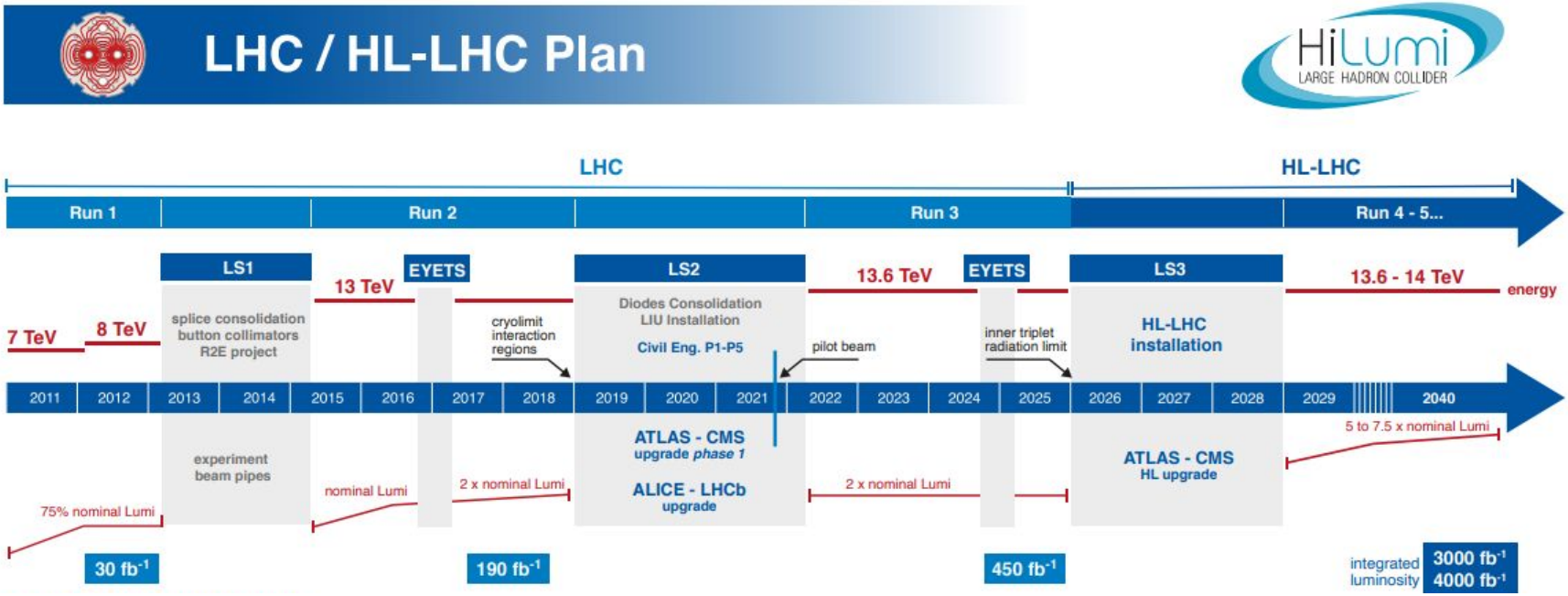
Embedded Recurrent Neural Networks on FPGAs for Real-Time Computation of the Energy Deposited in the ATLAS Liquid Argon Calorimeter (AIDAQ)

R&T Meeting
8/11/2023

Georges Aad
CPPM

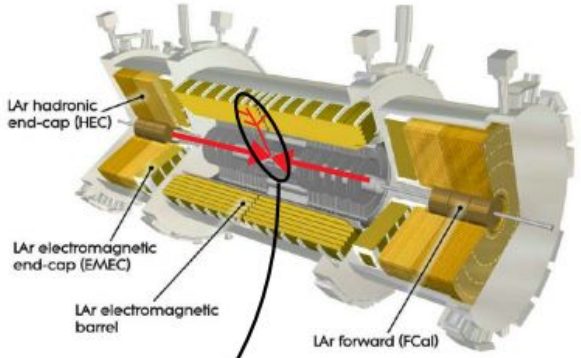
Introduction

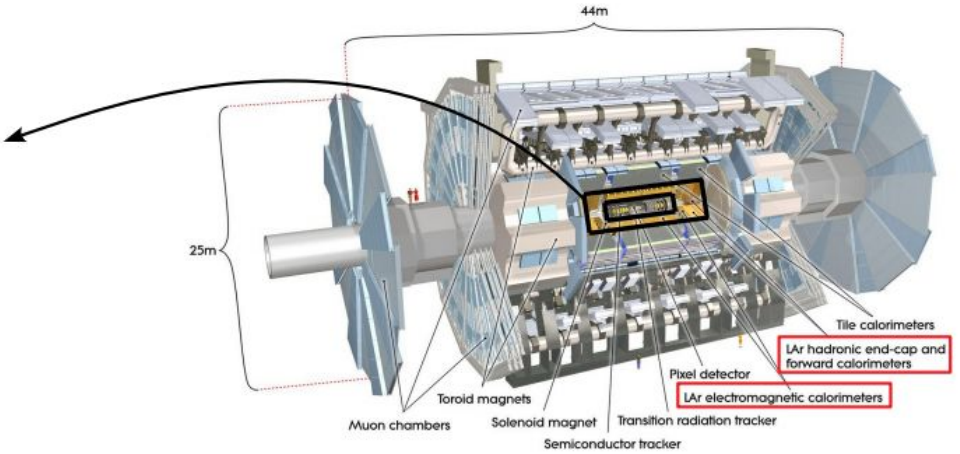
- LHC upgrade during the long shutdown starting 2026 leading to the HL-LHC
  - Increase the instantaneous luminosity by a factor 5 to 7
  - 140 to 200 simultaneous p-p collisions (pileup)
- ATLAS will be upgraded to cope with the HL-LHC conditions
  - Increase the first-level trigger rate from 100 kHz to 1MHz
  - New readout electronics for the liquid Argon calorimeter



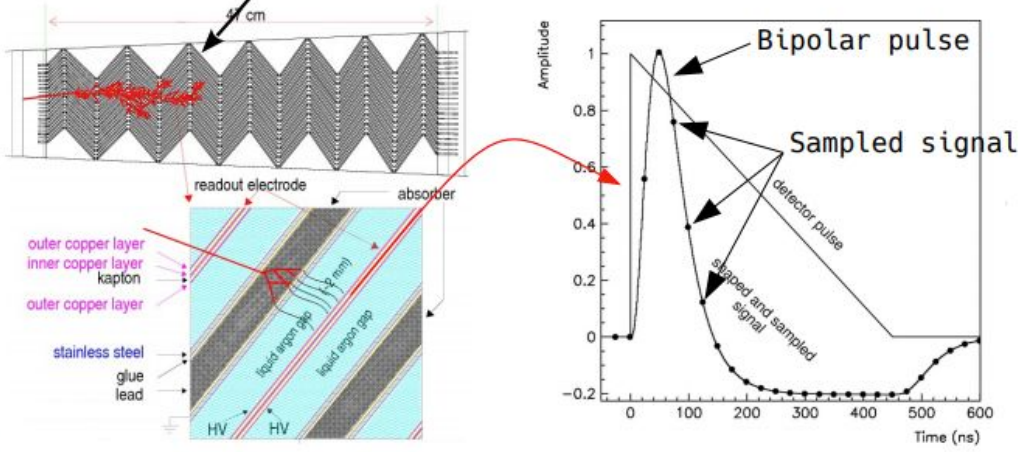
The ATLAS Liquid Argon Calorimeter

- Measures the energy of electromagnetically interacting particles mainly electrons and photons
- Trigger capabilities at the first level of triggering (implemented in hardware)
  - Very fast online processing of ~ 300 Tb/s of data
  - Use of FPGA technology for data processing





180000 channels

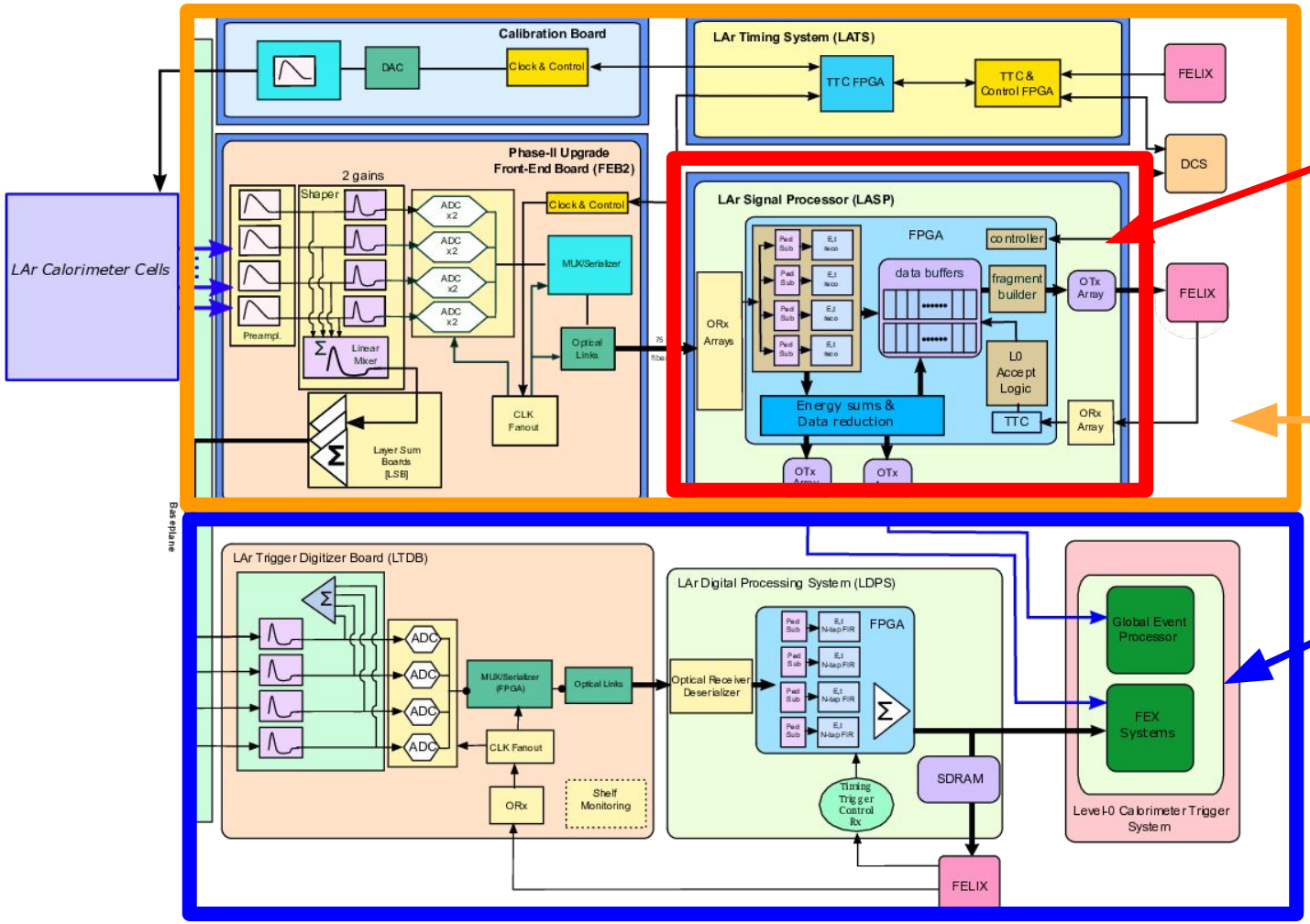


- Electronic signal with amplitude corresponding to the deposited energy in the calorimeter
- Shaped and sampled at 40 MHz
- Samples used to compute the deposited energy

