- 18x reduction in processing time for EmTrkMichellId module when using SONIC!

Break down of SONIC time

- We studied two batch sizes
 - Batch size 235, number of batches per event = 235
 - Batch size 1693, number of batches per event = 32
 - Big batch is preferred as it increases the inference speed and reduces latency.
- The total ML processing time using SONIC is ~12 s/event
 - = 7s (ML model preprocessing on CPU)
 - + 1.9s (Bandwidth latency @ 2Gbps)
 - + 0.4s (Travel latency to GCP in Iowa)
 - + 2.7s (GPU inference time)

Data and Time scaling model



- Processing time is a constant up to 190 (270) processes for small (big) batch size.
- Optimal ratio of CPU processes to a single GPU is **68:1**.
 - Ratio determined by the CPU time for non-ML module and the SONIC time for ML module.

Conclusions

- SONIC accelerates ML inference for ProtoDUNE reconstruction
 - 18x speed up of the ML module
 - 2.7x speed up of full ProtoDUNE workflow
- Acceleration needs: 1 T4 GPU per ~68 CPU processes
- $\Delta T_{\text{event}} = 110\text{s}$ (Non-ML Modules)
 - + 7s (ML model preprocessing on CPU)
 - + 1.9s (Bandwidth latency @ 2Gbps)
 - + 0.4s (Travel latency to GCP in Iowa)
 - + 2.7s (GPU inference time)
 - = **122s + any additional latency from saturated GPU**

Future work

- We are testing the ProtoDUNE offline production using GPUaaS on Google Cloud.
- Continue optimizing GPU implementation with TensorRT with tools such as quantization.
- Try other hardware such as FPGAs, IPU
- Explore deployment at scale at Feynman (FNAL) and HPCs.

Thank you for your attention!

Layer	Output Shape	# Parameters
Conv2D	44 x 44 x 48	1248
Dropout-1	44 x 44 x 48	0
Flatten	92928	0
Dense-1	128	1189491
Dropout-2	128	0
Dense-2	32	4128
Output "emtrk_none_out"	3	99
Output "michel_out"	1	33
Total Number of Parameters		11,900,420

