Studies on track finding algorithms based on machine learning with CPU, GPU and FPGA

Maria Carnesale on behalf of the ATLAS collaboration

ICHEP 2024

Overview

- Studying the performance of ML-based tracking algorithms on different architectures (CPU / GPU / FPGA)
- Discuss possibility to include ML algorithms for muon tracking at ATLAS muon High Level Trigger (HLT)
    - At the HL-LHC, an heterogeneous high-level triggering farm is considered, compromise between performance, costs, power-consumption
- Performance studies on standalone G4 simulation of a generic setup similar to the spectrometer and the ATLAS New Small Wheels

- Models tested are
    - **Dense NN (DNN)** for cluster reconstruction on strip detectors
    - **Convolutional NN (CNN)** and **Recurrent NN (RNN)** for pattern recognition

U250

Designed for machine learning inference, video transcoding, and database search & analytics

U50

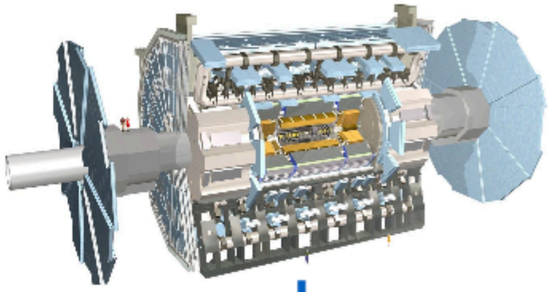
Designed for financial computing, machine learning, computational storage, and data search and analytics

VCK5000

It is an AI development card, more versatile than the other two

How do commercial FPGA perform?

HL-LHC ATLAS trigger system

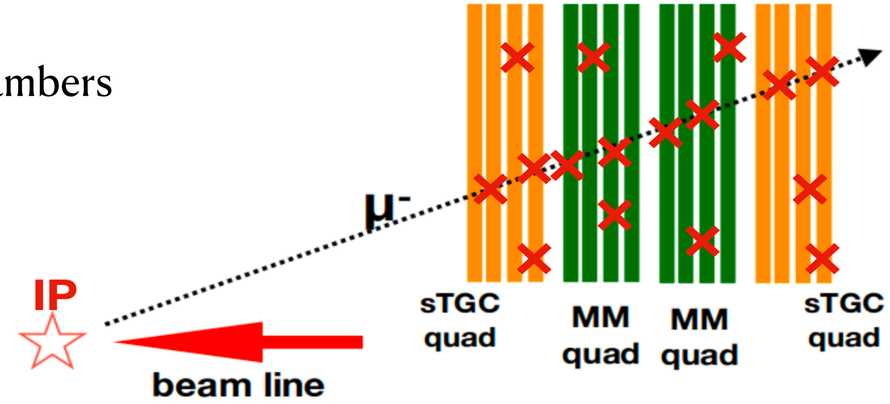


1 MHz

10 kHz

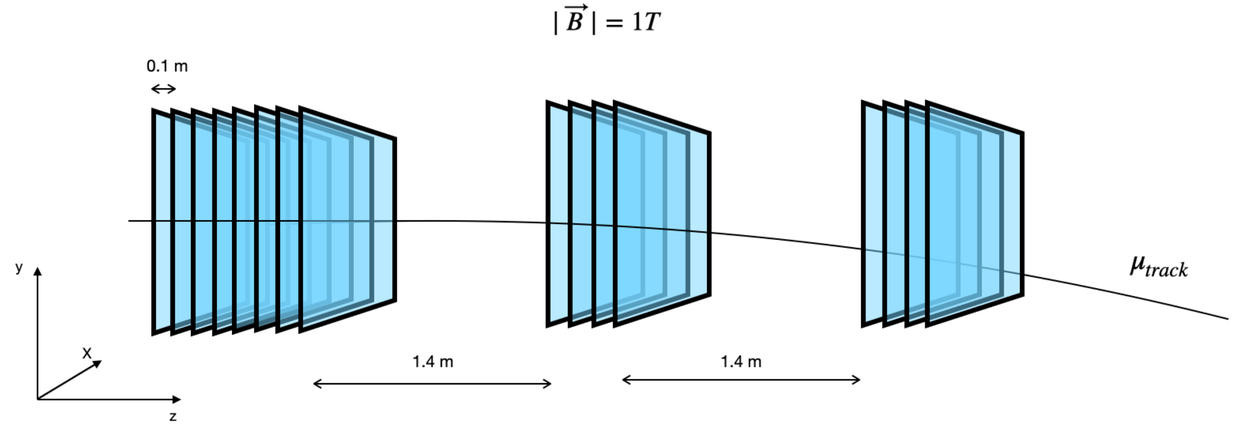
ML algorithms for muon pattern recognition

- Algorithms for **cluster reconstruction** in MicroMegas (MM) and small Thin Gas Chambers (sTGC) like detectors and **pattern finding**
- To speed up R&D part of the study, a toy model is simulated

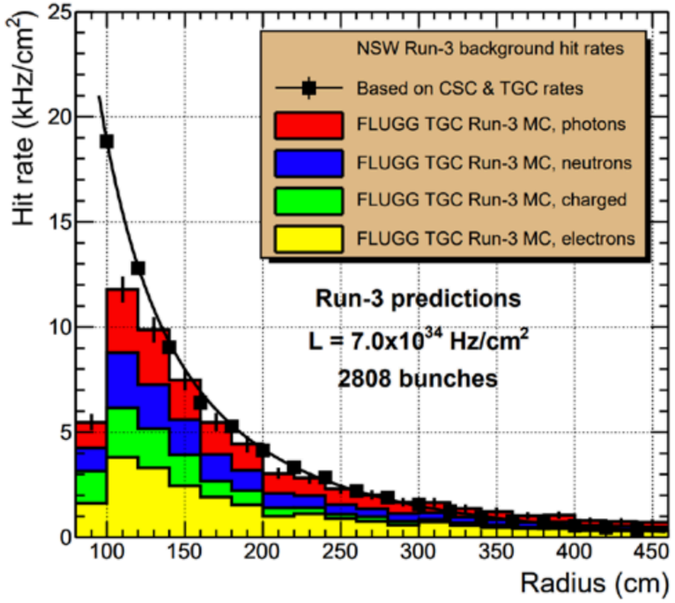


Simple MC standalone for a generic strip detector, G4 fullsim

- layers, b-field, background rate can be set
- 4 samples produced with different noise rates: $2,\ 5,\ 10,\ 15\ kHz/cm^2$ inspired by NSW HL-LHC rates
- Effect from correlated background is also emulated

