Joint ICTP-IAEA School on Detector Signal Processing and Machine Learning for Scientific Instrumentation and Reconfigurable Computing

Using FPGAs to Accelerate Machine Learning Algorithms

smr4110 | Trieste, Italy - 2025

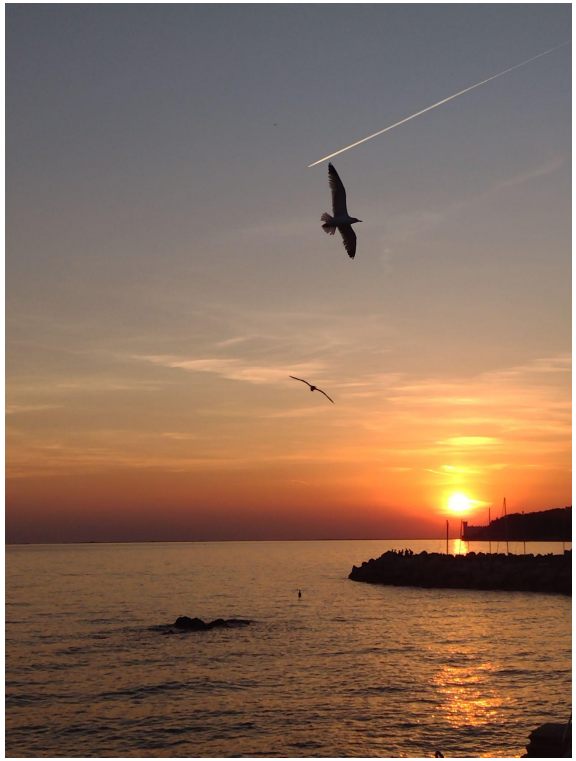
Romina Soledad Molina, Ph.D.

IAEA International Atomic Energy Agency
Atoms for Peace and Development

The Abdus Salam International Centre for Theoretical Physics

Outline

- Summing up the previous content.
- High-Level Synthesis for Machine Learning - hls4ml.
- hls4ml workflow
- Workflow for Deep Neural Network Deployment On Embedded Architectures.



Summing up the previous content

Summing up the previous content

- **Rise of real-time ML**
  - ML is now used in latency-critical domains: HEP, robotics, autonomous systems, IoT.

Summing up the previous content

- **Rise of real-time ML**
  - ML is now used in latency-critical domains: HEP, robotics, autonomous systems, IoT.
- **The need for fast and efficient inference**
  - Low-latency, low-power inference.

Summing up the previous content

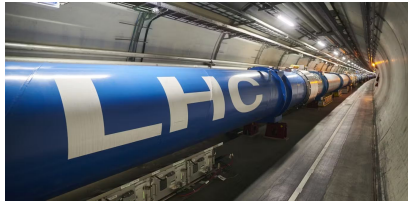
- **Rise of real-time ML**
  - ML is now used in latency-critical domains: HEP, robotics, autonomous systems, IoT.
- **The need for fast and efficient inference**
  - Low-latency, low-power inference
- **Why FPGAs?**
  - Highly parallel and reconfigurable, tuned for latency and energy efficiency.

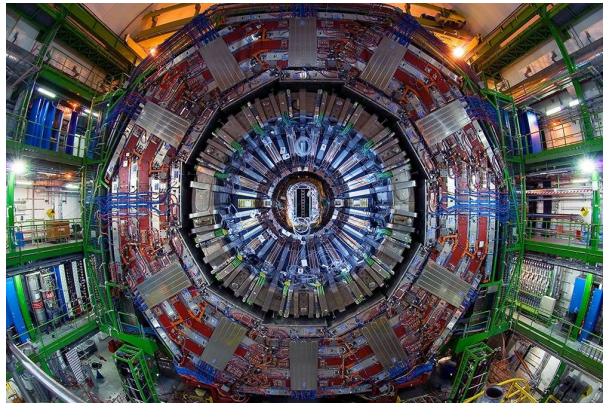
Summing up the previous content

- **Rise of real-time ML**
  - ML is now used in latency-critical domains: HEP, robotics, autonomous systems, IoT.
- **The need for fast and efficient inference**
  - Low-latency, low-power inference.
- **Why FPGAs?**
  - Highly parallel and reconfigurable, tuned for latency and energy efficiency.
- **The challenge**
  - ML models are software-native.
  - Hardware mapping is complex and manual.
  - Needs automation and abstraction.

hls4ml

"**The inspiration for the creation of the hls4ml** package stems from the **high energy physics community at the CERN Large Hadron Collider (LHC)**. While machine learning has already been proven to be extremely useful in analysis of data from detectors at the LHC, it is typically performed in an "offline" environment after the data is taken and agglomerated." [hls4ml]