LAr Upgrade

CERN-LHCC-2017-020

- Full electronics of the readout path will be exchanged
  - New on-detector electronics that will digitize the signal at 40 MHz and send it to the backend
  - New off-detector electronics to compute the energy at 40 MHz



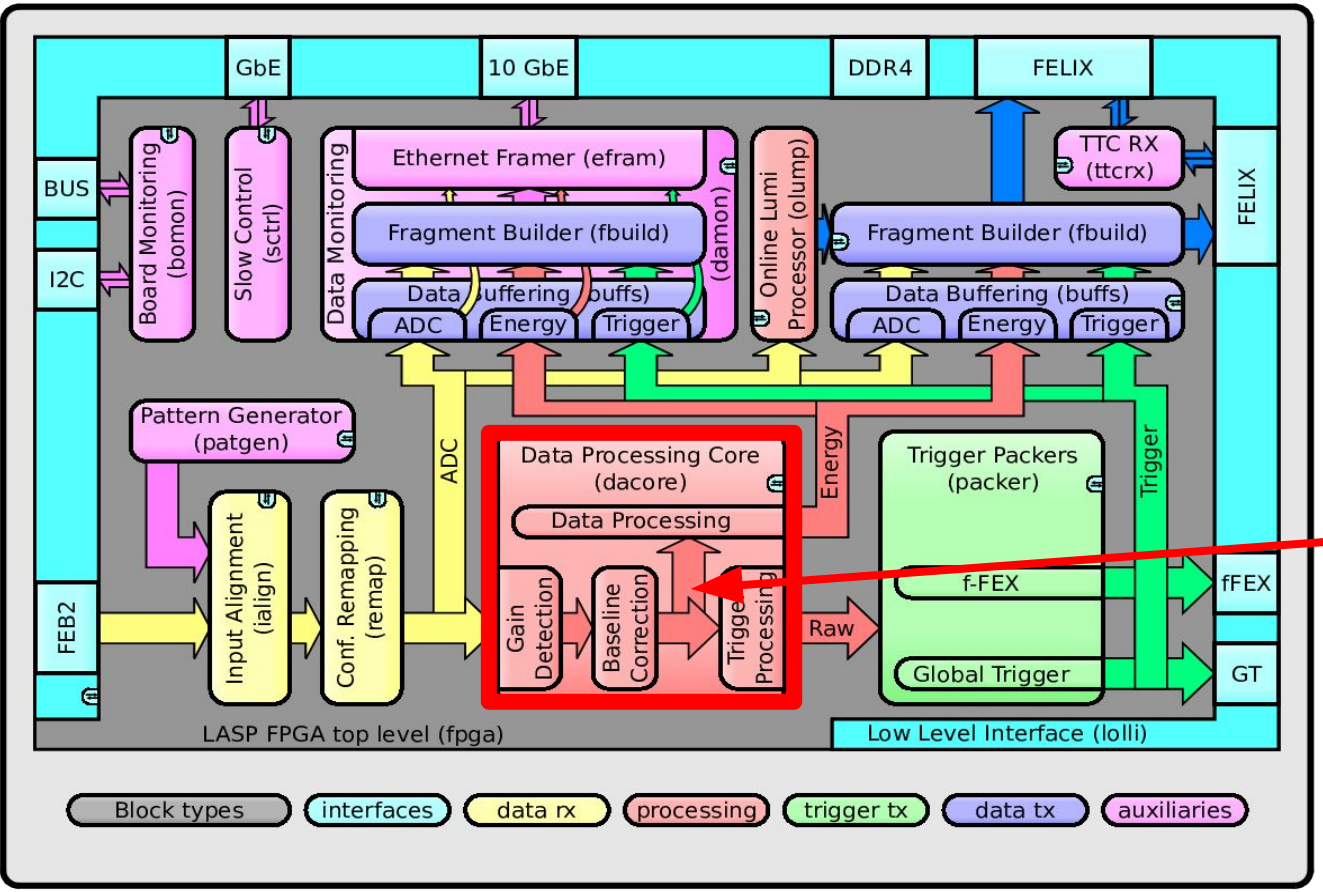
LASP board
Off-detector board responsible of energy computation

Phase 2 electronics
Trigger path + readout path

Phase 1 electronics
Trigger path

LASP Firmware

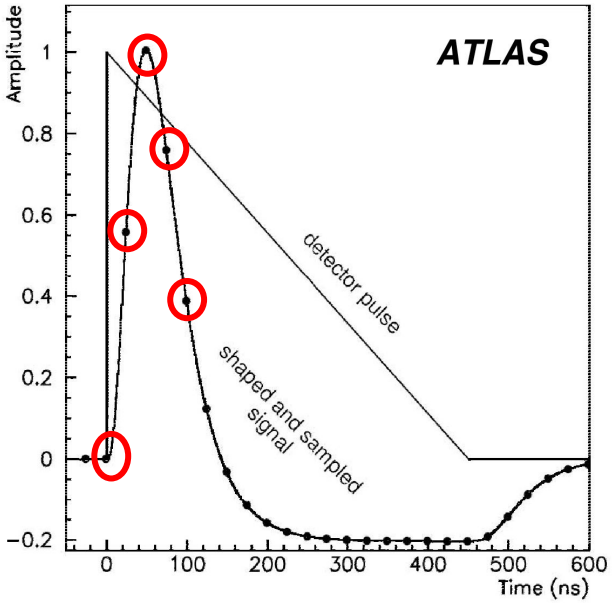
- LASP board containing 2 processing units based on INTEL FPGAs
  - Demonstrator board available with stratix 10 FPGAs
    - baseline for the firmware development shown in this talk
  - Final board will be equipped with Agilex FPGAs
- One FPGA should process 384 channels
  - About 125 ns allocated latency for energy computation

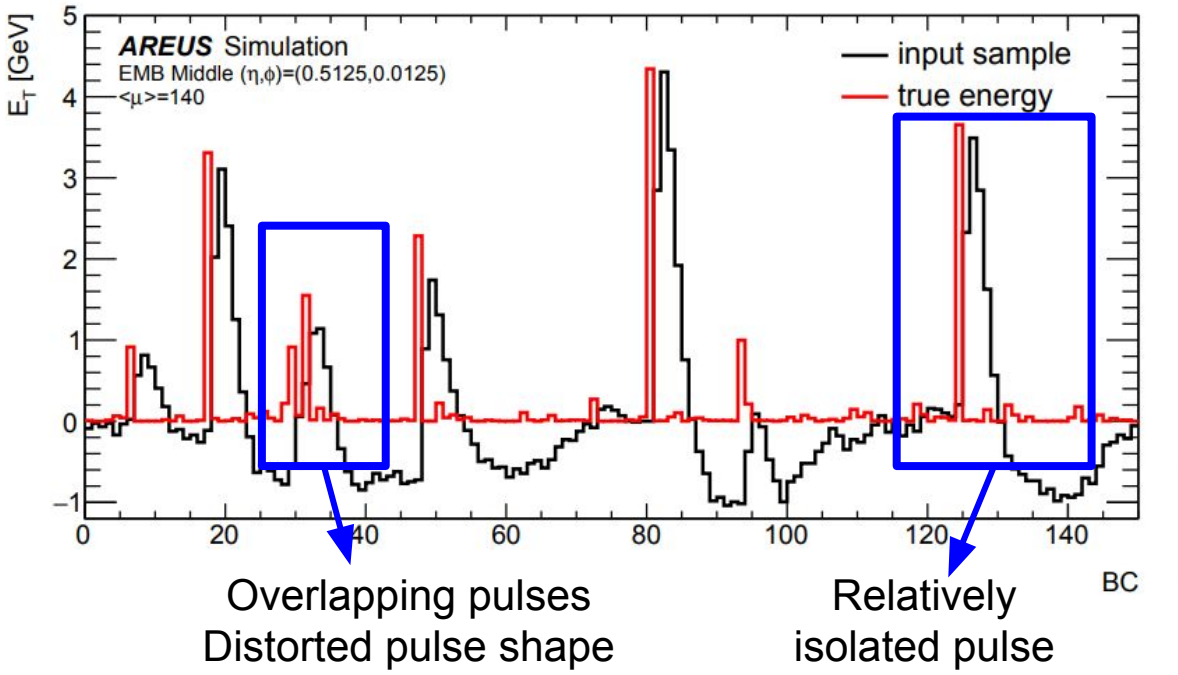


Compute energy at 40 MHz
Assign the energy to the correct bunch crossing (collision time)

Energy reconstruction

- Legacy energy reconstruction using an optimal filtering algorithm with max finder (OFMAX)
  - Optimal filtering to reconstruct the pulse and determine its amplitude (∝ energy)
  - Max finder to determine the correct time (bunch crossing)
- Not robust in case of distorted shapes due to pileup





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)