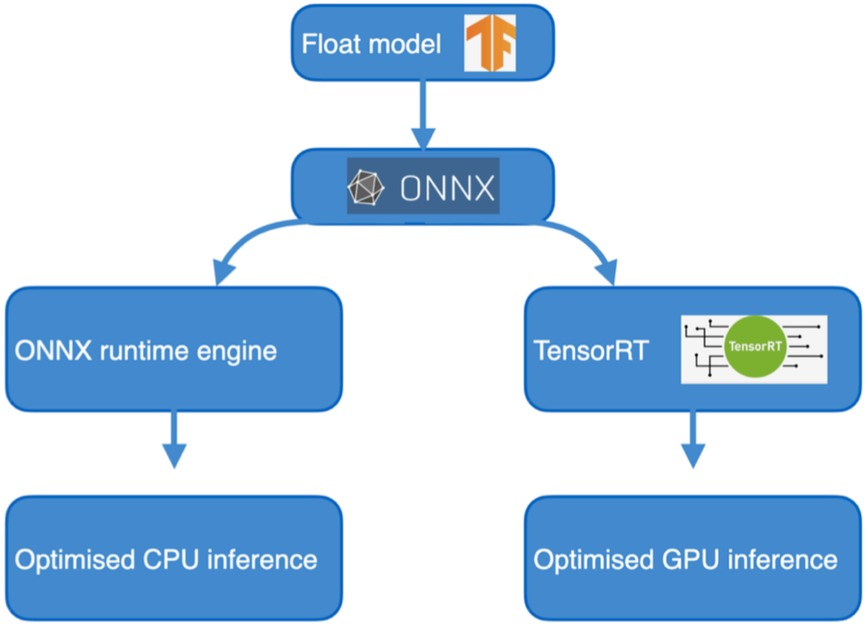


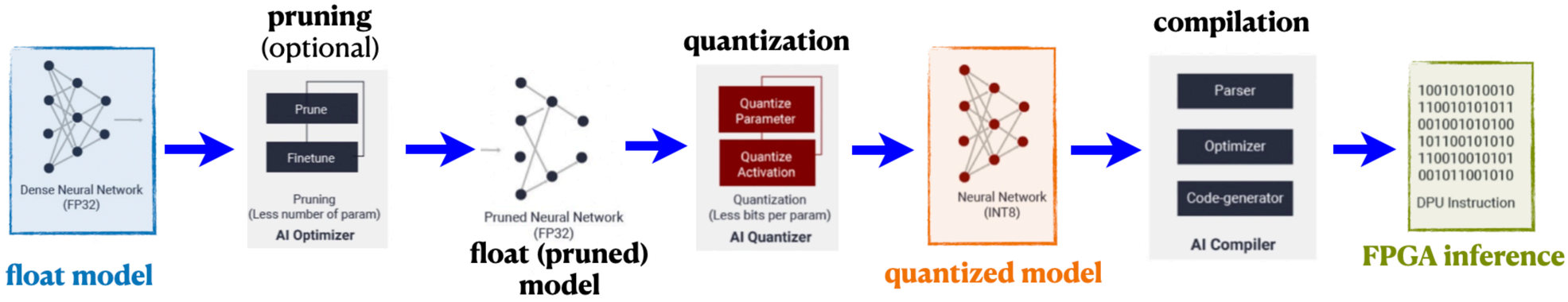
Simulated NSW rates at HL-LHC (NSW TDR)



ML algorithms tested on CPU/GPU/FPGA

- **Study of inference time and performance on different architectures:**
  - **CPU**: using ONNX
    - Open Neural Network Exchange: open source framework that optimizes the usage of CPU resources
  - **GPU**: using tensor flow and tensorRT
    - Framework produced by NVIDIA to run optimized inference on GPU
  - **FPGA**: using Vitis-AI workflow provided by **Xilinx** for inference acceleration



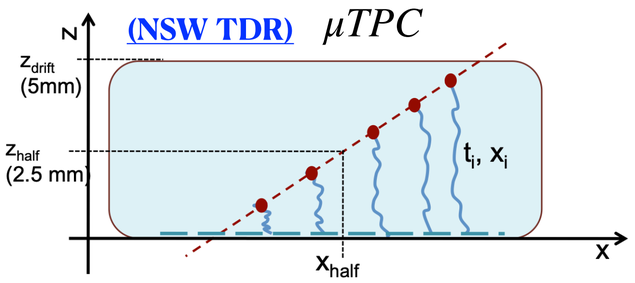


Cluster reconstruction with DNN

Current cluster position reconstruction in NSW with different algorithms:

centroid

$$x_{cluster} = \frac{\sum_{strips} q_{strip} \cdot x_{strip}}{q_{tot}}$$



Challenges

1. Resolution depends on incidence angle. 2."Correlated" background

Simple DNN for hit position reconstruction

NN architecture

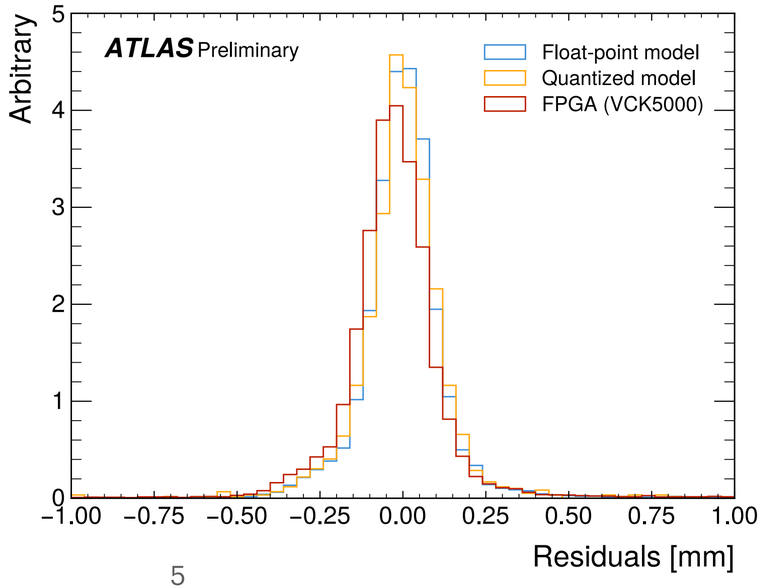
input_1: InputLayer	input:	[(?, 11)]
	output:	[(?, 11)]
dense: Dense	input:	(?, 11)
	output:	(?, 40)
dense_1: Dense	input:	(?, 40)
	output:	(?, 20)
dense_2: Dense	input:	(?, 20)
	output:	(?, 1)

Input variables:
- cluster size
- cluster theta
- strips charge
- strips time
- strips position

Label: track hit position

A simple DNN O(50k parameters and 20 input) improves the resolution around up to 50% wrt centroid method