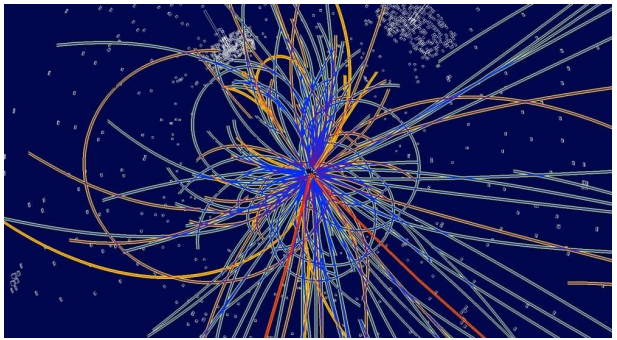




[hls4ml] https://fastmachinelearning.org/hls4ml/

hls4ml

“However, **one of the largest problems at detectors on the LHC is that collisions, or “events”, generate too much data for everything to be saved**. As such, filters called **“triggers” are used to determine whether a given event should be kept. Using FPGAs allows for significantly lower latency so machine learning algorithms can essentially be run “live” at the detector level for event selection**. As a result, more events with potential signs of new physics can be preserved for analysis.” [hls4ml]



[hls4ml] https://fastmachinelearning.org/hls4ml/

Fast inference of deep neural networks in FPGAs for particle physics

J. Duarte,[a] S. Han,[b] P. Harris,[b] S. Jindariani,[a] E. Kreinar,[c] B. Kreis,[a] J. Ngadiuba,[d] M. Pierini,[d] R. Rivera,[a] N. Tran[a,1] and Z. Wu[e]

[a] *Fermi National Accelerator Laboratory, Batavia, IL 60510, U.S.A.*
[b] *Massachusetts Institute of Technology, Cambridge, MA 02139, U.S.A.*
[c] *HawkEye360, Herndon, VA 20170, U.S.A.*
[d] *CERN, CH-1211 Geneva 23, Switzerland*
[e] *University of Illinois at Chicago, Chicago, IL 60607, U.S.A.*

E-mail: hls4ml.help@gmail.com

Abstract: Recent results at the Large Hadron Collider (LHC) have pointed to enhanced physics capabilities through the improvement of the real-time event processing techniques. Machine learning methods are ubiquitous and have proven to be very powerful in LHC physics, and particle physics as a whole. However, exploration of the use of such techniques in low-latency, low-power FPGA (Field Programmable Gate Array) hardware has only just begun. FPGA-based trigger and data acquisition systems have extremely low, sub-microsecond latency requirements that are unique to particle physics. We present a case study for neural network inference in FPGAs focusing on a classifier for jet substructure which would enable, among many other physics scenarios, searches for new dark sector particles and novel measurements of the Higgs boson. While we focus on a specific example, the lessons are far-reaching. A companion compiler package for this work is developed based on High-Level Synthesis (HLS) called `hls4ml` to build machine learning models in FPGAs. The use of HLS increases accessibility across a broad user community and allows for a drastic

Duarte, J., Han, S., Harris, P., Jindariani, S., Kreinar, E., Kreis, B., ... & Wu, Z. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *Journal of instrumentation*, *13*(07), P07027.

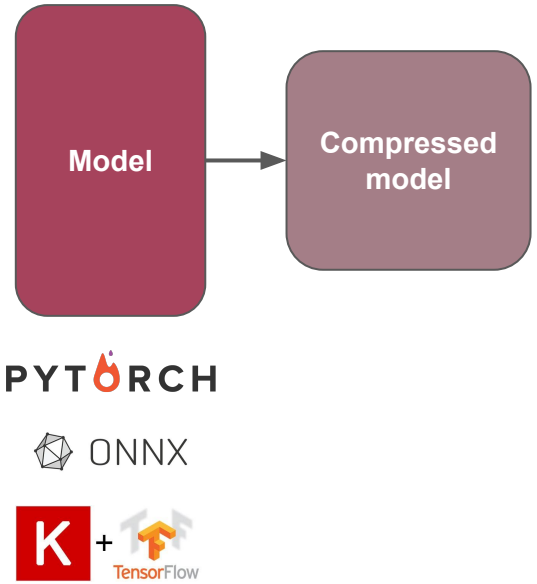
- Binary & Ternary neural networks: [2020 Mach. Learn.: Sci. Technol]
  - Compressed weights for low resource inference
- Boosted Decision Trees: [JINST 15 P05026 (2020)]
  - Low latency for Decision Tree ensembles
- GarNet / GravNet: [arXiv: 2008.03601]
  - Distance weighted graph neural networks suitable for sparse/irregular point-cloud data
- Quantization aware training QKeras + support in hls4ml: [arXiv: 2006.10159]
- Convolutional neural networks: Mach. Learn.: Sci. Technol. 2 045015 (2021)

- Python package.
- It enables the transformation of neural network models into firmware deployable on FPGAs for efficient inference.
  - https://fastmachinelearning.org/hls4ml/
  - https://github.com/hls-fpga-machine-learning/hls4ml
  - pip install hls4ml
- Extremely configurable: precision, resource vs latency/throughput tradeoff.
- Easy to use.