Energy reconstruction with NNs

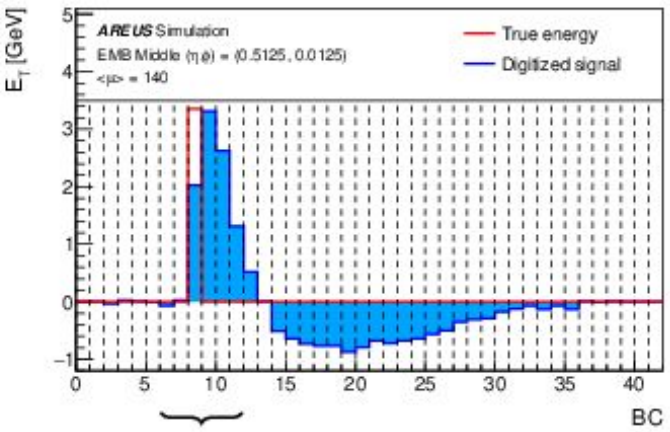
Two neural networks types tested:
Convolutional Neural Networks (CNNs) (Dresden)
and
Recursive Neural Networks (RNNs) (CPPM)

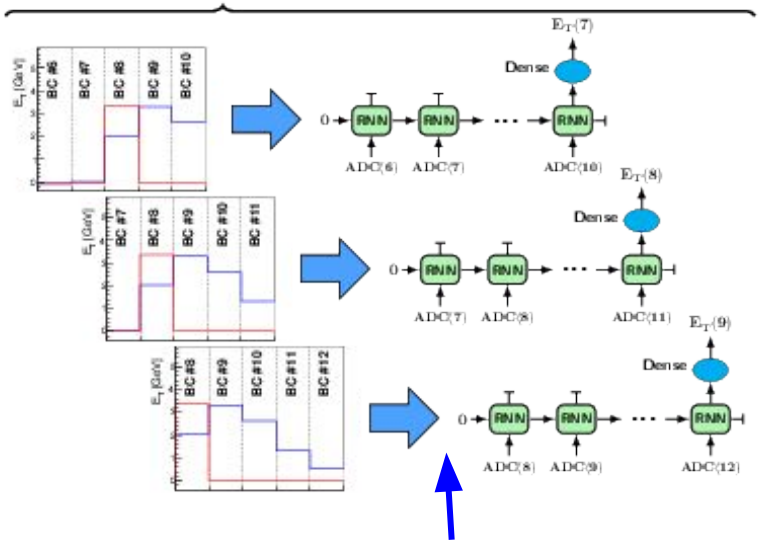
This talk will cover only RNNs

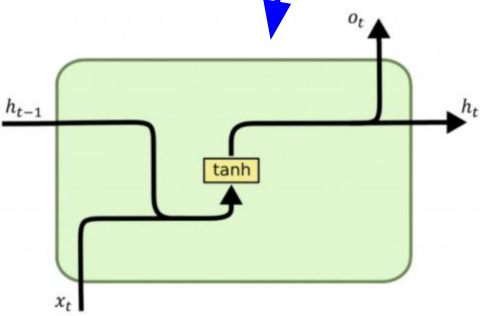
RNN structure

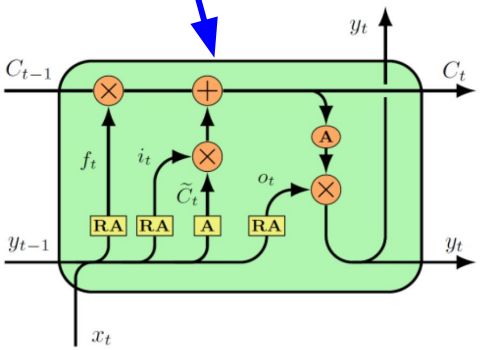
Comput Softw Big Sci **5**, 19 (2021)

- Sequence of RNN cells each taking as input an ADC sample at a given BCID
  - 4 samples on the pulse
  - N samples in the past to correct for pileup
- Two general parameters control the size of the network
  - Sequence length (number of samples)
  - NN units (internal dimension of the NNs in the cell)
- Several cell structures tested
  - Vanilla RNN, GRU, LSTM









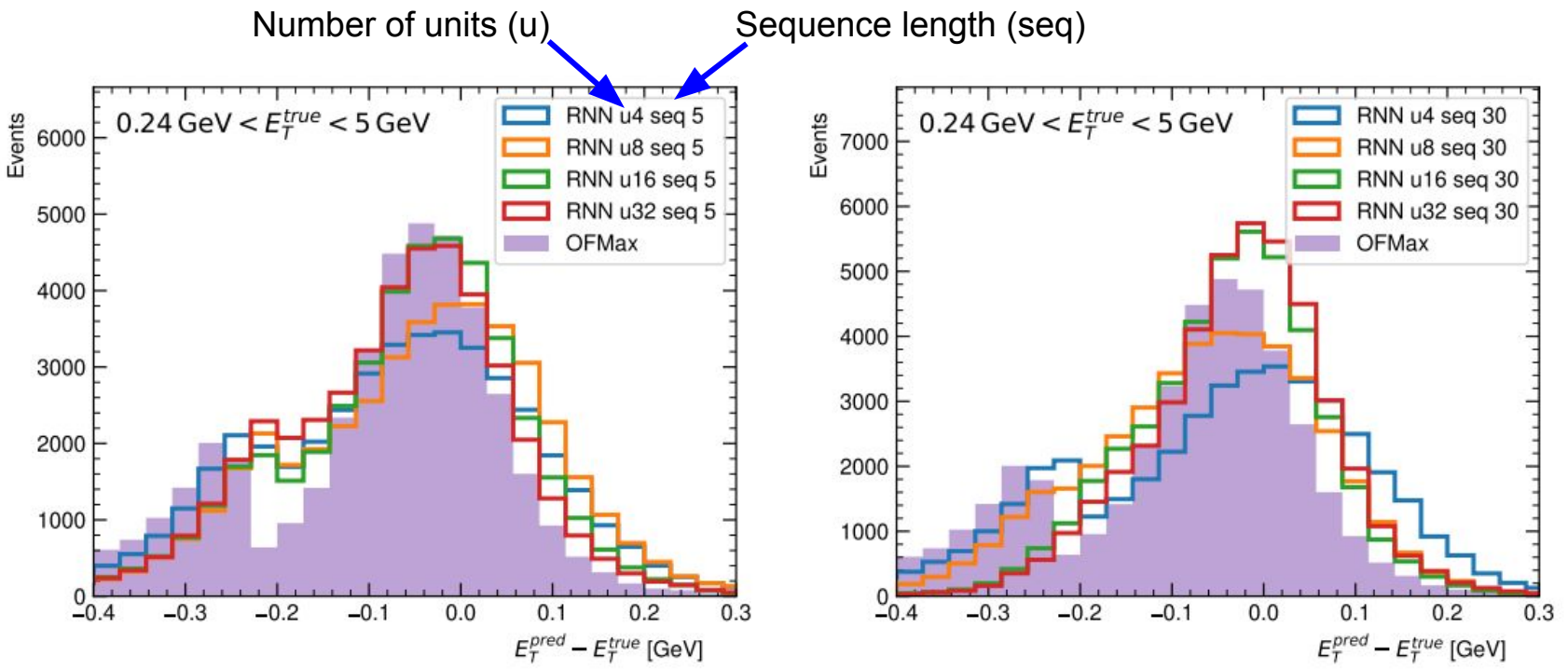
Sliding windows architecture

Computation on a moving slice of the data in fixed intervals

Takes into account a limited set in the past (1 sample in the past in this example)

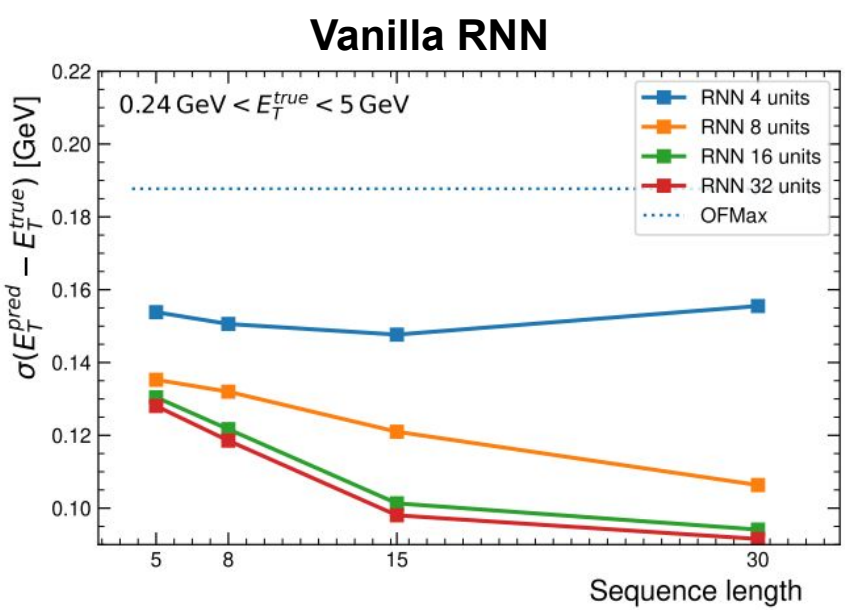
RNN Performance

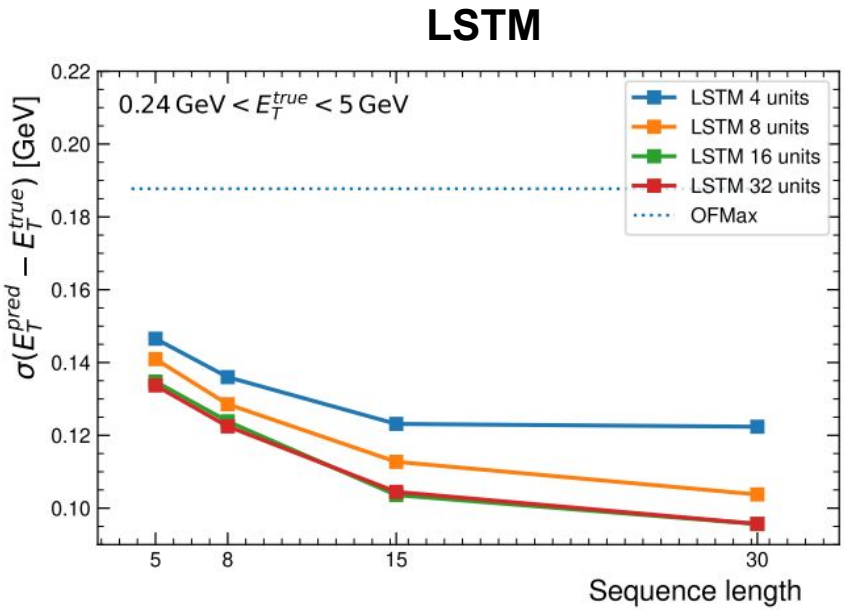
- Compare energy resolution between RNNs and OFMax
  - RNNs with increased size
  - Keep size under control to fit FPGAs
- Second peak in resolution due to overlapping events
- Use Std. Dev. as metric (although the shape is not very gaussian)