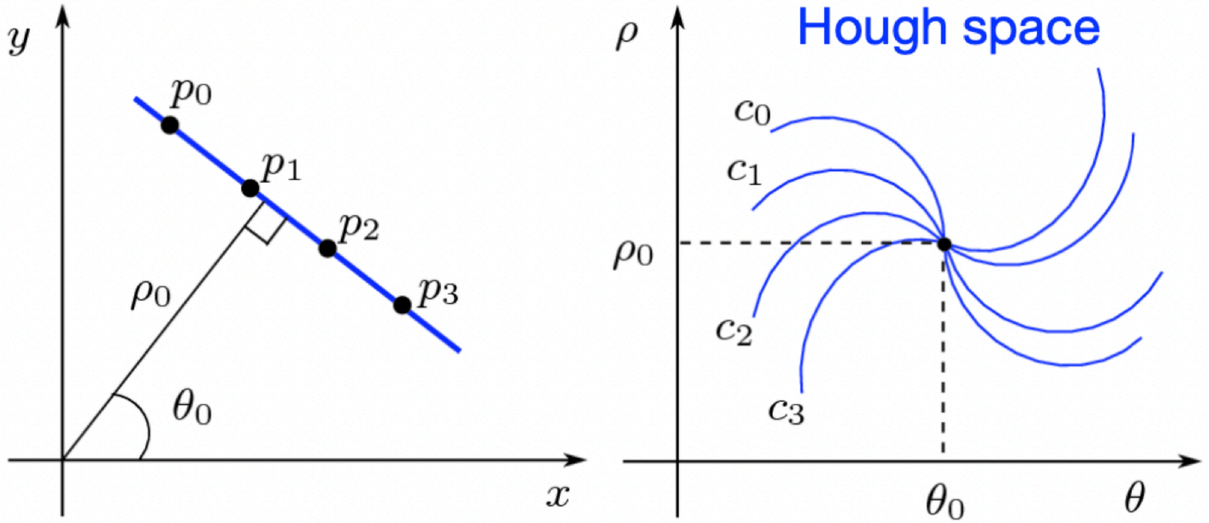
Different tests performed:
- Test on the trained model (**floating-point model**) with GPU
- Test on a **quantized model with GPU**
- Test of the **quantized model on the FPGA**



XILINX VITIS™ AI

- Tested using Vitis-AI workflow
- Small resolution degradation after quantisation

Pattern recognition with RNN



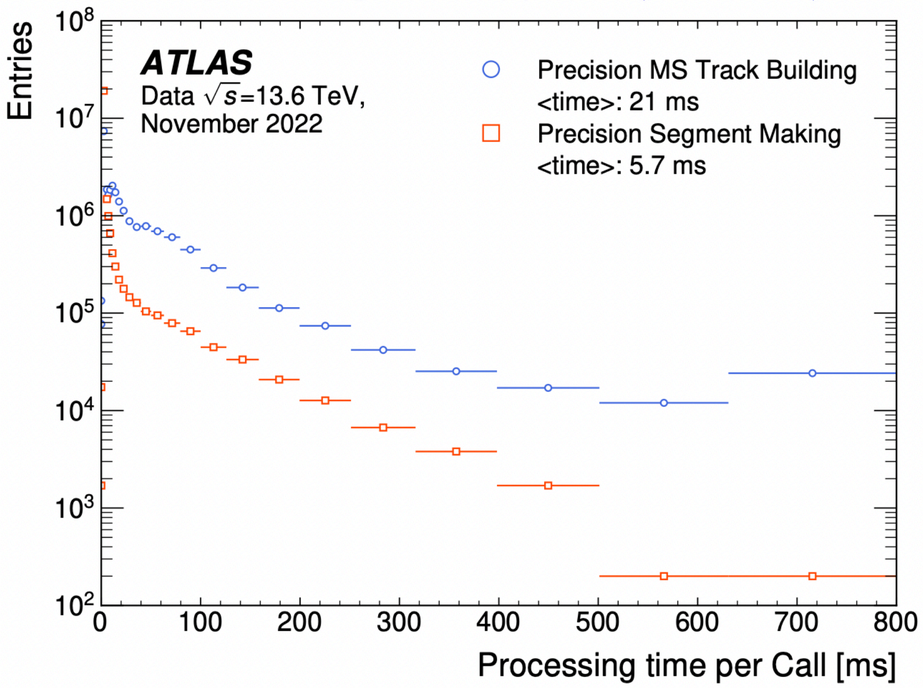
$$\rho = x \cdot cos\theta + y \cdot sen\theta$$

- Not optimized for NSW reconstruction:
  - 2D histogram binning optimization
  - Choice of cut on number of hits in maxima
  - Large number of fakes with high occupancy
  - Time increase with occupancy

Implementing new machine learning algorithm RNN

- Pattern recognition to discriminate muons hits from background hits exploiting info throughout layers
- Standard current alg: **Hough Transform**
  - Each curve in the Hough space represents the family of lines passing through a reconstructed point
  - We define a 2D histogram in the Hough space
  - Segments are represented by maxima (ρ_0, θ_0)

https://doi.org/10.1088/1748-0221/19/06/P06029

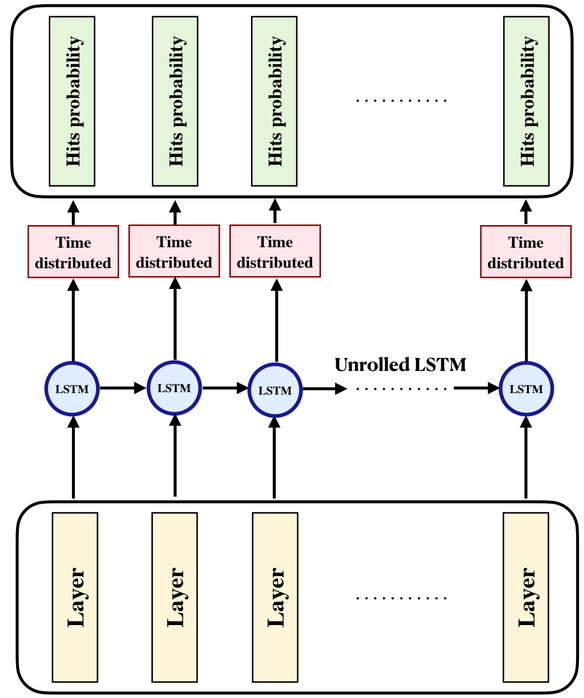


Pattern recognition with RNN

- **RNN are optimized to identify sequences, and work with sparse data**
- Results on the standalone G4 sim for all the levels of bakcground
  - cluster position reconstructed with DNN

Input variables: 2D vector with hits position in a layer (using padding) for each layer