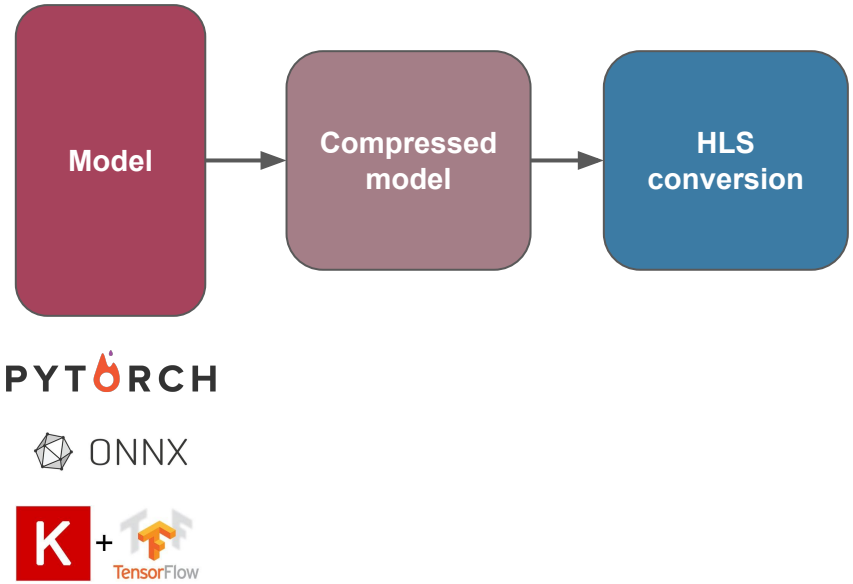
Duarte, J., Han, S., Harris, P., Jindariani, S., Kreinar, E., Kreis, B., ... & Wu, Z. (2018). Fast inference of deep neural networks in FPGAs for particle physics. *Journal of instrumentation*, *13*(07), P07027.