RNN Performance

- Compare energy resolution between RNNs and OFMax
  - RNNs with increased size
  - Keep size under control to fit FPGAs
- Second peak in resolution due to overlapping events
- Use Std. Dev. as metric (although the shape is not very gaussian)

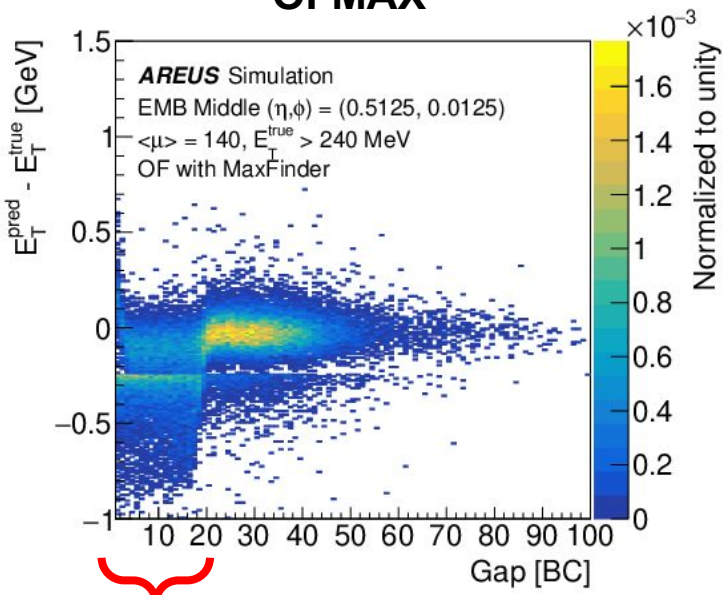




Performance as Function of Time Gap

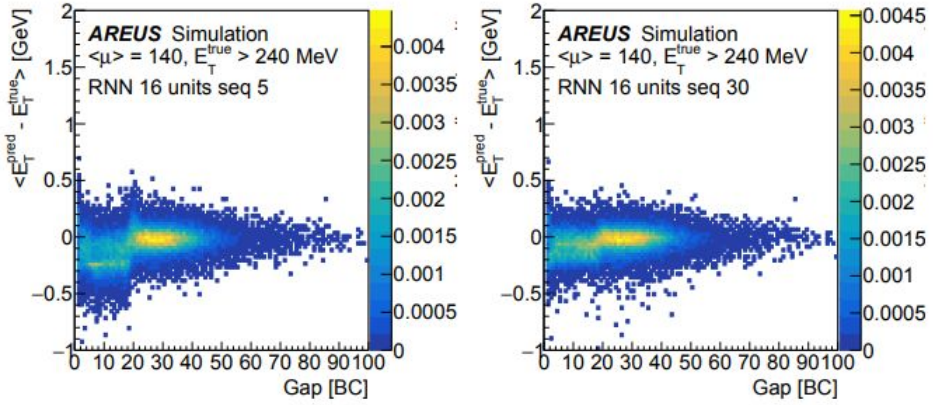
- Clear drop in OFMax performance in overlapping region
  - Time gap of less than ~ 20 BC
- Neural networks recover the performance in this region
  - Depending on the sequence length that is used

OFMAX

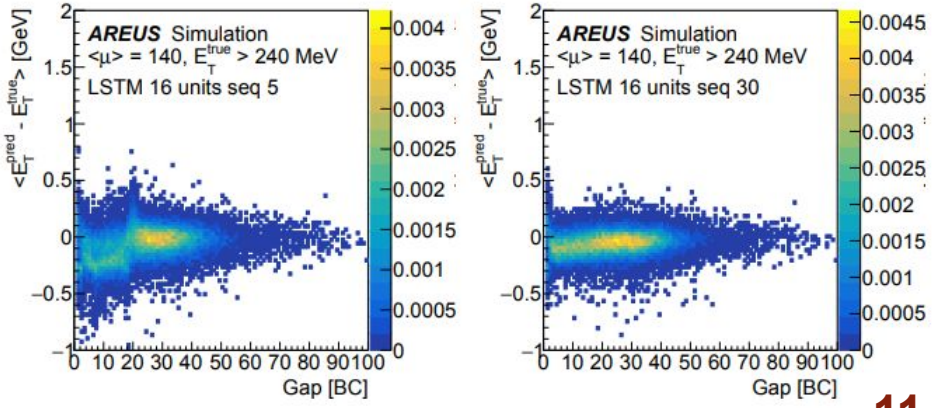


Overlapping signals region

Vanilla RNN

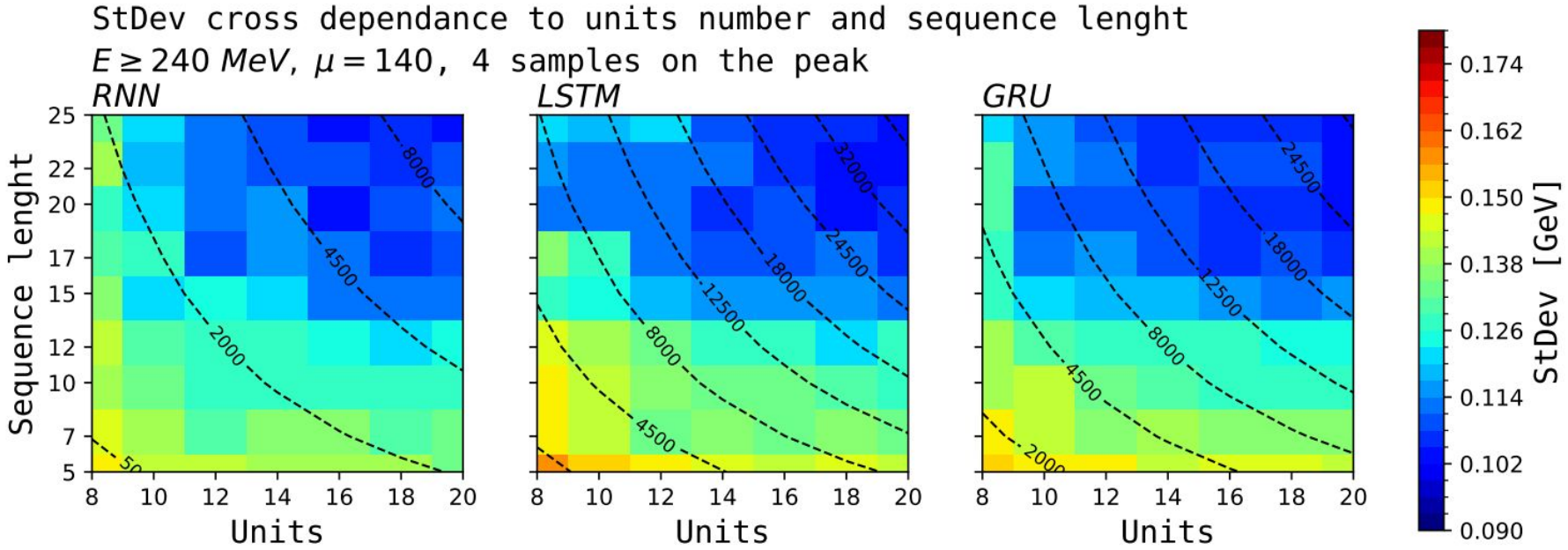


LSTM



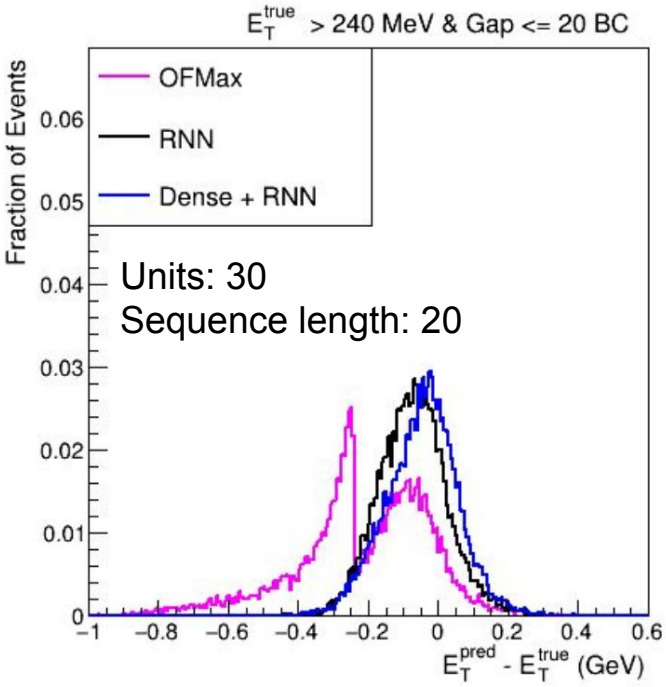
RNN Performance vs RNN Cell Type

- Checking performance of Vanilla-RNN, GRU and LSTM
  - Increased NN size by increasing sequence length and number of units
- Network size probed by number of multiplications (MAC units)
  - Dashed lines in the plots
- Vanilla-RNN can reach the same performance of GRU and LSTM with much less required MACs
  - Best adapted to fit in FPGAs

