Machine Learning for Real-Time Processing of ATLAS Liquid Argon Calorimeter Signals with FPGAs

Nairit Sur

CPPM - CNRS/IN2P3

on behalf of the ATLAS Liquid Argon Calorimeter Group

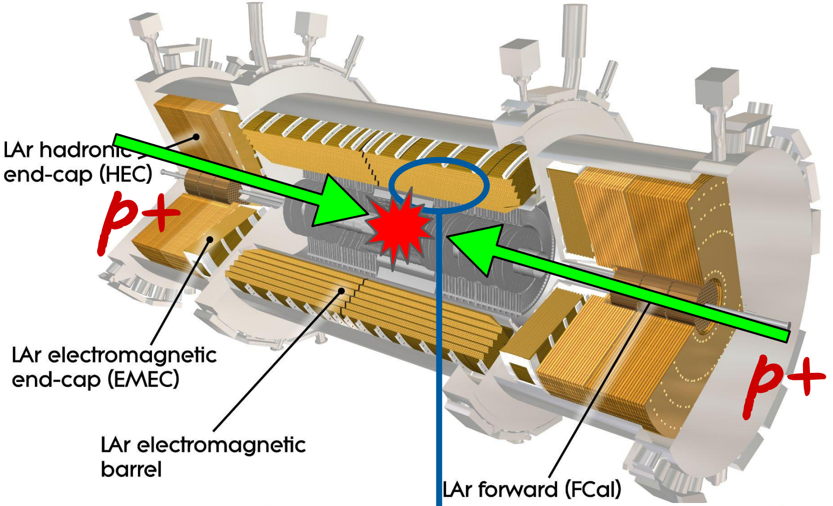
The Liquid Argon Calorimeter:

A crucial component of the ATLAS detector

- So far, 160 fb^{-1} p-p collision data were reconstructed with high quality and precision

- Designed to measure the **time**, **position**, and **energy** deposited by **electrons** and **photons**, and in addition, **hadrons** in the forward region

- ~**180K** readout channels - Lead, copper, and tungsten as absorbers, cryogenically cooled liquid argon as active material



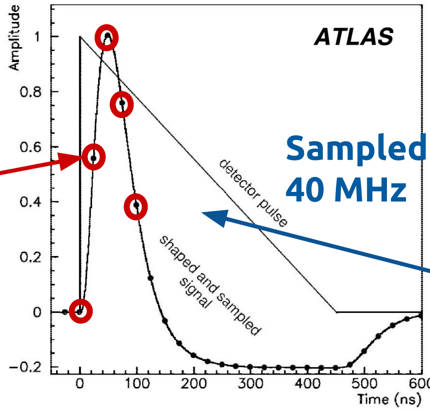
Energy from Optimal-Filter (OF)

n = 5 in this talk

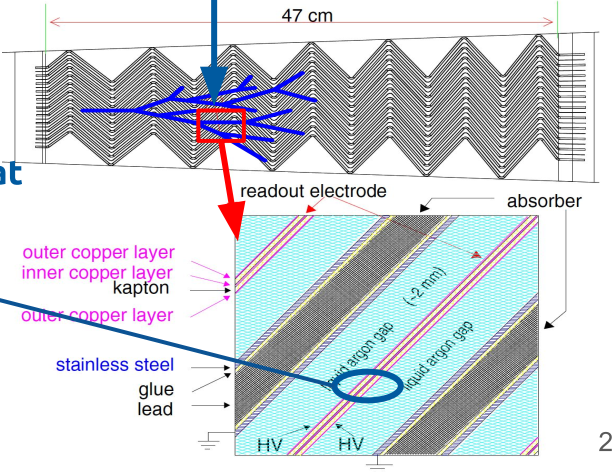
$$E(t) = \sum_{i=t}^{t+n} a_i \cdot s_i$$

Pulse Samples

Pre-set coefficients (fit of the peak)



Sampled at 40 MHz



Towards HL-LHC

The high luminosity phase of the LHC (**HL-LHC**) will produce **140-200** simultaneous p-p interactions (pile-up), compared to the current value of around **40**

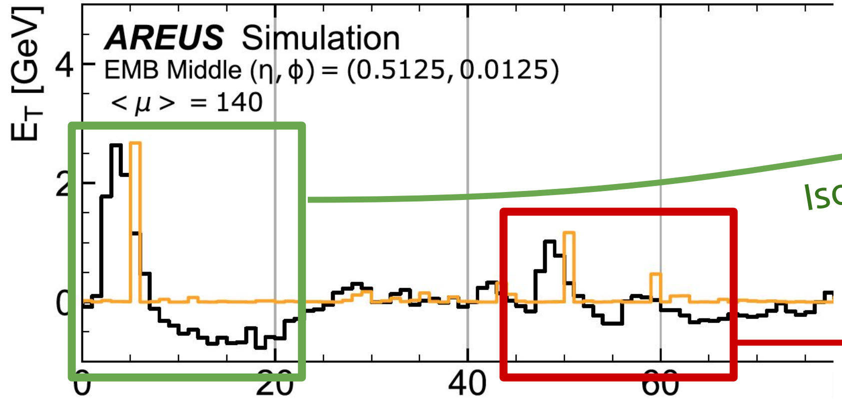
Energy deposits **continuously** sampled and digitized at 40 MHz :

⇒ requires peak finder/trigger (to select the correct BCIDs)

Real-time energies for triggers :