hls4ml workflow

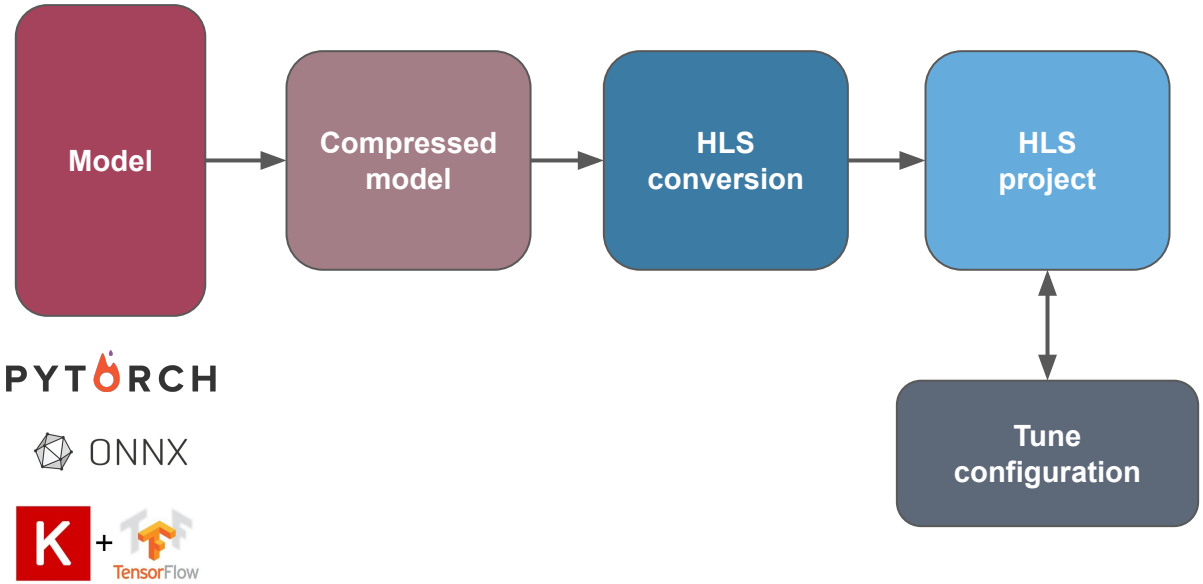
hls4ml workflow



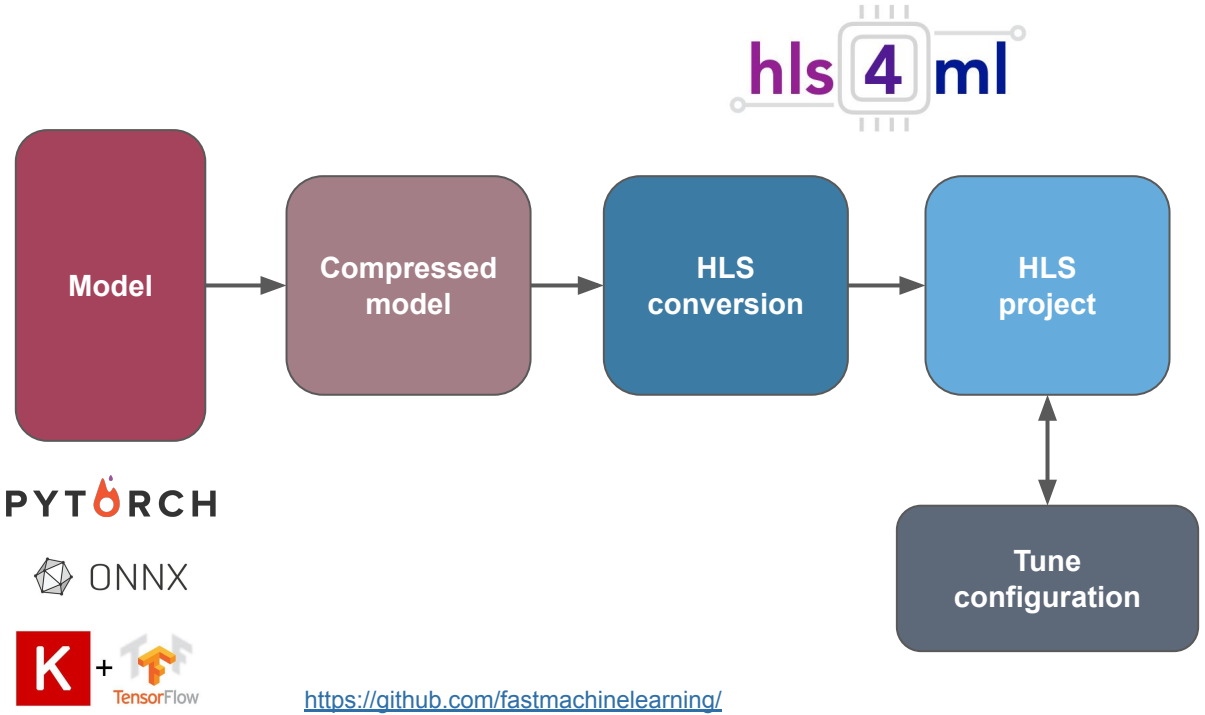
hls4ml workflow



hls4ml workflow

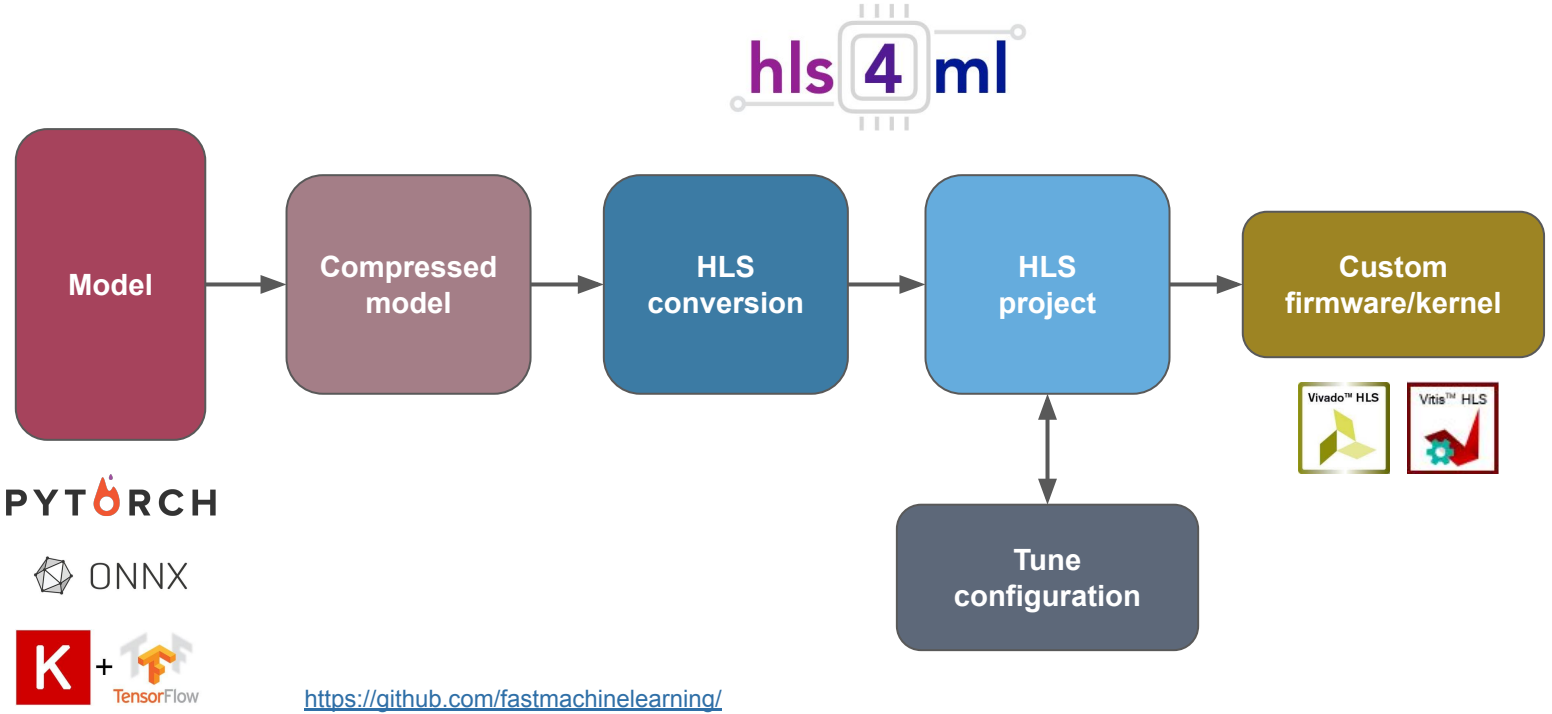


hls4ml workflow



https://github.com/fastmachinelearning/

hls4ml workflow



https://github.com/fastmachinelearning/

High-Level Synthesis for Machine Learning

ML framework support:

- (Q)Keras
- PyTorch
- (Q)ONNX

https://fastmachinelearning.org/hls4ml/

High-Level Synthesis for Machine Learning

ML framework support:

- (Q)Keras
- PyTorch
- (Q)ONNX

Neural networks architectures:

- Fully Connected NN
- Convolutional NN
- Recurrent NN
- Graph NN

https://fastmachinelearning.org/hls4ml/

High-Level Synthesis for Machine Learning

ML framework support:

- (Q)Keras
- PyTorch
- (Q)ONNX

Neural networks architectures:

- Fully Connected NN
- Convolutional NN