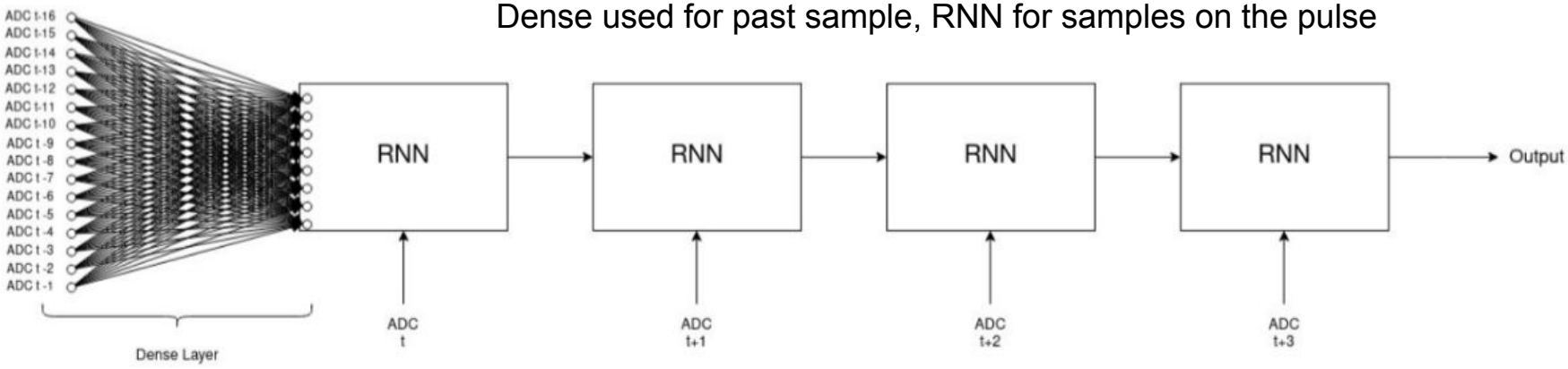


Optimized NN Architecture

- Long sequences (20-30) needed to efficiently correct for pileup
- Number of cells scales with the sequence length
  - RNN cells needs significant processing resources
  - # MACs ∝ s×n^2
    - s=sequence length, n=units
- Use dense layer to acquire pileup correction
  - # MACs ∝ s×n
  - Init RNN with dense output

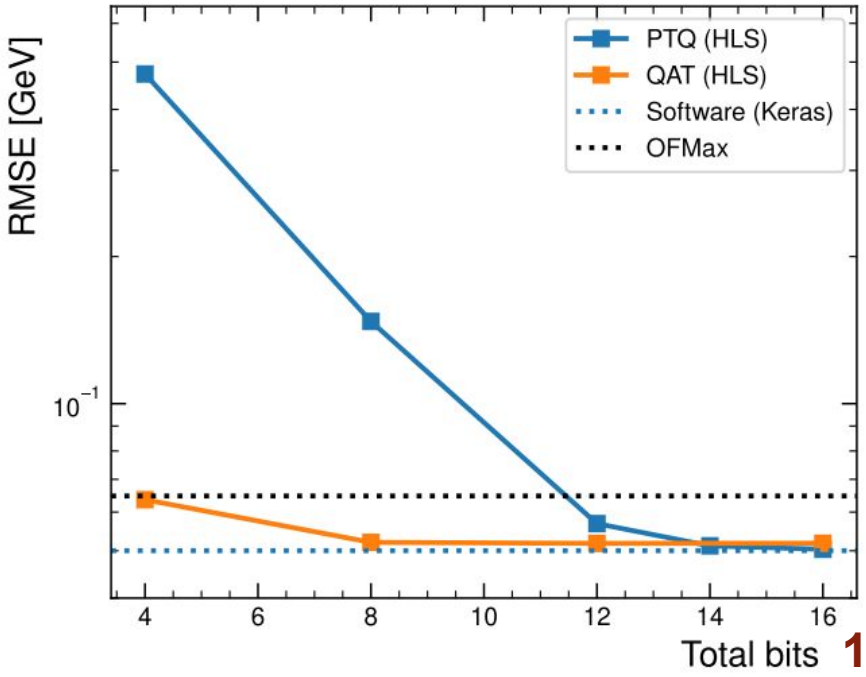


Dense used for past sample, RNN for samples on the pulse



RNN Quantization

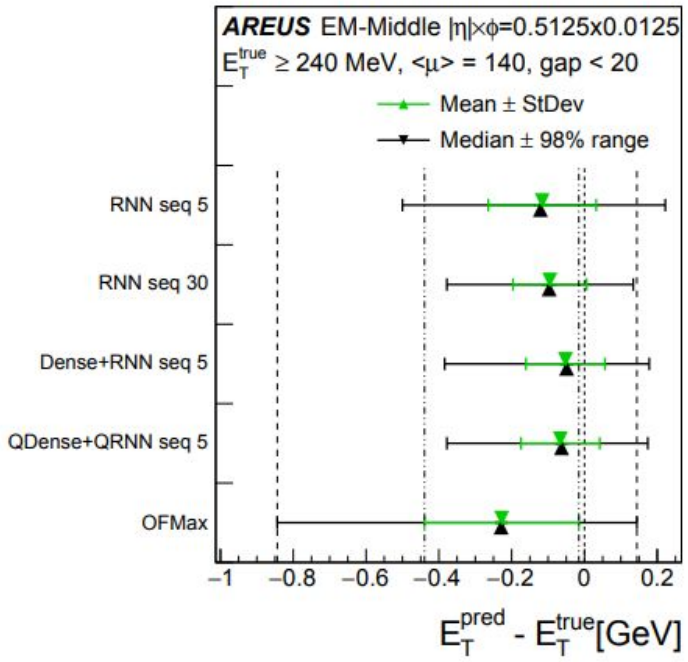
- RNNs need quantization to fit on FPGAs
  - Full floating point arithmetics takes a lot of FPGA resources
  - Need to use fixed points representations with small number of bits
- Quantize weights post training (PTQ)
  - Reduced resolution due to truncation/rounding
  - Can reach float precision with 16 bits
- Quantized Aware Training (using qKeras)
  - Optimize weights that are already quantized
  - Can reach float precision with 8 bits
- Stratix 10 considerations
  - One floating point multiplication per DSP
  - Two fixed point multiplication with 18x19 bits
- Agilex considerations
  - Additional DSP mode with four 9x9 bits multiplications
- Reduced number of bits allow to use more multiplications
  - Also matters for additions and timing closer



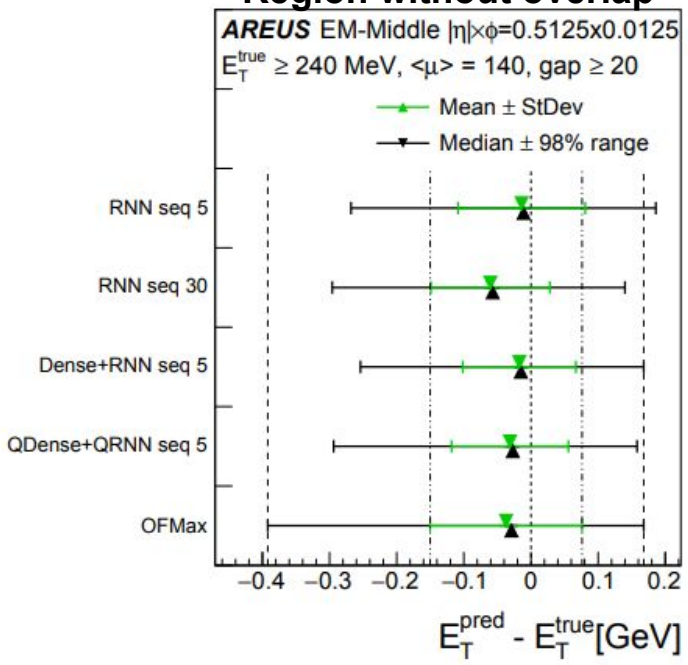
RNN Performance (Summary)

- Small RNNs (sequence length 5) can outperform OFMax overall
  - But not in all regions (details not discussed here)
- Need to go to larger sequence length (20-30)
  - Much more resources in FPGA
- Can reduce NN resources by using a dense layer to init the RNN
  - Solution adapted for the LAr usecase: no drop in performance
- Important to consider QAT to reduce the number of bits
  - Reach same performance with lower number of bits
  - Can have a large effect on FPGA implementation

Region with overlap



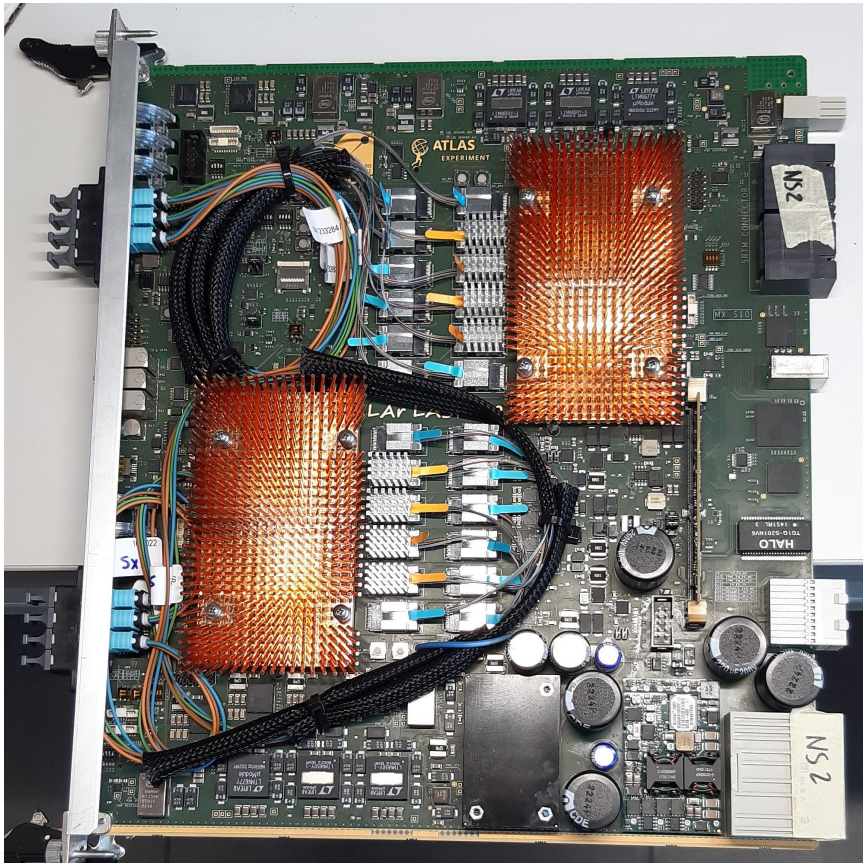
Region without overlap



Firmware Implementation

- Implemented on Stratix 10 FPGA
  - Reference 1SG280HU1F50E2VG
  - Implementation on Agilex will follow
- Challenges:
  - 384 channels per FPGA
  - 125 ns latency
- Preliminary implementation in HLS shows that LSTM is too large to fit
  - Stick to Vanilla RNN
  - Start with small RNN with 8 units and sequence length of 5
    - 491 parameters, 480 MAC units