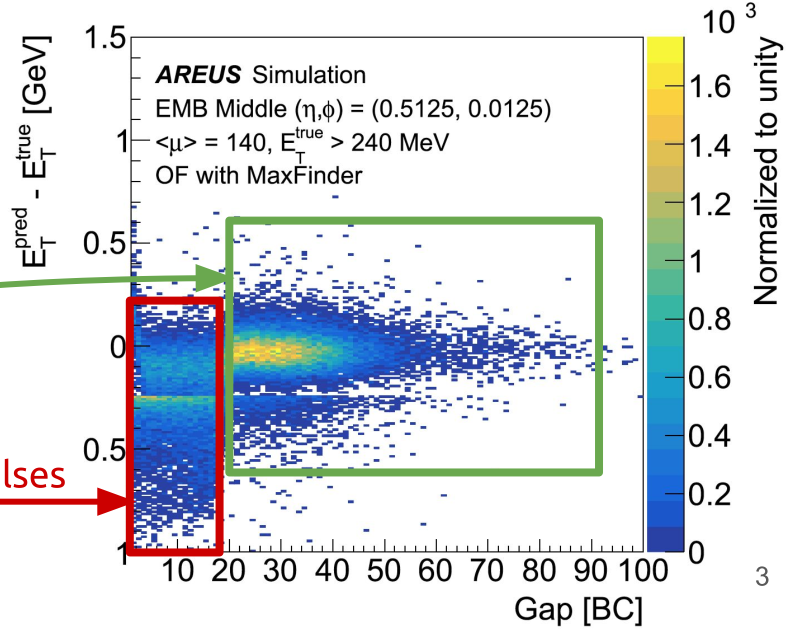
⇒ requires compact algorithms on high-end FPGAs

Legacy algorithms cannot compensate for past events affecting the present



Isolated pulse

Overlapping pulses



LAr Upgrade

LAr cells

New off-detector electronics :

LAr Signal Processor (LASP)

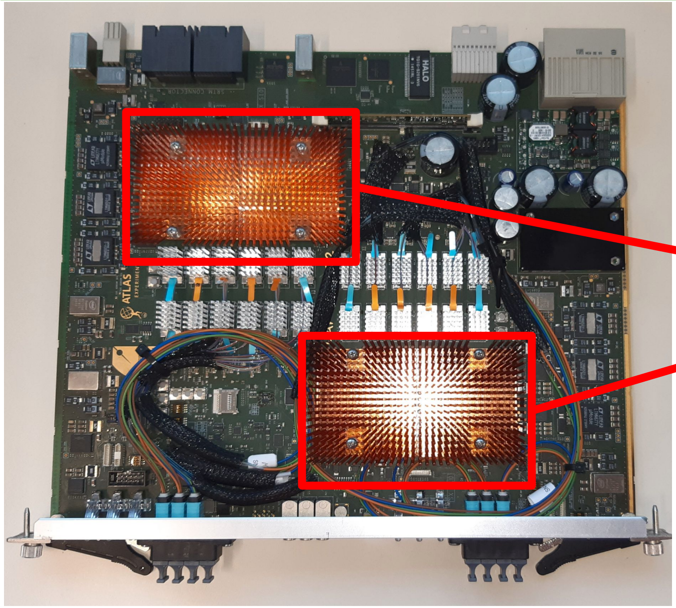
- Two FPGAs (Intel Agilex)
- ~Tb/s(~700 channels)
- ~300 boards

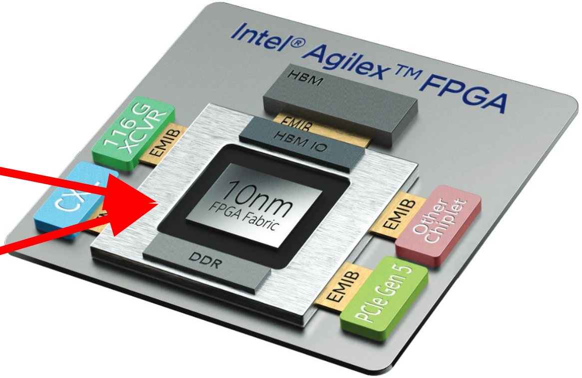
$E_{reconstructed}$

Upgrade of readout electronic chain for AI algorithms

- One FPGA should process **384 channels**
- About **150 ns** allocated latency for energy computation

H/t: Fabrice Gensolen (CPPM)





Test boards were built with Stratix10, but production ones will use higher grade Agilex FPGAs

Energy reconstruction with NNs

[Comput Softw Big Sci **5**, 19 (2021)]

[LArCalo Upgrade Public Results]

Two neural network types are tested:

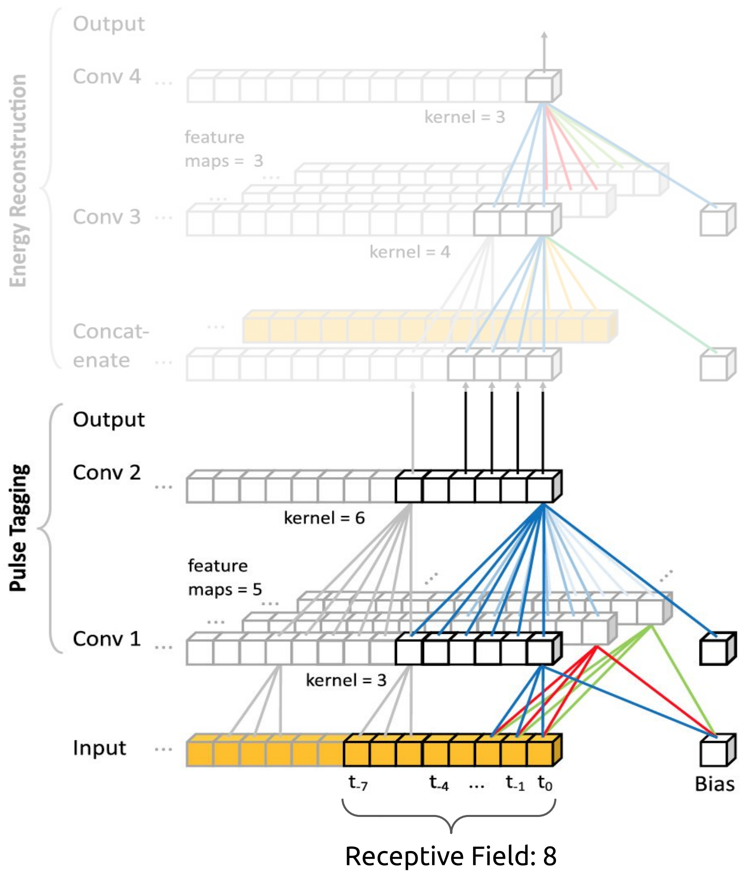
Convolutional Neural Networks (CNN) - Developed by the TU Dresden group

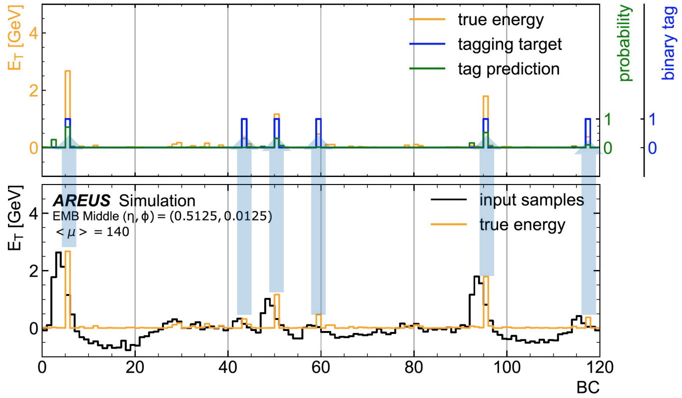
and

Recurrent Neural Networks (RNN) - Developed by the CPPM group

CNN step 1: Pulse tagging

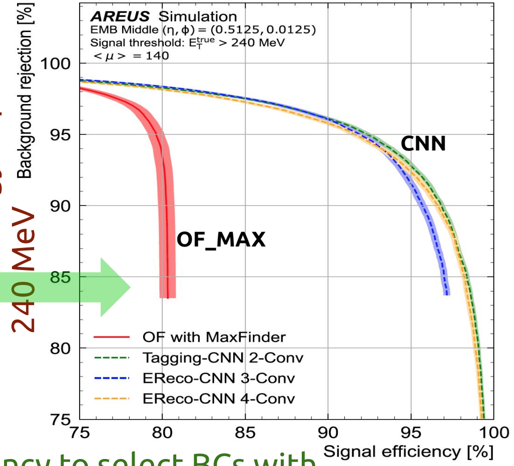
CNN for pulse tagging:
Trained to detect energy deposits 3σ above noise (240 MeV) using pulse samples for 8 bunch crossings





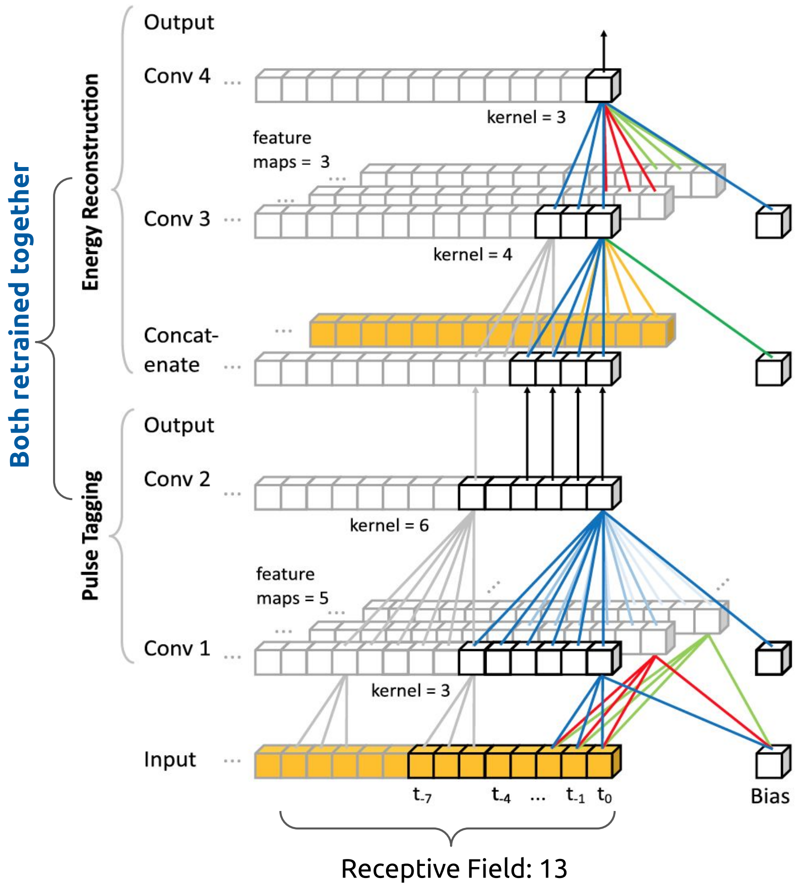
significant gain in efficiency with CNN tagger with respect to OF with MaxFinder