- Recurrent NN
- Graph NN

HLS backends:

- Vivado HLS
- Intel HLS
- Vitis HLS
- Catapult HLS
- oneAPI (experimental)

https://fastmachinelearning.org/hls4ml/

High-Level Synthesis for Machine Learning

hls4ml is tested on the following platforms:

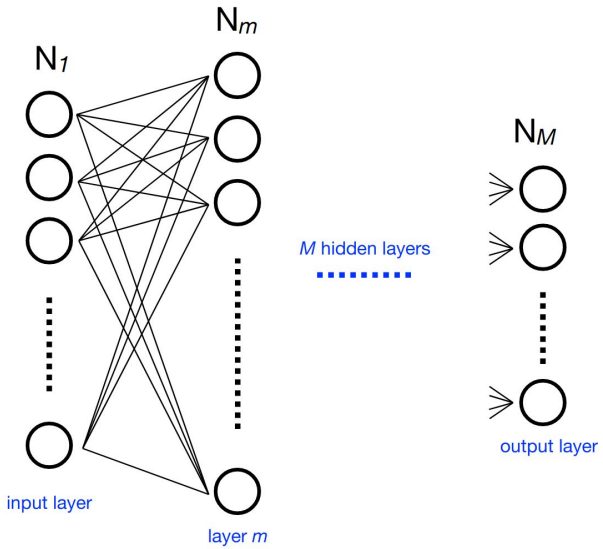
Vivado HLS versions 2018.2 to 2020.1
Intel HLS versions 20.1 to 21.4. Versions > 21.4 have not been tested.
Vitis HLS versions 2022.2 to 2024.1. Versions <= 2022.1 are known not to work.
Catapult HLS versions 2024.1_1 to 2024.2
oneAPI versions 2024.1 to 2025.0

https://fastmachinelearning.org/hls4ml/

hls4ml

How does it work?

With hls4ml, each layer of output values is calculated independently in sequence, using pipelining to speed up the process by accepting new inputs after an initiation interval. The activations, if nontrivial, are precomputed.