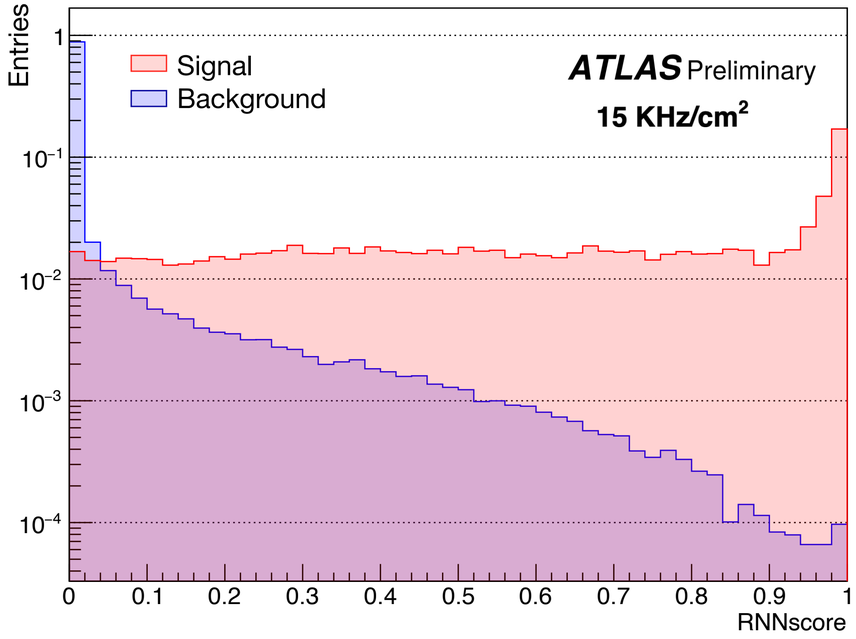
Label: bool variable identifying layers with track hits + vector with info on hit association to a track

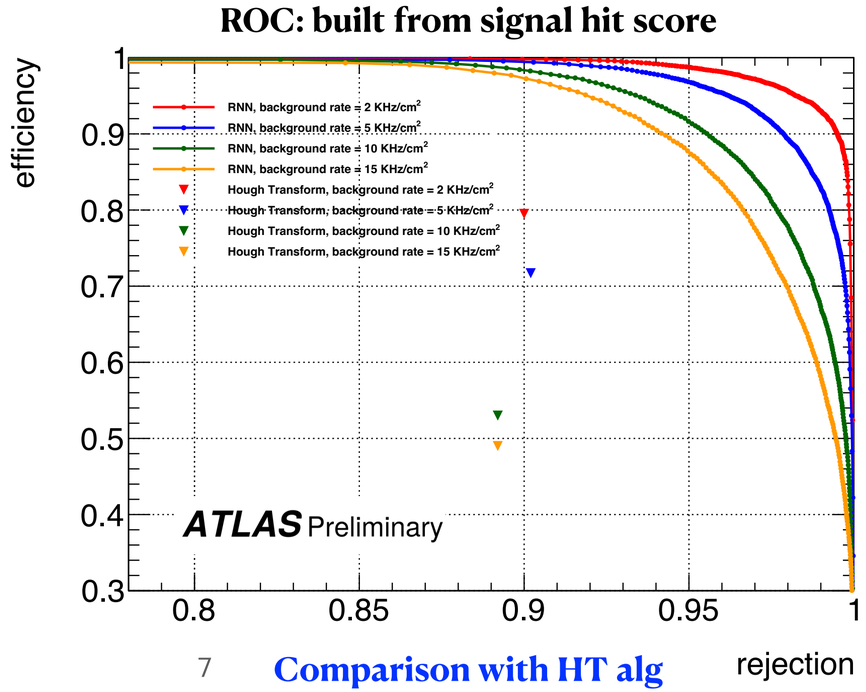


Single hit score: output for each hit in the event

0 →background hits

1 →muon track hits





Comparison with HT alg

Pattern recognition with RNN

Batch size 1	Inference time (s)	CPU load/core	GPU load
CPU 1 core ONNX	$1.5 \cdot 10^{-3}$	100 %	
CPU 10 core ONNX	$1 \cdot 10^{-3}$	100 %	
GPU TensorRT	$1 \cdot 10^{-2}$		23 %

Batch size 1000	Inference time (s)	CPU load/core	GPU load
CPU 1 core ONNX	$8.5 \cdot 10^{-1}$	100 %	
CPU 10 core ONNX	$1 \cdot 10^{-1}$	100 %	
GPU TensorRT	$2 \cdot 10^{-2}$		50 %

(Batch size →number of features set processed in parallel)

- Measurement of **inference time VS batch_size** on CPU and GPU
  - Using **ONNX for CPU optimization**
  - Using **TensorRT for GPU optimization**

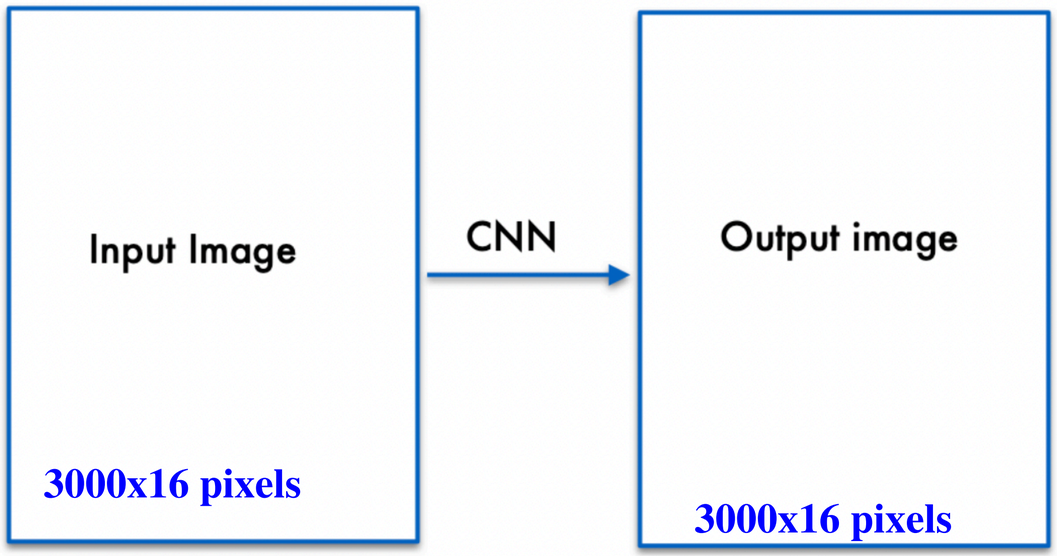
- For small batch size CPUs with ONNX are preferable
  - TensorRT best at optimizing GPU usage at large batch sizes
- Latency per event on GPU with TensorRT $\sim 10\mu s$ for $batch\_size \sim 4096$
- CPUs and GPUs are already within HLT requirements

- Total Parameters: 1.3M
- Also tested on different models with different #parameters (x1.6 and x2.5 parameters)
  - Inference time increases up to 25 %

- It was supposed to be possible to include it in the Vitis-AI workflow
  - Could work with the hls4ml workflow (studies ongoing)