Efficiency to reject BCs with energy deposits < 240 MeV



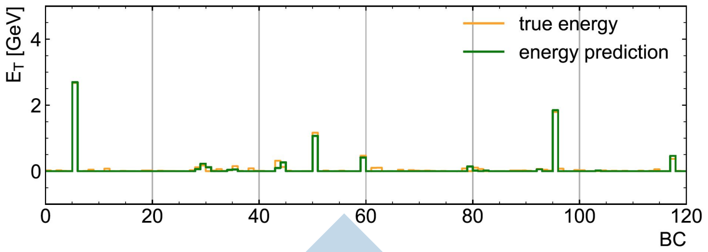
Efficiency to select BCs with energy deposits > 240 MeV

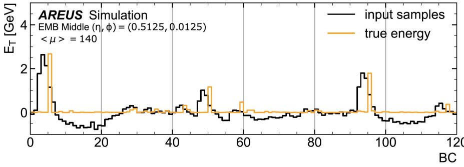
CNN step 2: Energy inference



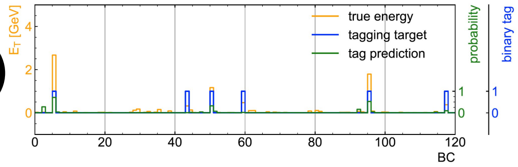
CNN for energy reconstruction:

Energy reconstruction layers are added to the tagging layers and retrained together





$\otimes$



4-Conv CNN

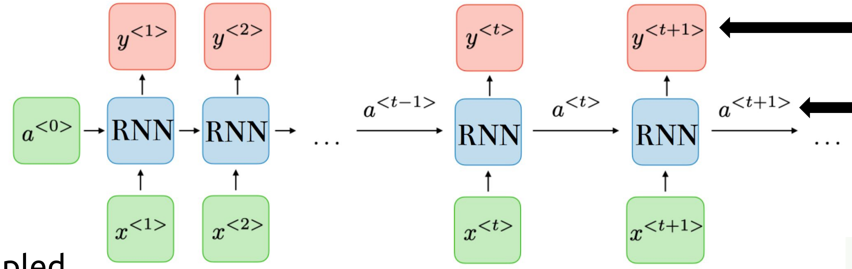
- **2** layers for energy reconstruction
- Receptive field: **13** BC
- **88** parameters in total

3-Conv CNN

- **1** layer for energy reconstruction
- Receptive field: **28** BC
- **94** parameters in total

Recurrent Neural Networks

Designed for handling sequential data, RNNs consist of internal neural networks that process new input combined with the past processed state



Extracts energy at all times (in-built pulse tagging)

Decides what information to forward in time

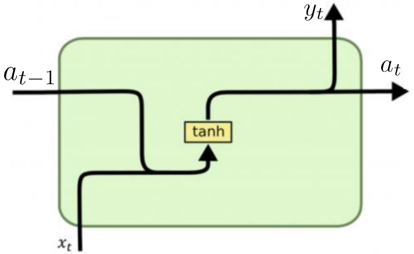
⇒ Pulse span 20-30 BC, overlap …

Pulse sampled at every BC

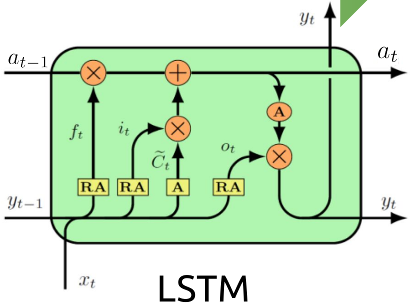
Better performance, more stability in time

Two RNN internal architectures explored:

- Optimised for smaller number of parameters
- Long Short-Term Memory (LSTM) - 10 internal dimensions
- Vanilla-RNN - 8 internal dimensions



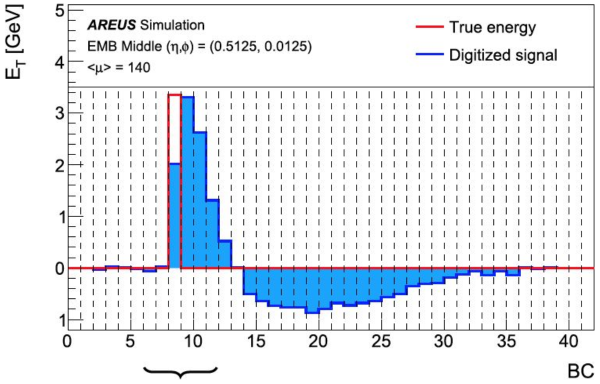
Vanilla-RNN
(89 parameters)

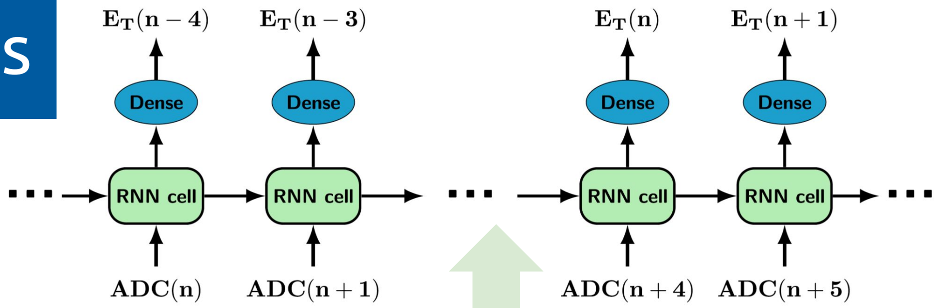


LSTM
(491 parameters)