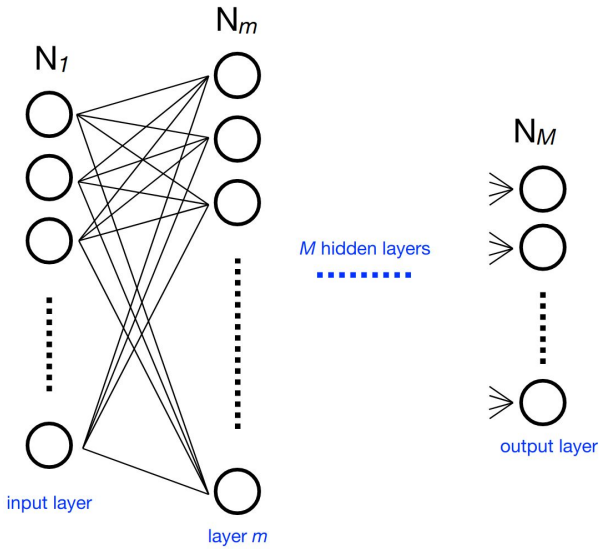
[hls4ml] https://fastmachinelearning.org/hls4ml/

hls4ml

How does it work?

With hls4ml, each layer of output values is calculated independently in sequence, using pipelining to speed up the process by accepting new inputs after an initiation interval. The activations, if nontrivial, are precomputed.

Simplifying the input network must be done before using hls4ml to generate HLS code, for optimal compression to provide a sizable speedup.



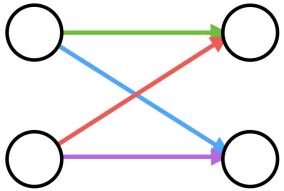
[hls4ml] https://fastmachinelearning.org/hls4ml/

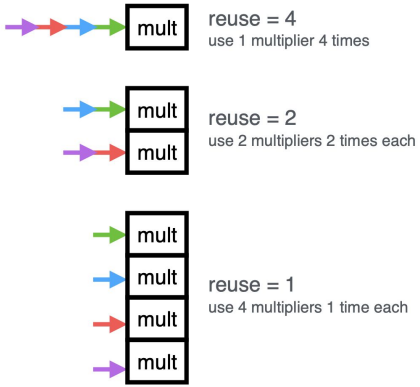
Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer.

hls4ml

Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer.

Reuse factor: number of times a multiplier is used to do a computation.



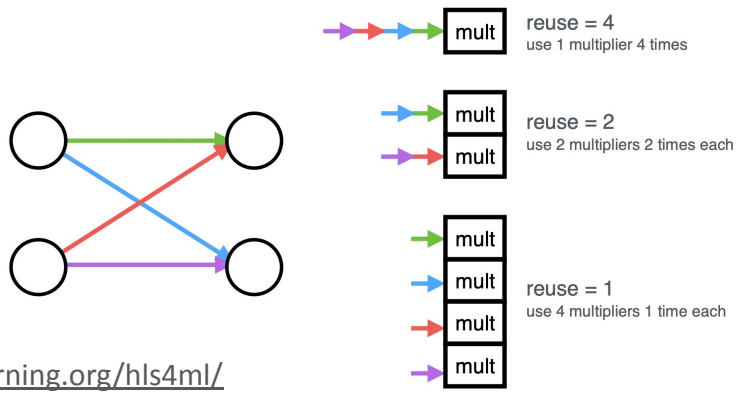


https://fastmachinelearning.org/hls4ml/

hls4ml

Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer.

Reuse factor: number of times a multiplier is used to do a computation.



Fewer resources,
Lower throughput,
Higher latency

More resources,
Higher throughput,
Lower latency

https://fastmachinelearning.org/hls4ml/

I/O types: supports multiple styles for handling data transfer to/from the network and between layers.

https://fastmachinelearning.org/hls4ml/

I/O types: supports multiple styles for handling data transfer to/from the network and between layers.

io_parallel

io_stream

https://fastmachinelearning.org/hls4ml/

I/O types: supports multiple styles for handling data transfer to/from the network and between layers.

io_parallel

Data is passed in **parallel** between the layers.

This style allows for **maximum parallelism** and is well suited for MLP networks and small CNNs which aim for lowest latency.

I/O types: supports multiple styles for handling data transfer to/from the network and between layers.

io_stream

Data is passed **one "pixel" at a time**.

Each pixel is an **array of channels**, which are always sent in parallel. This method for sending data between layers is recommended for larger CNN and RNN networks.

With this IO type, each layer is connected with the subsequent layer through **first-in first-out (FIFO) buffers.**

The implementation of the FIFO buffers contribute to the overall resource utilization of the design, impacting in particular the **BRAM or LUT utilization**.

https://fastmachinelearning.org/hls4ml/

Strategy: the implementation of core matrix-vector multiplication routine, which can be **latency-oriented, resource-saving oriented, or specialized**.

Different strategies will have an **impact on overall latency and resource consumption** of each layer and users are advised to choose based on their design goals.

https://fastmachinelearning.org/hls4ml/

Strategy: the implementation of core matrix-vector multiplication routine, which can be **latency-oriented, resource-saving oriented, or specialized**.