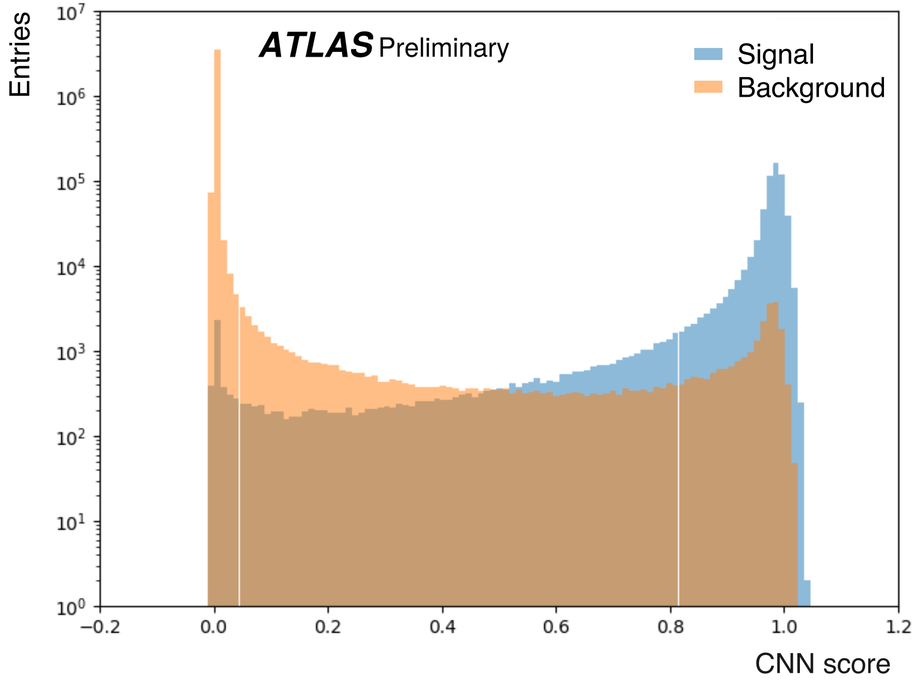
CNN for pattern recognition

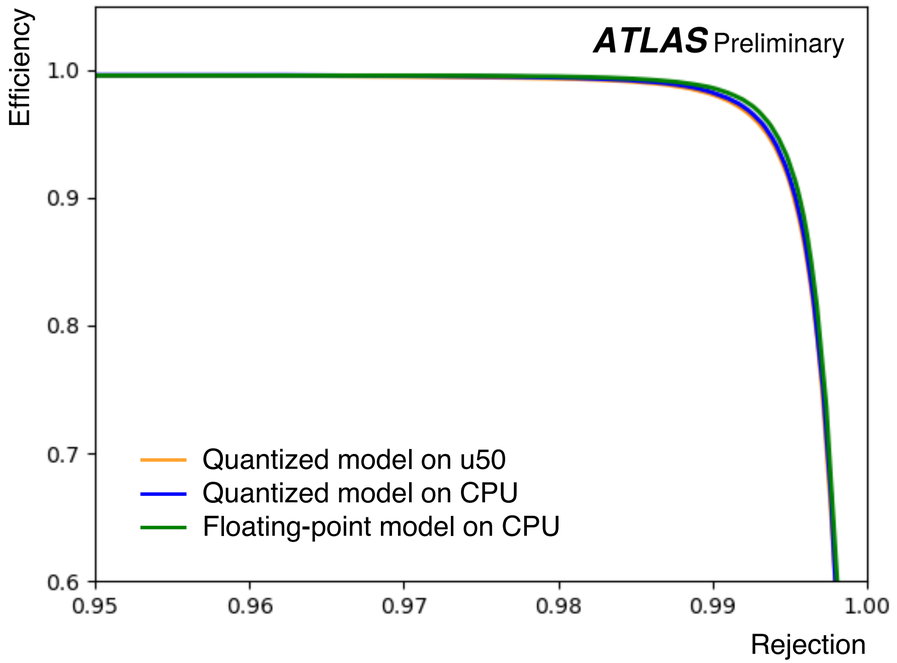
- RNN models not supported for FPGA inference by Xilinx (for the moment)
- To test ML algs for pattern recognition → development of **convolutional neural networks**
- Using same dataset as for RNN without the clusterization step →images built directly from firing strips



CNN → pre-processing step to built images (to be estimated)

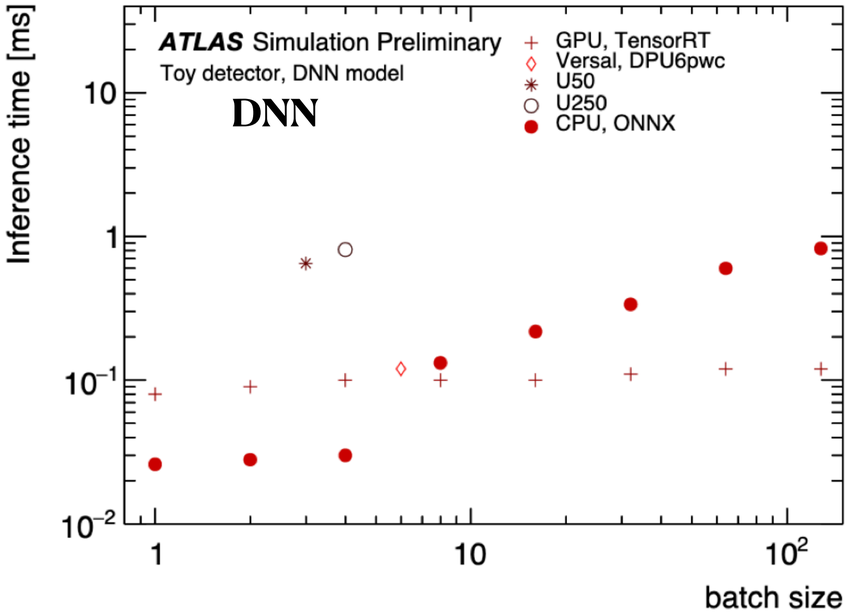
Not ideal for trigger → existing algs working directly on sparse data (RNN)