LASP demonstrator board built at CPPM



- Added support for both Vanilla RNNs and LSTMs on INTEL FPGAs to HLS4ML for wider usage

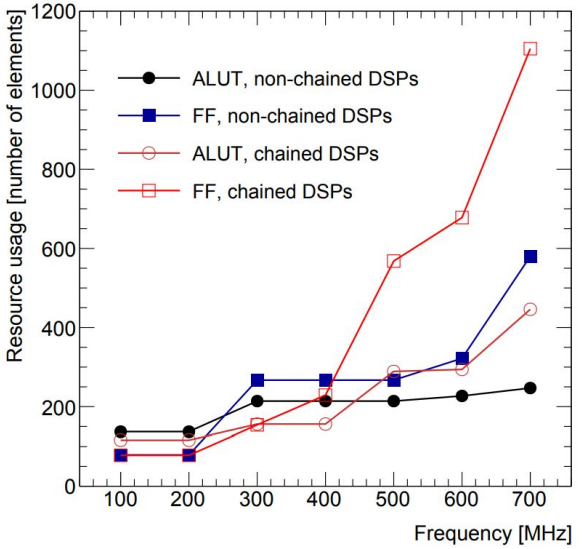
HLS optimisations

JINST **18** (2023) P05017

Etienne Fortin, PhD thesis (2022)

- Optimisation needed to fit RNNs within resource and latency limitations
  - Impossible to fit 384 NNs in the FPGAs, need to serialize (time multiplexing)
  - Need to go to high frequency
- Several optimisations are performed
  - Activation functions in LUT (only for LSTM)
  - Number of bits in fixed point representation (18x19 to match Stratix 10 DSP)
  - Rounding and truncation in arithmetic operations
  - Implementation of vector/matrix multiplication (Dot product)
- Dot product implementations
  - Naive C++: let HLS do it all
  - ACC37: accumulate (sum) in DSPs by chaining them
  - ACC19: ACC in ALUT

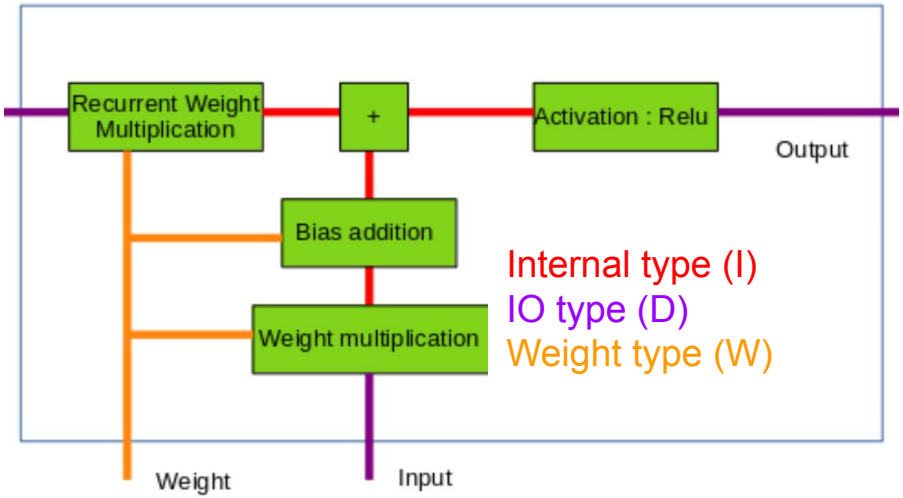
$$A.B = \begin{bmatrix} a_0 \\ a_1 \\ \vdots \\ a_8 \end{bmatrix} . \begin{bmatrix} b_0 \\ b_1 \\ \vdots \\ b_8 \end{bmatrix} = \sum_{i=0}^{7} a_i.b_i$$

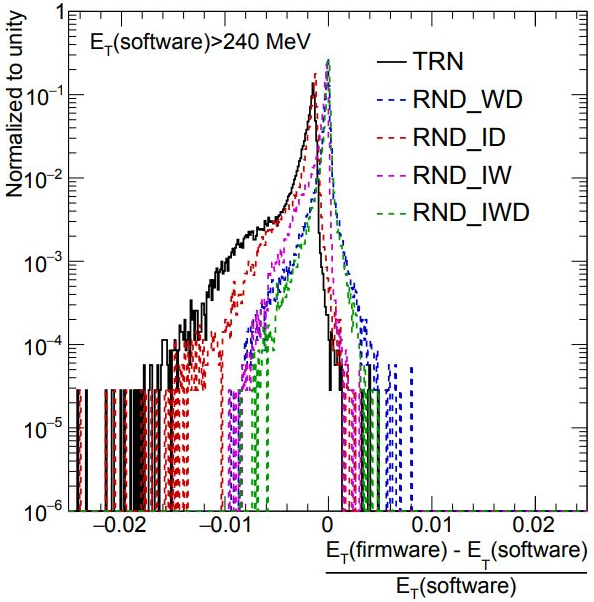


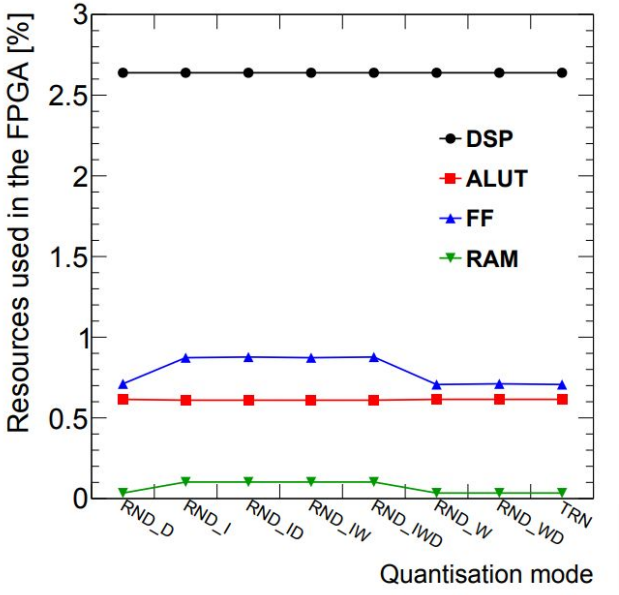
	Implementation	ALUTs	FF	DSP
@100 MHz	C++ style	709	222	8
	ACC37	116	79	4
	ACC19	137	78	4

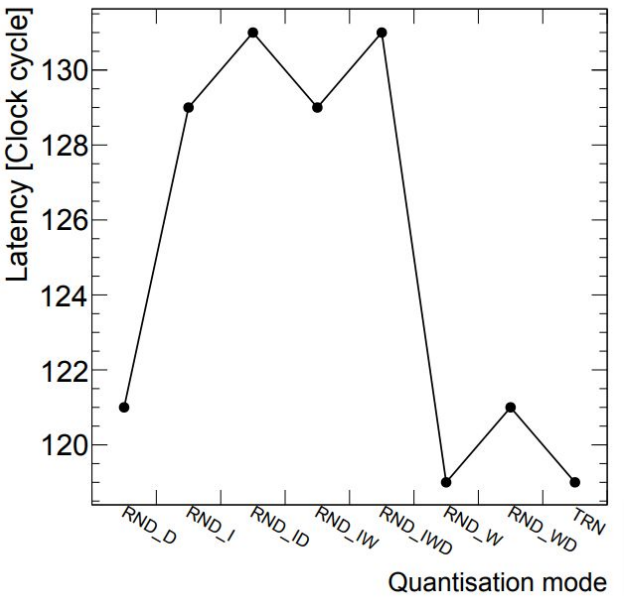
Rounding vs Truncation

- Compromise between resolution and resource usage and latency
  - Truncation of IO and Internal types leads to important reduction of latency with small impact on energy resolution
  - Weight type rounded in software
    - No impact on latency
- Use truncation for internal (I) operations





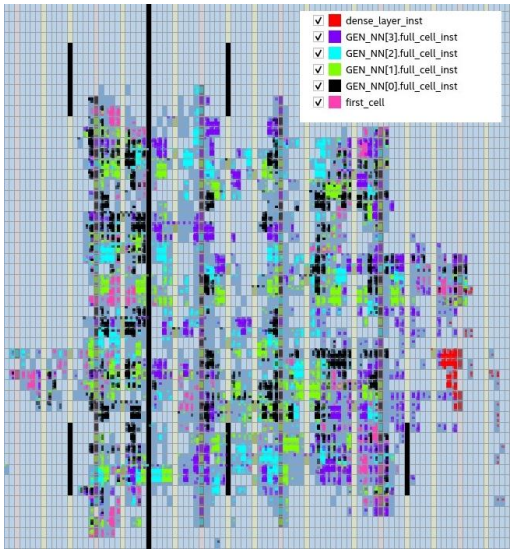




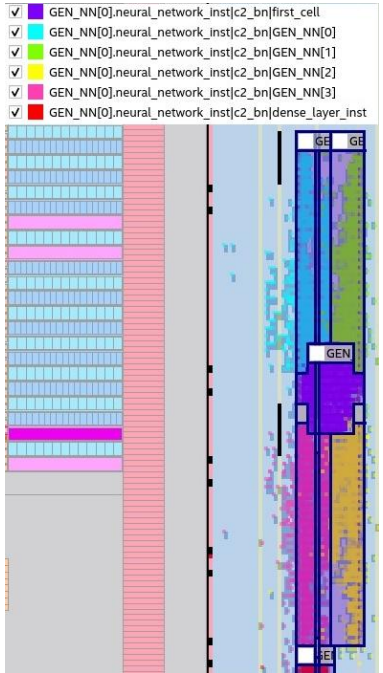
VHDL implementation of Vanilla RNN

- HLS does not allow to reach the target frequency and resource usage
  - Increase of the RNN ALM resources and reduction of FMax as we add networks to the FPGA
- Move to VHDL for the final fine tuning
- Force placement of the RNN components
  - Allow to better tackle timing violations and improve FMax
- Use incremental compilation
  - Keep networks with no timing violations and recompile only the rest