Different strategies will have an **impact on overall latency and resource consumption** of each layer and users are advised to choose based on their design goals.

If one layer would have >**4096 elements**, we should set **['Strategy'] = 'Resource'** for that layer, or **increase the reuse factor** by hand.

4096 elements is related to the maximum size of an array to be partitioned. This value might change, it should be checked with the hardware synthesis tool.

https://fastmachinelearning.org/hls4ml/

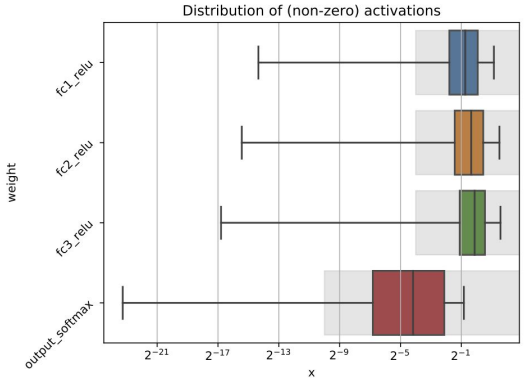
Activations - Implementation parameter

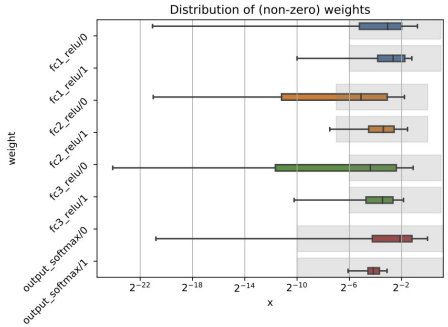
- **latency:** Good latency, high resource usage. **It does not work well if there are many output classes**.
- **stable:** Slower but with better accuracy, useful in scenarios where **higher accuracy is needed**.
- **legacy:** An older implementation with poor accuracy, but good performance. Usually the latency implementation is preferred.
- **argmax:** If you don't care about normalized outputs and only care about which one has the highest value, using argmax saves a lot of resources. This sets the highest value to 1, the others to 0.

https://fastmachinelearning.org/hls4ml/

Profiling

- The tools in **hls4ml.model.profiling** can help to choose the right precision for the model.
- **hls4ml.model.profiling.numerical** method with three objects: **a Keras model object**, **test data**, and an **HLSModel.**





https://fastmachinelearning.org/hls4ml/
Image from ttps://fastmachinelearning.org/hls4ml/api/PROFILING.html

Trace

- When we start using **customised precision** throughout the model, it can be useful to collect the output from each layer. We enable this trace collection by setting **Trace = True** for each layer whose output we want to collect.

```
for layer in config['LayerName'].keys():
        config['LayerName'][layer]['Trace'] = True
```

https://fastmachinelearning.org/hls4ml/
https://fastmachinelearning.org/hls4ml-tutorial/part2_advanced_config.html

High-Level Synthesis for Machine Learning

Python integration

```
from qkeras.utils import _add_supported_quantized_objects
co = {}
_add_supported_quantized_objects(co)
model = load_model('../models/mnistPQKD.h5', custom_objects=co)
```

✓ 0.1s

```
model.summary()
```

✓ 0.0s

```
Model: "sequential"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 fc1_input (QDense)          (None, 5)                 3925

 relu_input (QActivation)    (None, 5)                 0

 fc1 (QDense)                (None, 7)                 42

 relu1 (QActivation)         (None, 7)                 0

 fc2 (QDense)                (None, 10)                80

 relu2 (QActivation)         (None, 10)                0

 output (QDense)             (None, 2)                 22

 sigmoid (Activation)        (None, 2)                 0

=================================================================
Total params: 4,069
Trainable params: 4,069
Non-trainable params: 0
_________________________________________________________________
```

High-Level Synthesis for Machine Learning

Python integration

hls4ml

```
# granularity='model'

hls_config = hls4ml.utils.config_from_keras_model(model, granularity='model')


# User Configuration

hls_config['Model']['Precision'] = 'ap_fixed<8, 6>'
hls_config['Model']['ReuseFactor'] = 16
hls_config['Model']['Strategy'] = 'Latency' # or resource

import plotting

print("-----------------------------------")
plotting.print_dict(hls_config)
print("-----------------------------------")
```