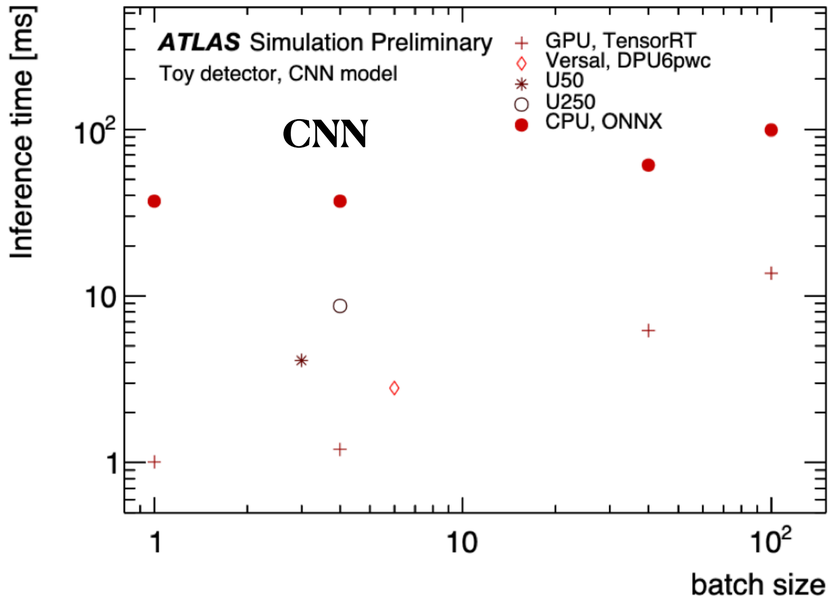




Timing and performance on CPU/GPU/FPGA

- DNN and CNN models successfully tested on CPU, GPU and several FPGAs
- Comparing CPU (ONNX) / GPU (TensorRT) / FPGA (Vitis AI) for DNN and CNN inference
  - Batch size is ~ fixed in the case of the FPGA, free for CPU/GPU
- VCK5000 can operate with batch sizes 4,6,8
  - FPGA times are not a simulation, are real processing times obtained on the U50, U250 and Versal VCK5000





XILINX VITIS™ AI

- Model optimization (pruning) not used yet
- **Quantization**: model parameters from float32 to int8
- Compiler transforms the model in a set of instructions and a dataflow model

- Overall CPU already meets the requirement imposed by the HLT latency
- Study on CPU load needs to be done, as well as the power dissipations

Conclusions

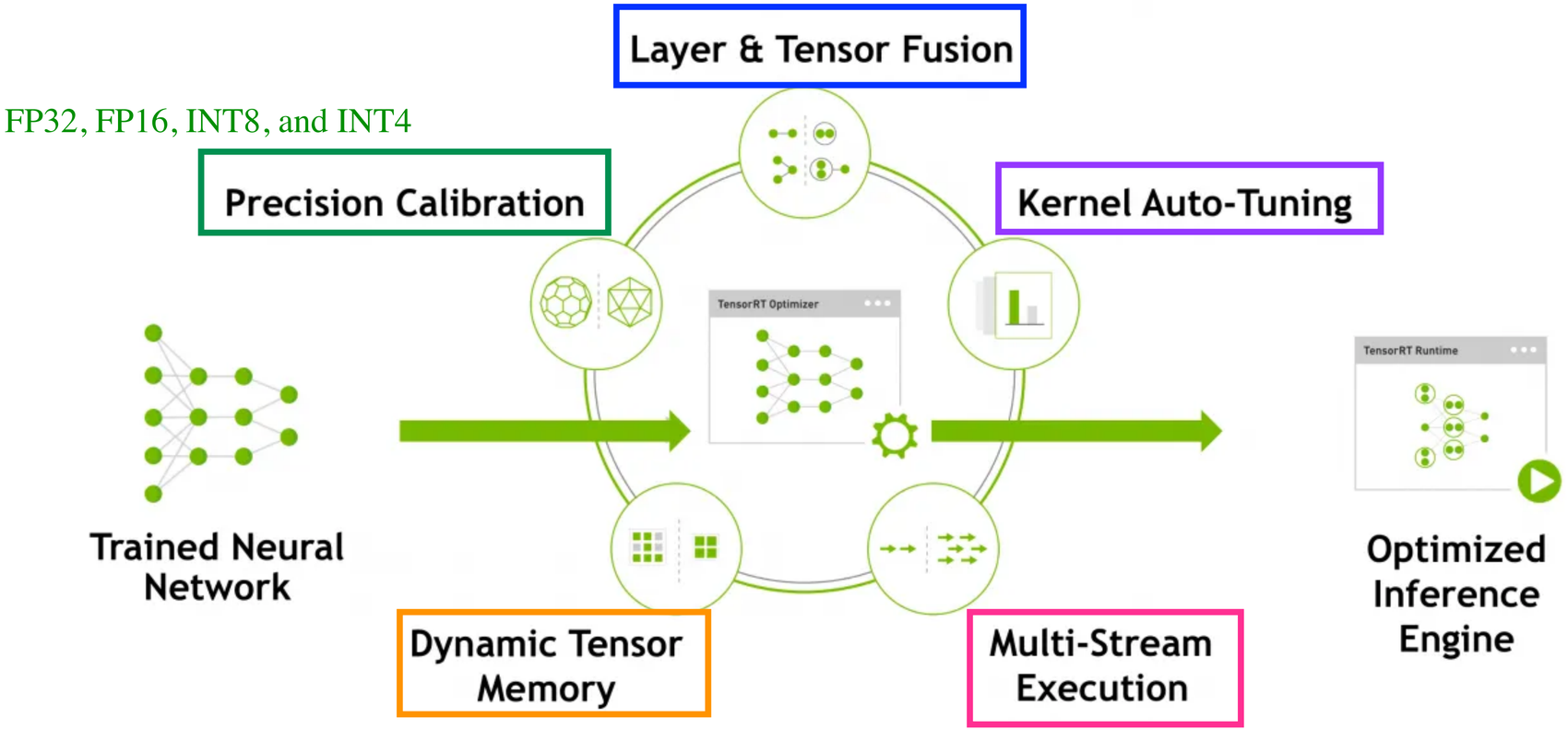
- Study of new possible algorithms to be implemented in the **muon HLT algorithms**
  - Maintain good performance in high occupancy environment in terms of residuals (DNN) and efficiency-rejection (RNN)
  - **DNN and CNN successfully tested on FPGA**
    - O(few ms) achieved for inference time
  - RNN has very good performances in terms of rejection and efficiency
    - to be compared with offline Hough-transform in terms of timing
  - No test on FPGA : LSTM layers currently not supported, only available in Demo Networks on specific FPGAs
      - Implementation with hls4ml on going

Thank you for your attention!