HLS placement



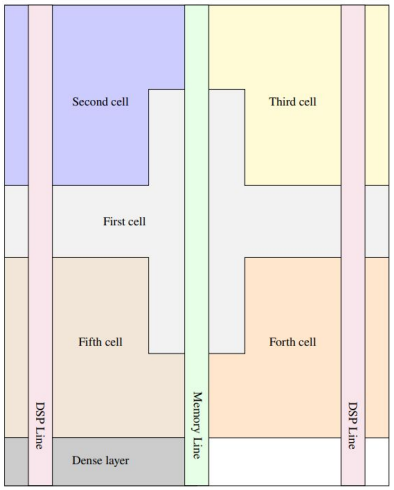
VHDL forced placement



Optimized placement of RNN cells

First cells in the middle and connected to all cells (common computations done only in first cell and propagated to the others

Dense layer next to last cell



RNN firmware results

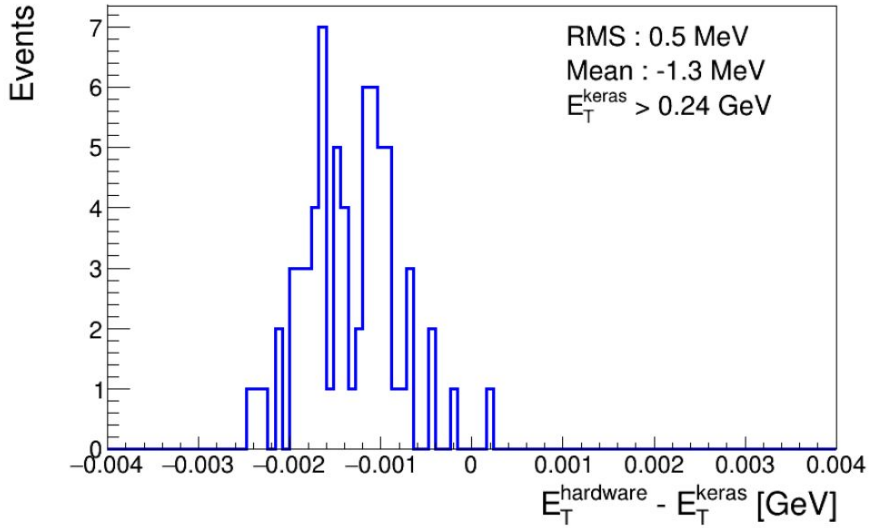
- HLS allows fast development and optimisation of the firmware
  - Multiple developments and firmware optimisations done in a short time
  - However less control on hardware specific implementation
- VHDL is needed to fine tune the design and fit the LAr requirements
- Vanilla RNN firmware produced and fit the requirements with Stratix 10
  - Better performance expected with the Agilex FPGA
  - However still need to test it within the full LASP firmware

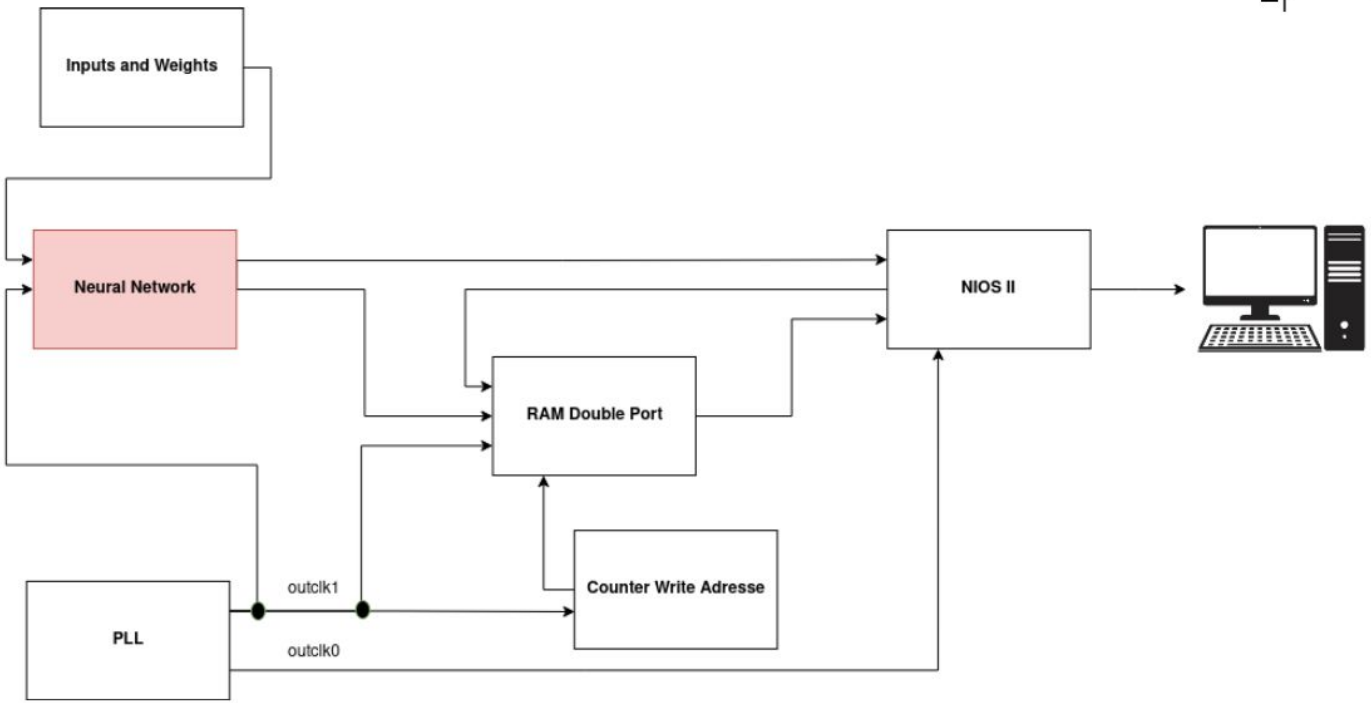
	N networks x multiplexing	ALM	DSP	FMax	latency
target	**384 channels**	**30%***	**70%***	**-**	**125 ns**
"Naive" HLS (no multiplexing)	384x1	226%	529%	-	322 ns
HLS optimized	37x10	90%	100%	393 MHz	277 ns
VHDL optimized	28x14	18%	66%	561 MHz	116 ns

*based on experience with the phase I upgrade

Testing on hardware

- VHDL implementation tested on Startix 10 DevKit
- Test firmware to inject input and weights and collect the output is built
  - Data extraction using a JTAG-UART connection with a NIOS
- Data match firmware simulation bit-by-bit
- Firmware resolution < 0.1% as expected from simulation



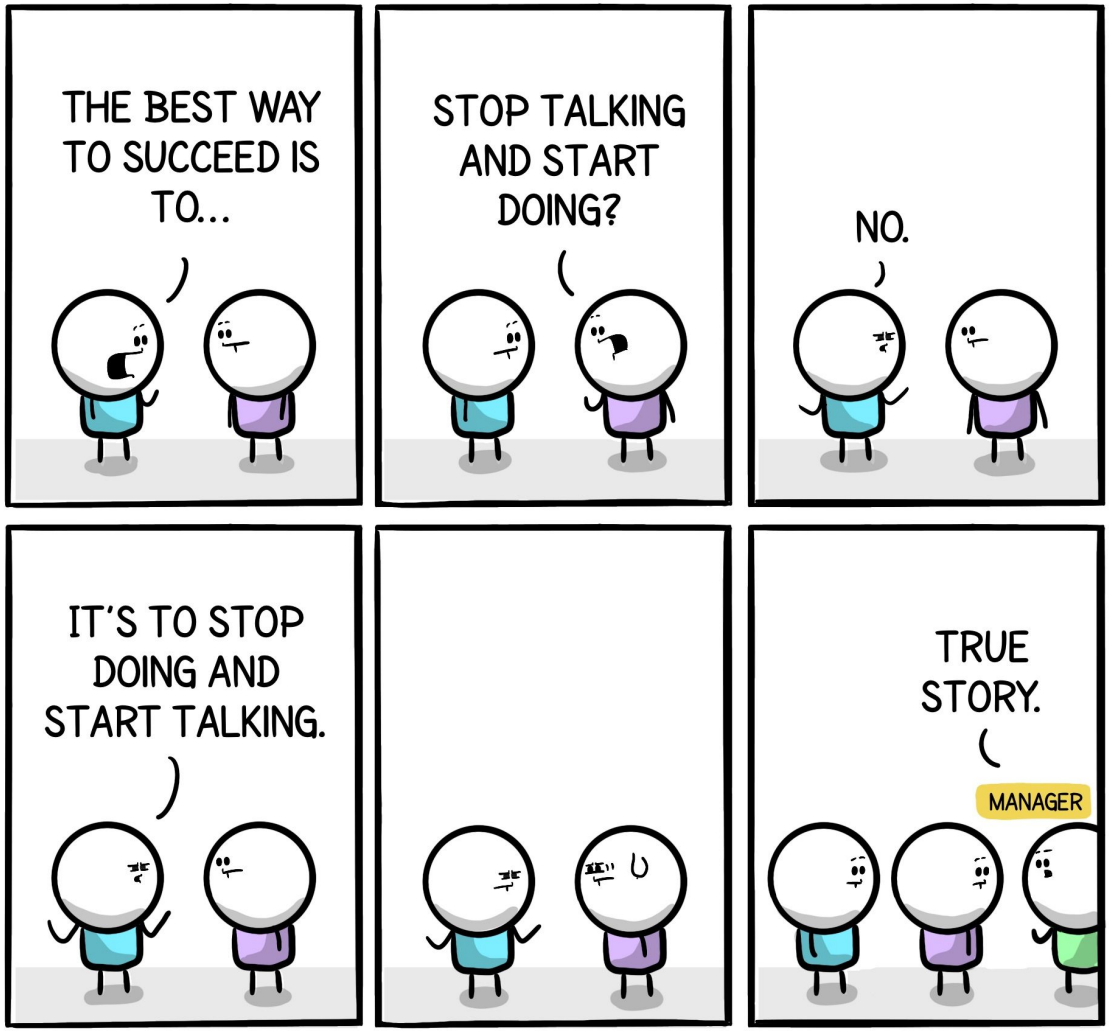


Conclusion

- Neural networks outperform the optimal filtering algorithm for the energy reconstruction in the ATLAS LAr Calorimeter
  - Particularly in the region with overlap between multiple pulses
- Several optimisations carried out to improve the RNN performance while keeping minimal resource usage
  - Next step is to quantify the effect on object (electrons, photons) reconstruction and physics performance

- Small Vanilla RNN implemented on Stratix 10 FPGAs
- HLS implementation allowed fast prototyping but did not fit resource and latency requirements
- Final implementation done in VHDL
  - Fits requirements and successfully tested on hardware
- Next steps is to implement larger networks in Agilex FPGAs
  - Then integrate the NNs in the full LASP firmware

Backup



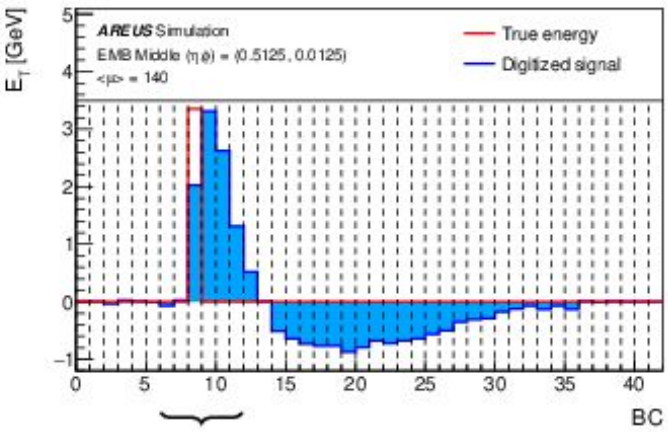
Hello. I make comics about work.
Follow me on Instagram / Twitter / Facebook.