Higher complexity, bigger size on hardware

RNN applications: two methods

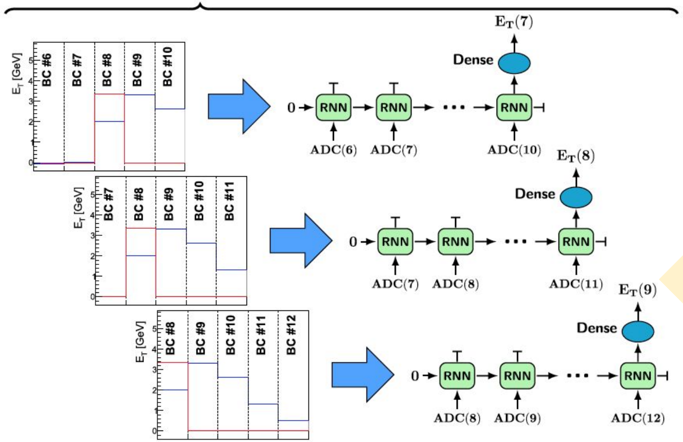




Single Cell Method:

✔ Long range correction, full signal is processed in a stream

✘ Significant amount of complexity needed to process data in time (LSTM only)



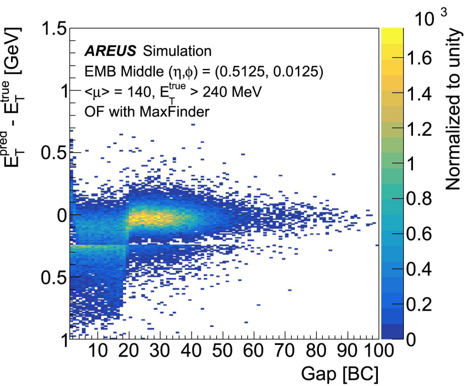
Sliding window Method (5 BC):

✔ Robust against long-lived effects due to unforeseen behaviour of the detector, simpler training

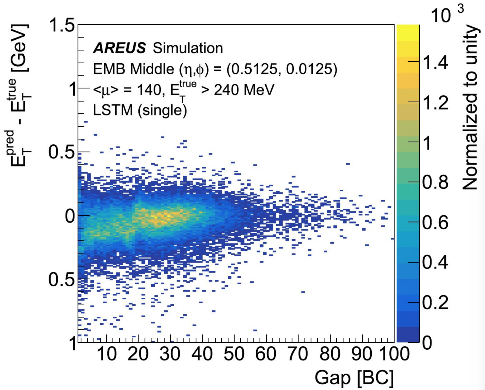
✘ Short range correction only (1 BC in the past)

Performance : HL-LHC condition with pileup of 140

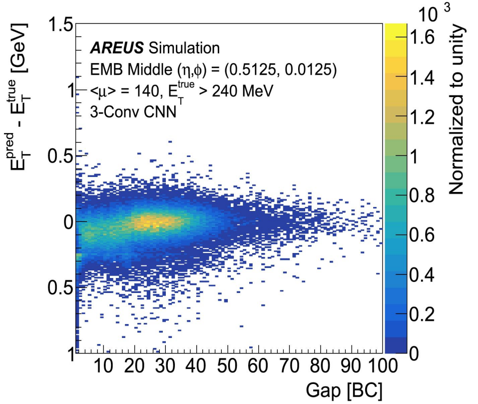
Comparisons on single LAr cell simulations (***AREUS*** software)