Back-up slides

TensorRT

- **Deep Learning Model Support:** wide range of deep learning frameworks including TensorFlow, PyTorch, ONNX, and Caffe
- **Plugin Architecture:** TensorRT's functionality extendable by implementing custom layers and operations using its plugin architecture. This enables support for specialized layers and custom network architectures



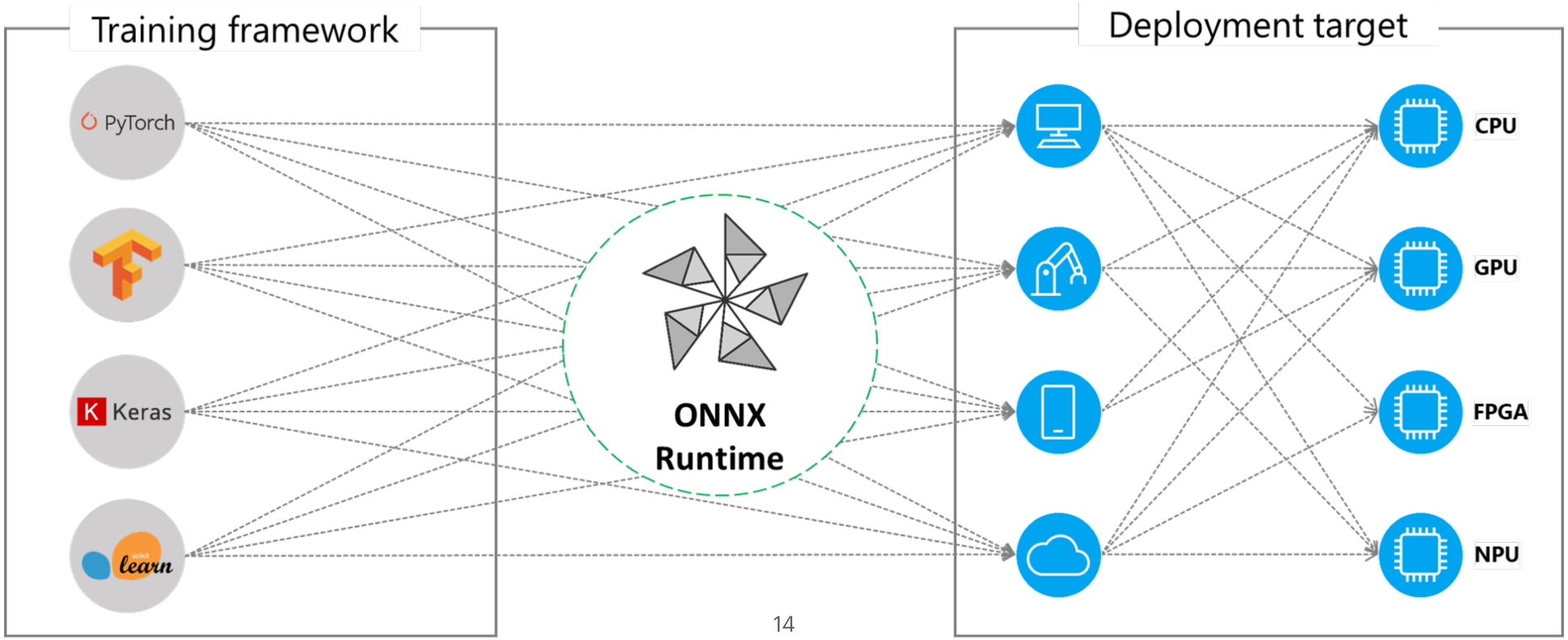
- Allocates memory to tensor only during usage
  - reducing memory footprints
  - avoiding allocation overhead

ONNX

- **ONNX Runtime:** high-performance execution engine specifically designed to efficiently run ONNX models
  - open-source project
- ONNX Runtime automatically performs optimizations such as weight quantization, layer fusion, and operation parallelization

Deep Learning Model Support: wide range of deep learning frameworks including TensorFlow, PyTorch, and Caffe

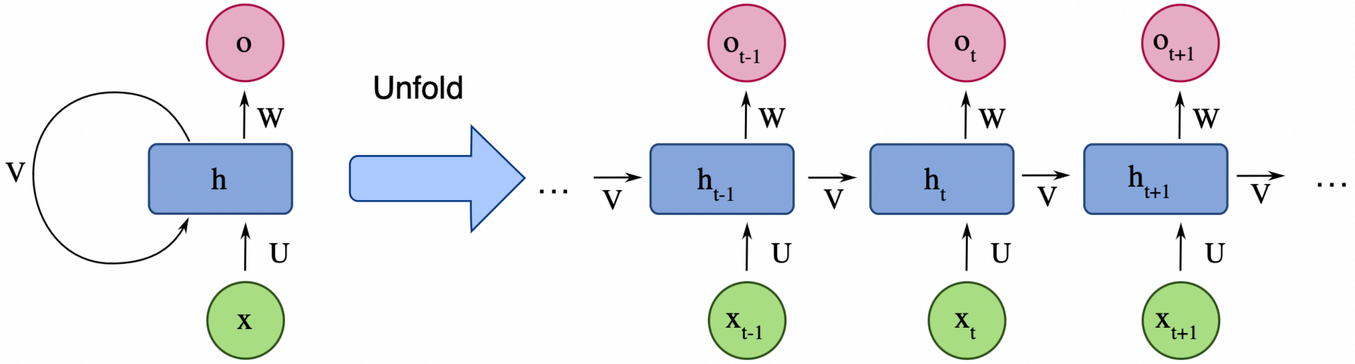
Cross-Platform: compatible with a wide range of hardware and software platforms, including edge devices, CPUs, GPUs, and specialized accelerators



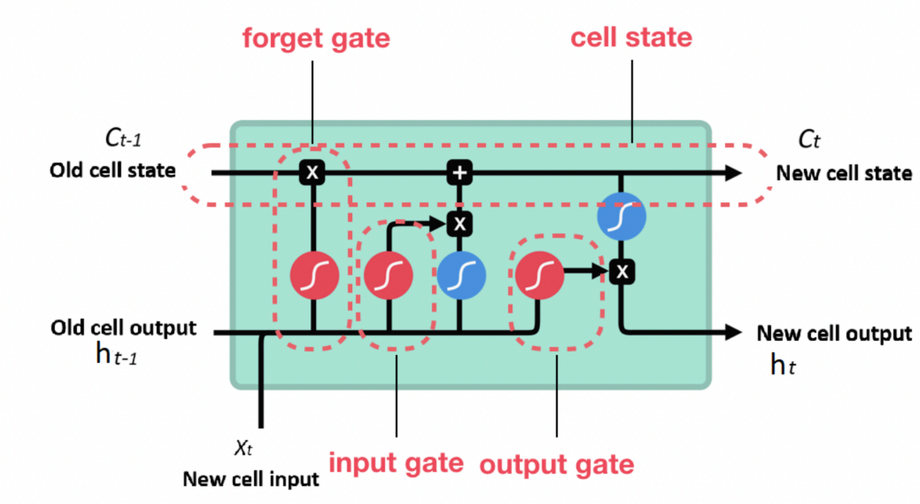
Recurrent neural network

Recurrent Neural Network

- Designed to deal with sequential data
- At each step, it takes as input the input at time t and the output at time t-1