5*5*48+48, 5x5 kernel, stride=1
ReLU activation

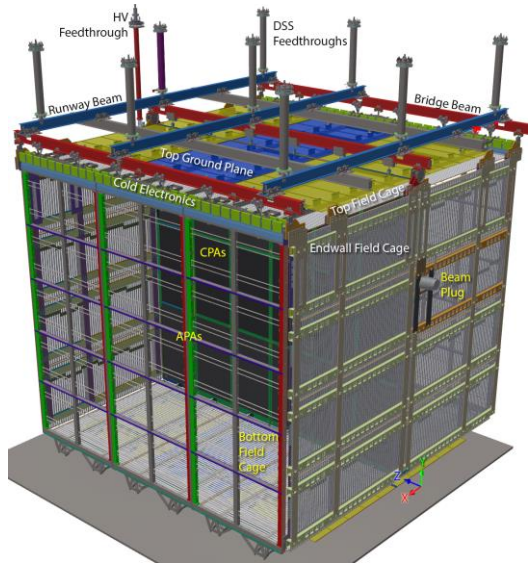
92928*128+128

128*32+32

32*3+3 softmax activation

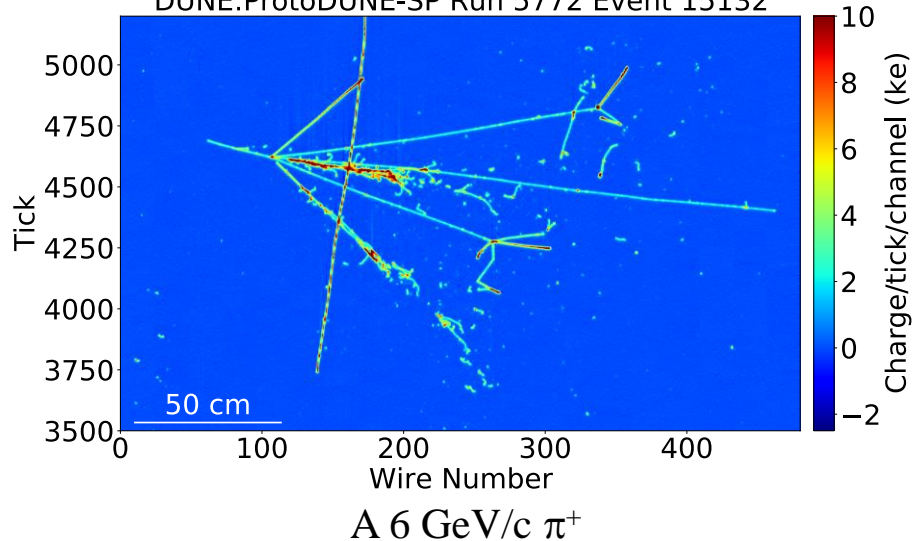
32*1+1 sigmoid activation

ProtoDUNE event reconstruction



- Largest LArTPC ever built
 - 7.2 x 6.0 x 6.9 m³
 - 15,360 channels
 - Wire spacing 5 mm
 - Readout window 3 ms
- Lots of activities in the TPC
 - Cosmic ray muons
 - Beam particles
- Reconstruction chain
 - Noise mitigation and deconvolution
 - Hit finder
 - Pandora pattern recognition
 - **CNN EmTrkMichellD**

DUNE:ProtoDUNE-SP Run 5772 Event 15132



GPUaaS for LArSoft

- The Triton client is now fully integrated in LArSoft
 - LArSoft: a common framework for LArTPC simulation and reconstruction, used by many experiments (MicroBooNE, SBN, DUNE/ProtoDUNE, etc.) – larsoft.org
 - TrtIS (Triton Inference Server) client libraries are available as a UPS product (trtis_clients)

```
mwts{mwang}1002% setup trtis_clients v19_11a -q e19:prof
mwts{mwang}1003% ups active
Active ups products:
gcc                v8_2_0          -f Linux64bit+3.10-2.17      -z /products
opencv            v4_2_0          -f Linux64bit+3.10-2.17 -q e19:p372 -z /products
protobuf         v3_11_2a       -f Linux64bit+3.10-2.17 -q e19      -z /products
python           v3_7_2         -f Linux64bit+3.10-2.17      -z /products
sqlite           v3_26_00_00    -f Linux64bit+3.10-2.17      -z /products
trtis_clients     v19_11a        -f Linux64bit+3.10-2.17 -q e19:prof -z /products
ups              v6_0_8         -f Linux64bit+3.10-2.17      -z /products
```

- EmTrkMichellD is modified to include a new Tritis inference client Model Interface.
 - PointIdAlgTrtis_tool.cc