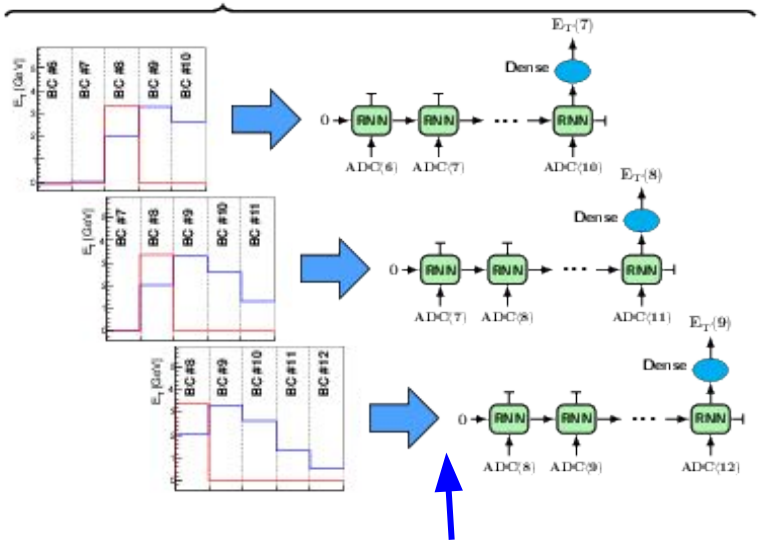
Work Chronicles
workchronicles.com

RNN structure

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

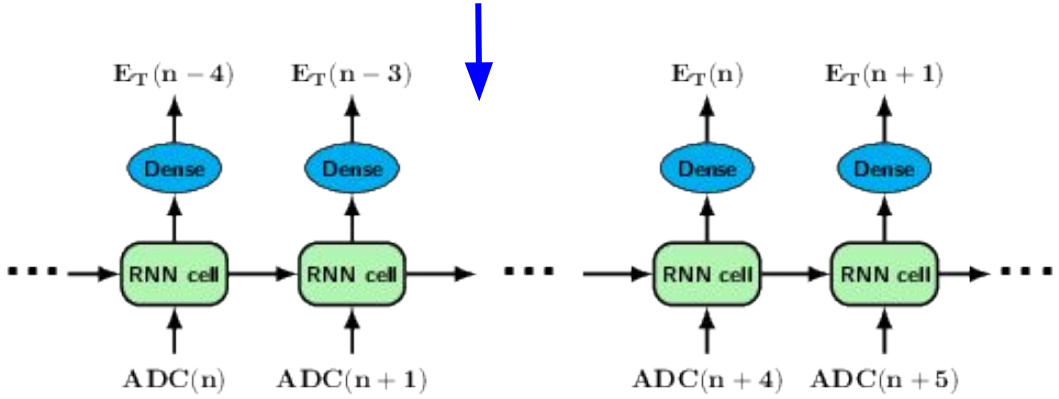
- Two architectures used
  - Single cell and Sliding windows
  - 4 samples corresponding to the signal pulse are used
    - + several in the past to correct for pileup
- Two types of RNNs
  - Vanilla RNN and LSTM
- Sliding window retained





Single cell architecture

Continuous computation with a single cell
Takes into account full past info (from the beginning of run)



Sliding windows architecture

Computation on a moving slice of the data in fixed intervals
Takes into account a limited set in the past (1 sample in the past in this talk)

RNN configuration

Table 2 Configurable key parameters of the single-cell and sliding-window algorithms.

		Single-cell LSTM	Sliding-window LSTM	Sliding-window Vanilla RNN
Time inference	Receptive Field	∞	5	5
	Samples after deposit	5	4	4
RNN layer	Dimension	10	10	8
	Activation	tanh	tanh	ReLU
	Recurrent Activation	sigmoid	sigmoid	N/A
Dense layer	Dimension	1	1	1
	Activation	ReLU	ReLU	ReLU
Number of Parameters		491	491	89
MAC units		480	2360	368

Dot product implementations

Naive C++ implementation

```
for (int i=0; i < 8; i++){
  acc += a[i] * b[i];
}
```

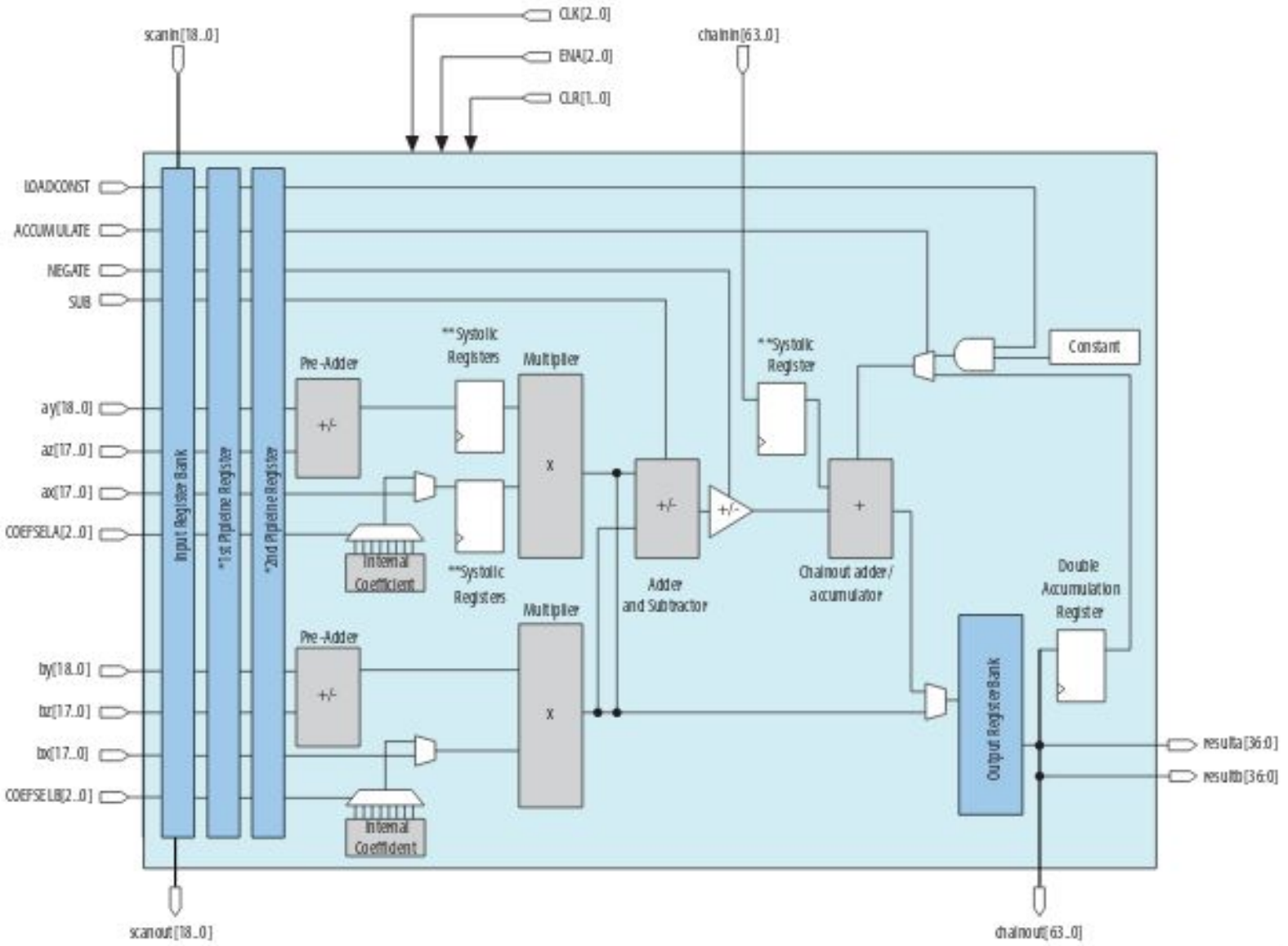
ACC37 implementation

```
for (int i=0; i < 4; i++){
  tmp[i] = a[i]*b[i] + a[7-i]*b[7-i];
}
for (int i=0; i < 4; i++){
  acc += tmp[i];
}
```

ACC19 implementation

```
for (int i=0; i < 4; i++){
  tmp[i] = hls_fpga_reg(a[i]*b[i] + a[7-i]*b[7-i]);
}
for (int i=0; i < 4; i++){
  acc += tmp[i];
}
```

DSP


scanin[18..0]
CLK[2..0]
ENA[2..0]
CLR[1..0]
chainin[63..0]
LOADCONST
ACCUMULATE
NEGATE
SUB
ay[18..0]
az[17..0]
ax[17..0]
COEFSELA[2..0]
by[18..0]
bz[17..0]
bx[17..0]
COEFSELB[2..0]
Input Register Bank
*1st Pipeline Register
*2nd Pipeline Register
Pre-Adder
+/-
**Systolic Registers
Multiplier
x
Internal Coefficient
Adder and Subtractor
**Systolic Register
Constant
Chainout adder / accumulator
Double Accumulation Register
Output RegisterBank
resulta[36:0]
resultb[36:0]
scanout[18..0]
chainout[63..0]

Pileup noise