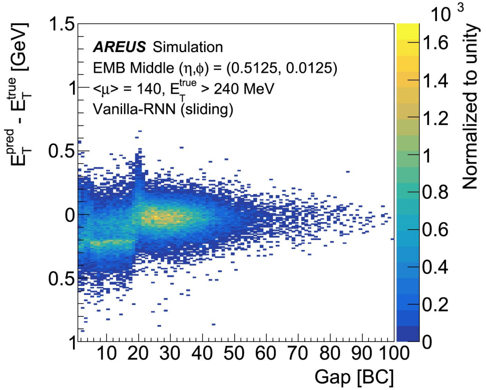
Legacy algorithm:
5 BC in the peak



LSTM (single cell):
5 BC in the peak, ∞ in the past



3-conv CNN:
5 BC in the peak, 23 in the past



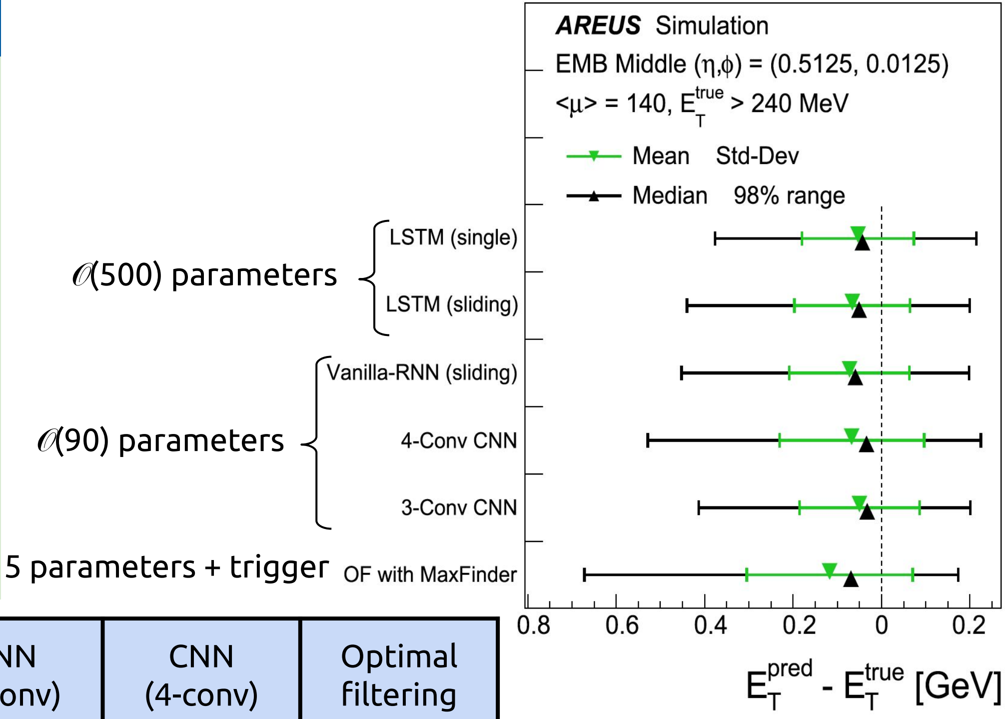
Vanilla (sliding window):
4 BC in the peak, 1 in the past

- Legacy algorithm exhibits big distribution tails especially at low gap
- The tails are reduced significantly with all the new NN methods

NN Performance :
HL-LHC condition with pileup of 140

[Comput Softw Big Sci **5**, 19 (2021)]

- Overall better energy scale and resolution for all NNs with respect to OF_MAX
  - Lower tails as seen from the 98% median range
- All NNs optimized to reduce as much as possible the required computing resources
  - While keeping good performance
- LSTM perform best but have a larger number of parameters
  - Harder to fit in the FPGAs



Algorithm	LSTM (single)	LSTM (sliding)	Vanilla (sliding)	CNN (3-conv)	CNN (4-conv)	Optimal filtering
Number of parameters	491	491	89	94	88	5
MAC units	480	2360	368	87	78	5

Implementation on FPGAs

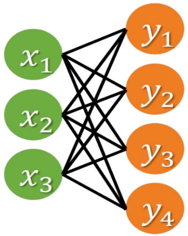
ANN model in Keras

- Set of weights optimised by training
- architecture(layers, dimensions, …)
- Mathematical operations

Converter (HLS and VHDL)

FPGA firmware

- **ALM**: adaptive logic modules
- **DSP**: digital signal processors
- Fixed-point arithmetic, LUT for non-linear functions



$$\begin{pmatrix} y_1 \\ y_2 \\ y_3 \\ y_4 \end{pmatrix} = A\left(\begin{pmatrix} w_{11} & w_{12} & w_{13} \\ w_{21} & w_{22} & w_{23} \\ w_{31} & w_{32} & w_{33} \\ w_{41} & w_{42} & w_{43} \end{pmatrix} \times \begin{pmatrix} x_1 \\ x_2 \\ x_3 \end{pmatrix} + \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ b_4 \end{pmatrix}\right)$$

Activation function for non-linear element operations

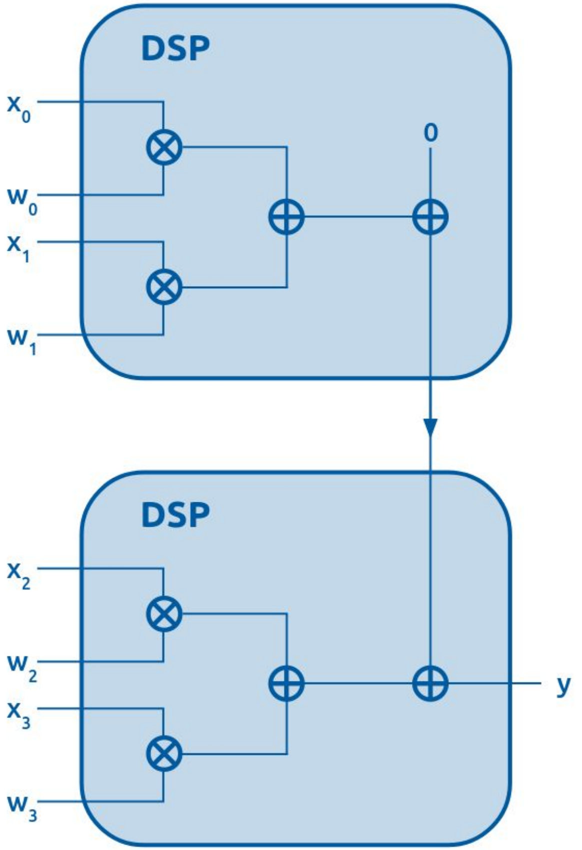
CNN Firmware Implementation

- CNN implemented in **VHDL** with full configurability -
  - Configurable layer building blocks: **I/O, activation functions**
  - Configurable component connections: **Kernel sizes, filters per layer, dilation**

- Model architecture parameters automatically extracted from Keras output

- Designed to support pipelining and time-division multiplexing:
  - Runs at 12 times the ADC frequency and processes 12 detector cells cyclically