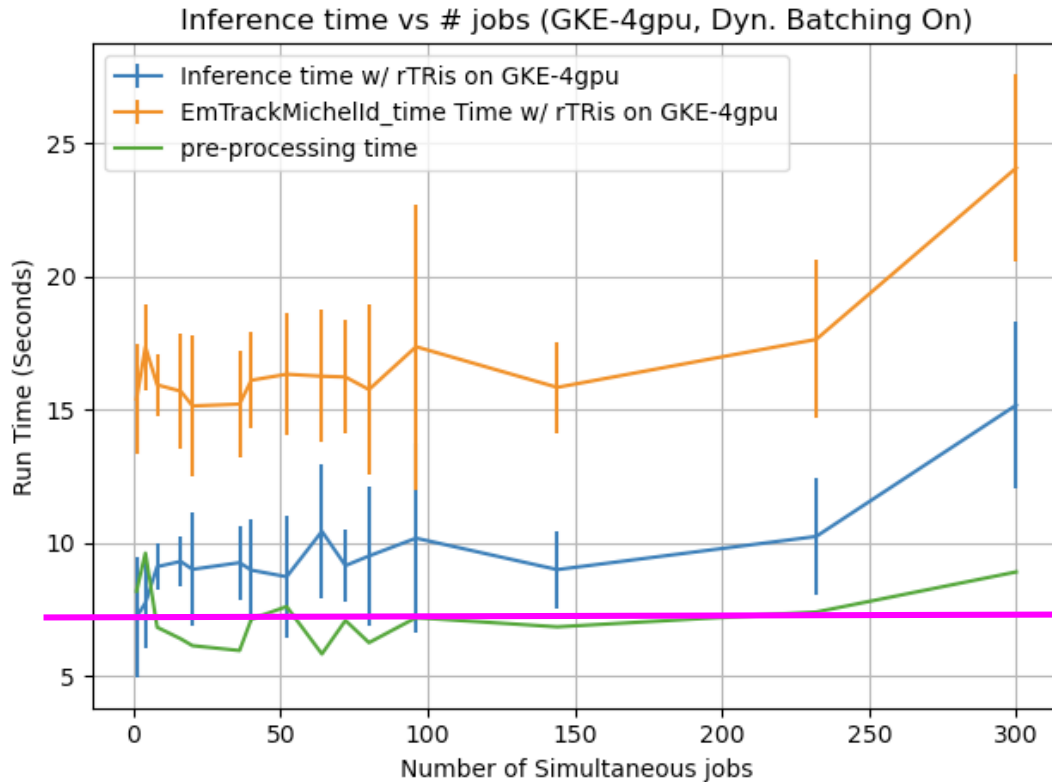
Different configurations being studied

- Small batch vs large batch
 - Batch size 235, number of batches = 235
 - Batch size 1693, number of batches = 32
 - Small batches means more gRPC calls and longer latency
- Dynamic batching on the Triton server
 - Accumulate requests from multiple events, then process together
 - Massive gain in efficiency and throughput
 - Simple configuration: ([docs](#))

```
dynamic_batching {  
    preferred_batch_size: [ 4, 8 ]  
    max_queue_delay_microseconds: 100}
```

Preprocessing time

Small batch



Preprocessing time should be constant and is around 7 seconds per event

- Preprocessing time: 7 s per event
 - Retrieve hit information, prepare patches, etc.

Communication latency

```
<des91.fnal.gov> ping 35.202.61.4
PING 35.202.61.4 (35.202.61.4) 56(84) bytes of data.
64 bytes from 35.202.61.4: icmp_seq=1 ttl=107 time=11.7 ms
64 bytes from 35.202.61.4: icmp_seq=2 ttl=107 time=11.7 ms
64 bytes from 35.202.61.4: icmp_seq=3 ttl=107 time=11.7 ms
64 bytes from 35.202.61.4: icmp_seq=4 ttl=107 time=11.7 ms
64 bytes from 35.202.61.4: icmp_seq=5 ttl=107 time=11.9 ms
```

Ping time is ~12ms.

Tests are run with large (~1693) and small batch size (~235)

For ~55k inferences per event this means:

~32 gRPC calls per event for large batch size

~235 gRPC calls per event for small batch size

Travel time (ΔT_{travel}) = 0.4 s for large batch and 2.6 s for small batch

Time on server vs. time on GPU

```
Concurrency: 20, 15933 infer/sec, latency 2476755 usec
Time elapsed
112
```

```
Concurrency: 20, 25333 infer/sec, latency 1538820 usec
Time elapsed
233
```

```
Inferences/Second vs. Client Average Batch Latency
Concurrency: 20, 17133 infer/sec, latency 2352441 usec
Time elapsed
123
```

```
Inferences/Second vs. Client Average Batch Latency
Concurrency: 20, 26666 infer/sec, latency 1491572 usec
Time elapsed
234
```

Perf Client tests show about ~20k inf/s +/- 2k (with larger errors)
For 55k inference per event, we expect time on GPU to be between 2.7+/-0.3s per event

Total time = 12s

$\Delta T_{\text{preproc}} \sim 7\text{s}$

$\Delta T_{\text{SONIC}} \sim 5\text{s}$

Indicates that there is additional non-trivial latency between when the request arrives at the server (ping) and the time it spends on the GPU:

$\Delta T_{\text{on GPU server}} \sim 5.1\text{s}$

$\Delta T_{\text{travel}} \sim 0.4\text{s}$ (large batch)

$\Delta T_{\text{on GPU}} \sim 2.7\text{s}$

$\Delta T_{\text{bandwidth}} \sim 2\text{s}$

One event:

48x48 image x 32b x 55,000 inferences = 3.9 Gigabits

Bandwidth = 2 Gbps

Bandwidth latency = 2s per event

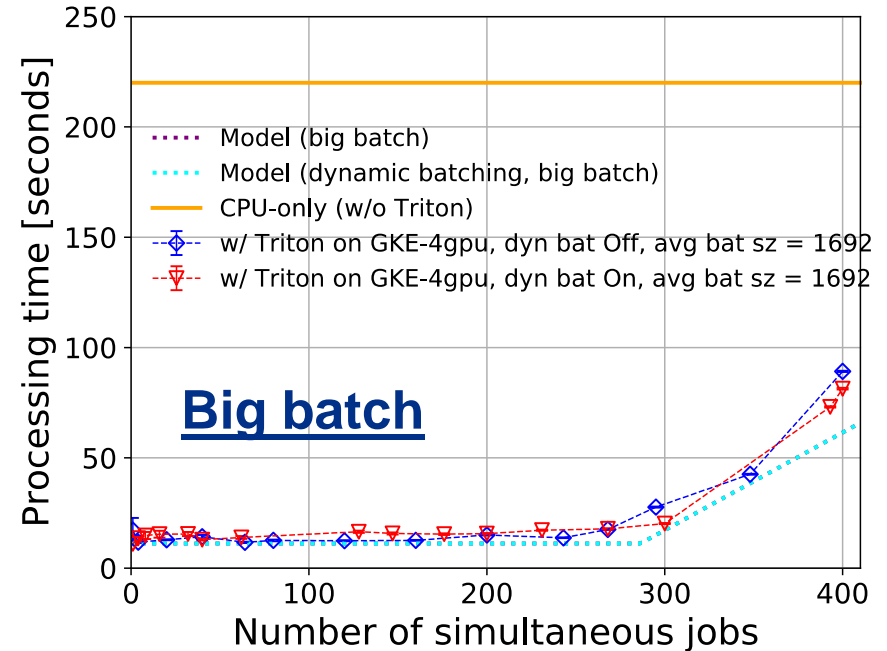
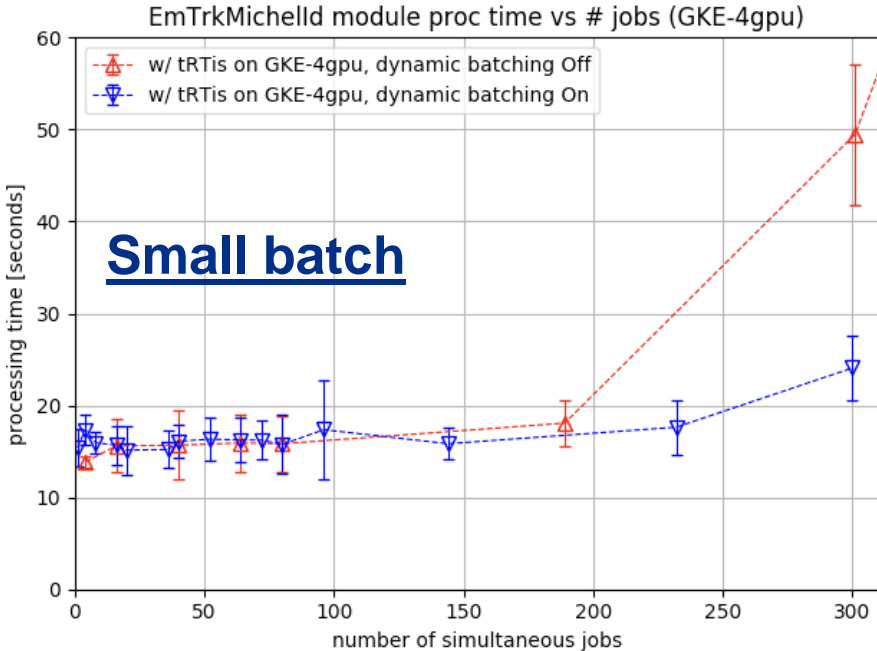
Processing time scaling

$$t_{\text{CPU}} = (1 - p) \times t_{\text{CPU}} + p \times t_{\text{CPU}}$$
$$t_{\text{ideal}} = (1 - p) \times t_{\text{CPU}} + t_{\text{GPU}} + t_{\text{latency}}$$

- p = fraction of parallelizable computations
- Ideal scenario: GPU not saturated, always available
 - Assume t_{GPU} small enough & CPU requests staggered enough
 - Blocking calls, CPU waits for GPU to finish
- GPU(s) can saturate if too many request sent: $\frac{N_{\text{CPU}}}{N_{\text{GPU}}} > \frac{t_{\text{ideal}}}{t_{\text{GPU}}}$
→ CPUs have to wait (effective t_{GPU} increases)

$$t_{\text{SONIC}} = (1 - p) \times t_{\text{CPU}} + t_{\text{GPU}} \left[1 + \max \left(0, \frac{N_{\text{CPU}}}{N_{\text{GPU}}} - \frac{t_{\text{ideal}}}{t_{\text{GPU}}} \right) \right] + t_{\text{latency}}$$

Dynamic batching



- Dynamic batching helps for small batch size but does not impact the case of big batch size
 - Does not seem to add any appreciable latency (or at least small on our time scales)