LSTM



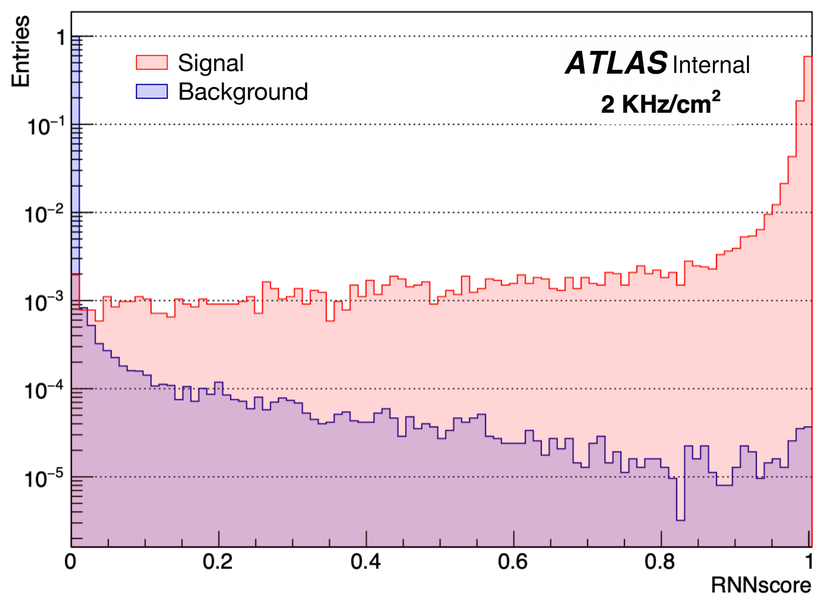
- Recurrent Neural Network have problems with long term memory
- The more time steps, the more the back-propagation gradient vanishes -> NN does not learn long term dependencies

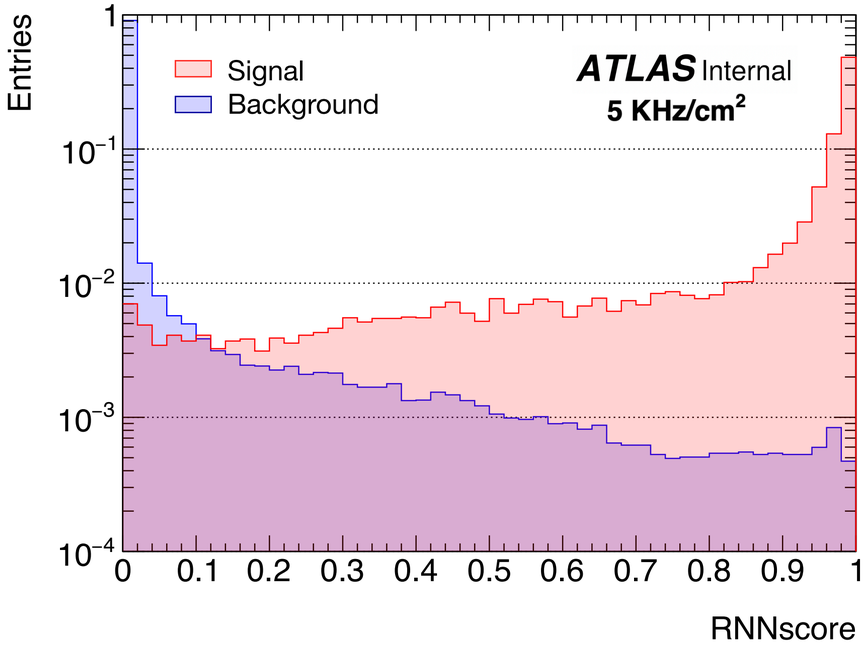
LSTM networks can solve the long term memory problem

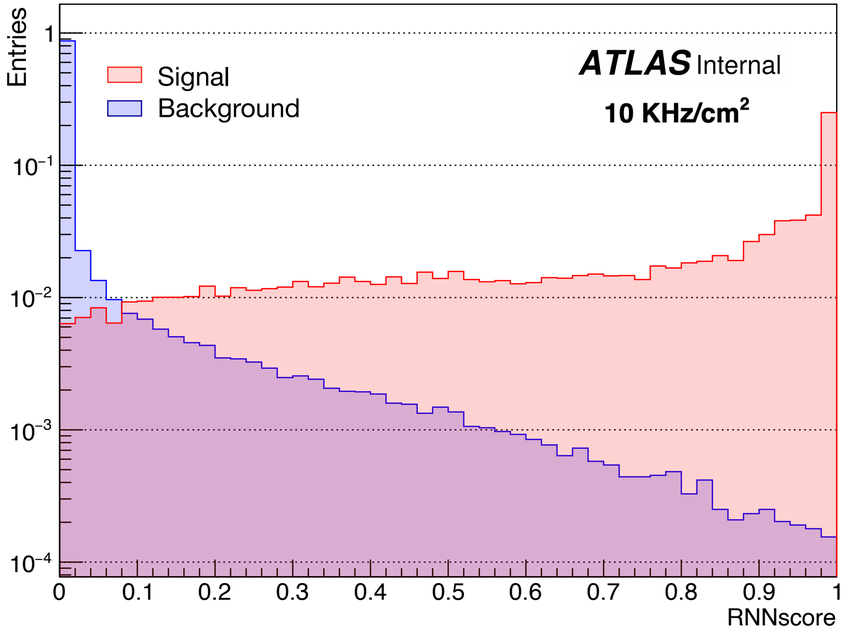
CNN structure

```
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input_1 (InputLayer)        [(None, 3000, 16, 1)]     0

 conv1 (Conv2D)              (None, 3000, 16, 2)       194

 pool1 (MaxPooling2D)        (None, 1500, 16, 2)       0

 conv2 (Conv2D)              (None, 1500, 16, 4)       100

 conv3 (Conv2D)              (None, 1500, 16, 8)       392

 pool3 (MaxPooling2D)        (None, 750, 16, 8)        0

 conv4 (Conv2D)              (None, 750, 16, 16)       6160

 pool4 (MaxPooling2D)        (None, 750, 8, 16)        0

 conv5 (Conv2D)              (None, 750, 8, 16)        32784

 up_sampling2d (UpSampling2D  (None, 750, 16, 16)      0
 )

 conv6 (Conv2D)              (None, 750, 16, 16)       2064

 up_sampling2d_1 (UpSampling  (None, 1500, 16, 16)     0
 2D)

 conv7 (Conv2D)              (None, 1500, 16, 8)       1032

 up_sampling2d_2 (UpSampling  (None, 3000, 16, 8)      0
 2D)

 conv8 (Conv2D)              (None, 3000, 16, 2)       130

 output (Conv2D)             (None, 3000, 16, 1)       7

=================================================================
Total params: 42,863
Trainable params: 42,863
```

RNN results


Entries
Signal
Background
ATLAS Internal
2 KHz/cm²
RNNscore


Entries
Signal
Background
ATLAS Internal
5 KHz/cm²
RNNscore


Entries
Signal
Background
ATLAS Internal
10 KHz/cm²
RNNscore