- Calculations can be done in 18-bit fixed point numbers

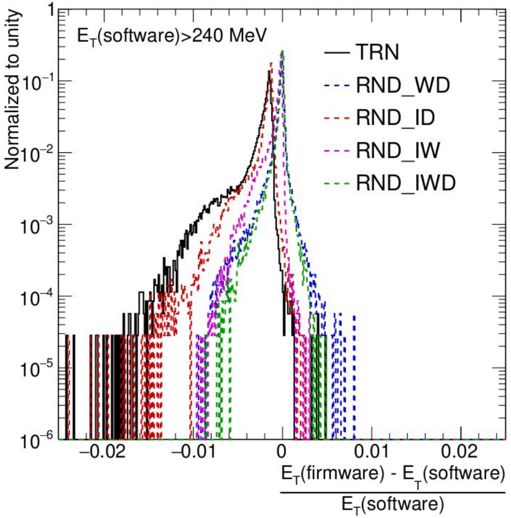


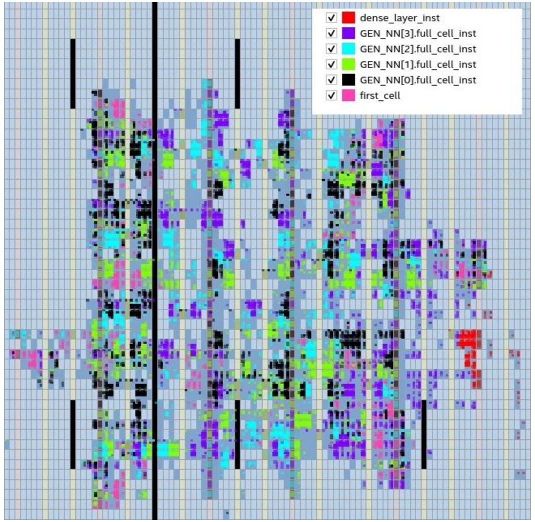
RNN Firmware Implementation: HLS

- RNN initially implemented in **Intel HLS** which adds an additional level of abstraction
  - Advantages: **fast and efficient optimisation of parameters and implementation**

- Optimisation of arithmetic operations:
  - Different quantisation schemes tested for optimal fixed point operation performance
  - truncation(**TRN**) vs. rounding (**RND**) of internal type(**I**), I/O type(**D**), and weight type(**W**) data categories

- Disadvantages:
  - Cannot achieve target frequency and resource utilisation constraints when several instances of NNs are placed on the same FPGA
  - During compilation, each instance gets different placement shape, which, moreover, gets randomised between compilations → complicates optimisation of timing-critical paths needed to reach higher frequency and multiplexing





HLS placement

RNN Firmware Implementation: VHDL

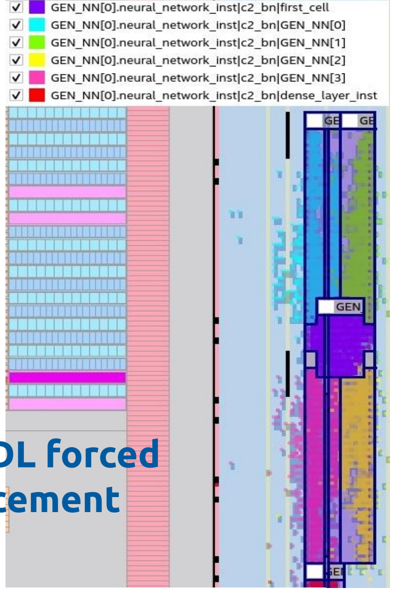
[*JINST* **18** P05017 (2023)]

- Implementation in **VHDL** enables finer optimisations which are not tunable in HLS

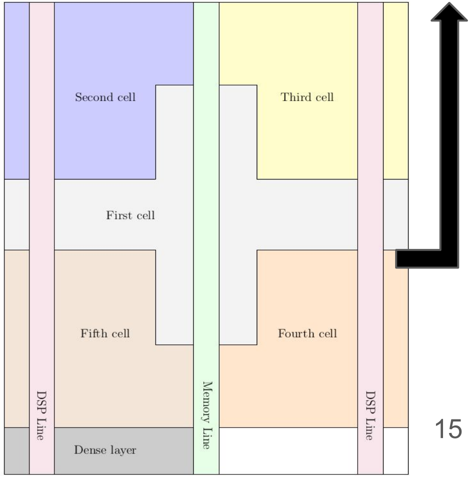
- Reuse of common computations:
  - Common computations in cells differing only in time are propagated between each other at the proper time instead of recalculation, unnecessary calculation for certain cells removed
  - **Reduction of DSP usage by 10%, ALM usage by 21%**

- Optimised placements:
  - Placement of 5 network cells designed to minimise distance between them
  - **Placement constraints force all NN instances to have the same shape**

- Incremental compilation of multi-partitioned firmware design:
  - Preserve partitions (1 NN instance each) that do not exhibit timing violations and recompile the rest until the target frequency is reached for the whole design → reached **560 MHz** with **28 NN** instances within 4 compilations



VHDL forced placement



Estimation of FPGA Resource Usage

- Both CNN and RNN implementations in VHDL satisfy ATLAS requirements:
  - Trigger latency ≈ **150 ns**
  - Process **384 channels per FPGA with multiplexing**
- Estimated resource usage based on Intel Quartus reports:

FPGA	Network	Multiplexing	Channels	F_{max} [MHz]	ALMs	DSPs
Stratix-10	**RNN (HLS)**	**10**	**370**	**393**	**90%**	**100%**
	RNN (VHDL)	**14**	**392**	**561**	**18%**	**66%**
	2-Conv CNN	**12**	**396**	**415**	**8%**	**28%**
	4-Conv CNN	**12**	**396**	**481**	**18%**	**27%**
Agilex	**2-Conv CNN**	**12**	**396**	**539**	**4%**	**13%**
	4-Conv CNN	**12**	**396**	**549**	**9%**	**12%**

Summary

- CNN and RNN networks outperform the optimal filtering algorithm for the energy reconstruction in the ATLAS LAr Calorimeter, particularly in the region with overlap between multiple pulses

- Studies to quantify the effect on object (electrons, photons) reconstruction and physics performance is underway using Phase-II montecarlo samples and Athena simulation.

- All networks are designed to reduce to a maximum the resource usage while keeping the performance

- CNN and Vanilla RNN are serious candidates that can fit the stringent requirements on the LASP firmware

- Network optimisations in VHDL allow reaching the requirements in terms of resource usage and latency
    - Testing on hardware (Stratix10 DevKits) started and shows good results
    - Need to check timing violations with all other components of the LASP firmware

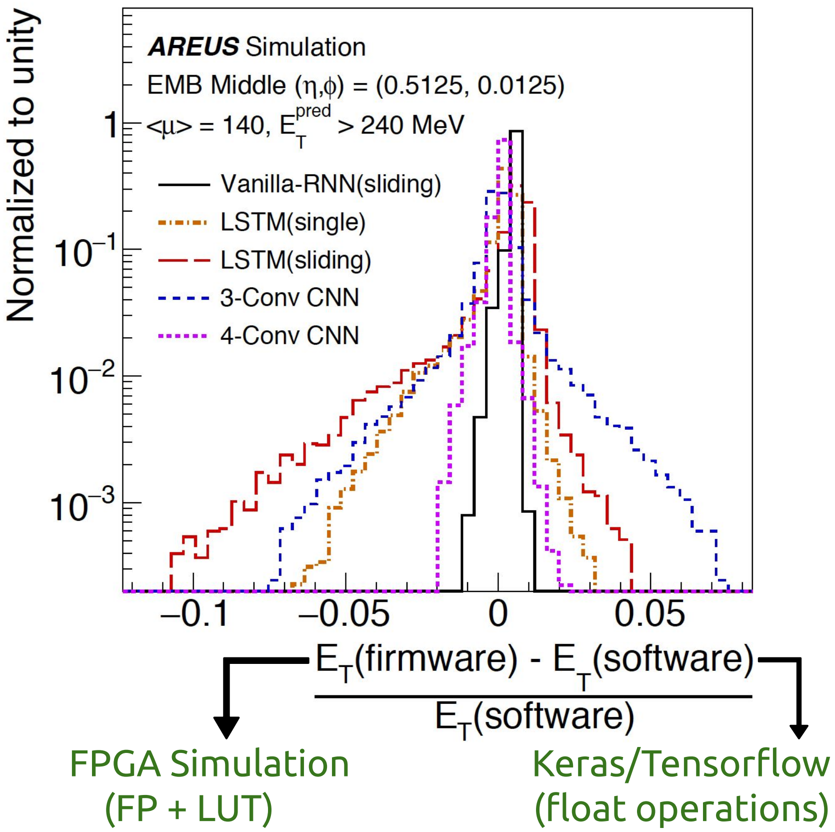
- Quantisation-aware training has proven to be effective in significantly reducing the required bit width for number representation on the FPGA, thereby minimising resource usage

DĚKUJU MOC !

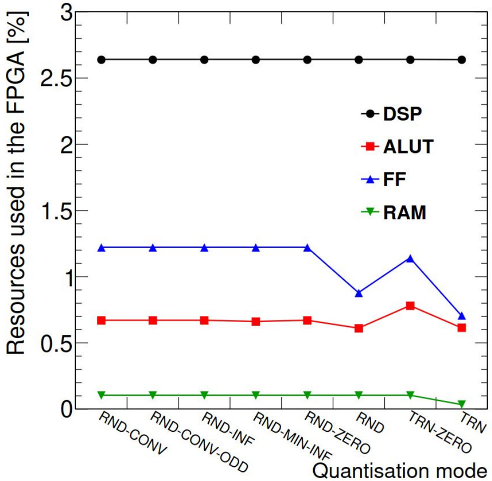
Backup

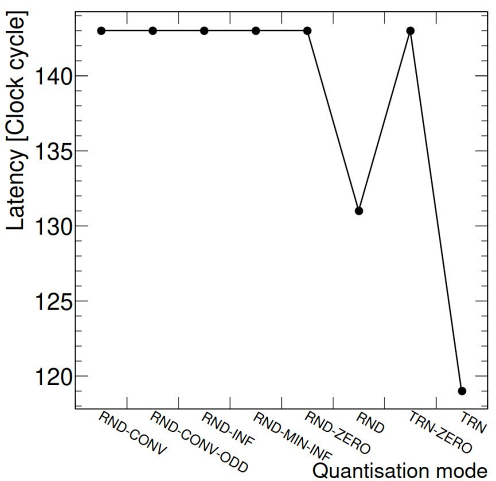
Firmware vs. Software

- Energy computed with Quartus simulation and verified on target
- **$\mathcal{O}$(1%) resolution** due to firmware approximation, viz. LUT for activation functions, Fixed point arithmetic



Quantisation mode optimisation for RNN





- Comparison of resource usage in terms of DSPs, arithmetic look-up tables (ALUT), flip-flops (FF), and random access memory (RAM) for different quantisation modes available in Intel HLS

- comparison of the transverse energy computed in firmware and the one computed in software, with a full floating point implementation, for the different quantisation modes