High-Level Synthesis for Machine Learning

Python integration

hls4ml

```
Interpreting Sequential
Topology:
Layer name: fc1_input_input, layer type: InputLayer, input shapes: [[None, 784]], output shape: [None, 784]
Layer name: fc1_input, layer type: QDense, input shapes: [[None, 784]], output shape: [None, 5]
Layer name: relu_input, layer type: Activation, input shapes: [[None, 5]], output shape: [None, 5]
Layer name: fc1, layer type: QDense, input shapes: [[None, 5]], output shape: [None, 7]
Layer name: relu1, layer type: Activation, input shapes: [[None, 7]], output shape: [None, 7]
Layer name: fc2, layer type: QDense, input shapes: [[None, 7]], output shape: [None, 10]
Layer name: relu2, layer type: Activation, input shapes: [[None, 10]], output shape: [None, 10]
Layer name: output, layer type: QDense, input shapes: [[None, 10]], output shape: [None, 2]
Layer name: sigmoid, layer type: Activation, input shapes: [[None, 2]], output shape: [None, 2]
---------------------------------
Model
  Precision:         ap_fixed<8, 6>
  ReuseFactor:       16
  Strategy:          Latency
  BramFactor:        1000000000
  TraceOutput:       False
---------------------------------
```

High-Level Synthesis for Machine Learning

Python integration

hls4ml

```
# granularity='name'

hls_config = hls4ml.utils.config_from_keras_model(model, granularity='name')

for layer in hls_config['LayerName'].keys():
    # to collect the output from each layer
    # hls_config['LayerName'][layer]['Trace'] = True

    hls_config['LayerName'][layer]['ReuseFactor'] = 16

hls_config['LayerName']['fc1_input_input']['Precision'] = 'ap_fixed<16, 6>'
hls_config['LayerName']['fc1']['Precision'] = 'ap_fixed<8, 4>'

hls_config['LayerName']['sigmoid']['Strategy'] = 'Stable'

# To ensure DSPs are optimized, "unrolled" Dense multiplication must be used before synthesizing HLS
hls_config['Model']['Strategy'] = 'Unrolled'
```

High-Level Synthesis for Machine Learning

Python integration


```
Interpreting Sequential
Topology:
Layer name: fc1_input_input, layer type: InputLayer, input shapes: [[None, 784]], output shape: [None, 784]
Layer name: fc1_input, layer type: QDense, input shapes: [[None, 784]], output shape: [None, 5]
Layer name: relu_input, layer type: Activation, input shapes: [[None, 5]], output shape: [None, 5]
Layer name: fc1, layer type: QDense, input shapes: [[None, 5]], output shape: [None, 7]
Layer name: relu1, layer type: Activation, input shapes: [[None, 7]], output shape: [None, 7]
Layer name: fc2, layer type: QDense, input shapes: [[None, 7]], output shape: [None, 10]
Layer name: relu2, layer type: Activation, input shapes: [[None, 10]], output shape: [None, 10]
Layer name: output, layer type: QDense, input shapes: [[None, 10]], output shape: [None, 2]
Layer name: sigmoid, layer type: Activation, input shapes: [[None, 2]], output shape: [None, 2]
```

High-Level Synthesis for Machine Learning

Python integration

```
-----------------------------------
Model
  Precision:         fixed<16,6>
  ReuseFactor:       1
  Strategy:          Unrolled
  BramFactor:        1000000000
  TraceOutput:       False
LayerName
  fc1_input_input
    Trace:           False
    Precision:       ap_fixed<16, 6>
    ReuseFactor:     16
  fc1_input
    Trace:           False
    Precision
      result:        fixed<16,6>
      weight:        fixed<8,4>
      bias:          fixed<8,4>
    ReuseFactor:     16
  fc1_input_linear
    Trace:           False
    Precision
      result:        fixed<16,6>
    ReuseFactor:     16
  relu_input
    Trace:           False
    Precision
      result:        fixed<16,7,RND_CONV,SAT>
    ReuseFactor:     16
  fc1
```

High-Level Synthesis for Machine Learning

Python integration


```
cfg = hls4ml.converters.create_config(backend='vitis')

# cfg['IOType']     = 'io_stream'   # Must set this if using CNNs!
cfg['HLSConfig']  = hls_config      # HLS configuraiton
cfg['KerasModel'] = model           # Keras model to be converted
cfg['OutputDir']  = 'hw/'           # Project name
cfg['Part'] = 'xc7z020clg484-1'     # Zedboard: xc7z020clg484-1  ARTIX-7 xc7a35tcsg325-1  # MPSoC xczu4eg-sfvc784-2-e  xczu3eg-sfvc784-1-e
```

High-Level Synthesis for Machine Learning

Python integration


```
cfg = hls4ml.converters.create_config(backend='vitis')

# cfg['IOType']     = 'io_stream'   # Must set this if using CNNs!
cfg['HLSConfig']  = hls_config      # HLS configuraiton
cfg['KerasModel'] = model           # Keras model to be converted
cfg['OutputDir']  = 'hw/'           # Project name
cfg['Part'] = 'xc7z020clg484-1'     # Zedboard: xc7z020clg484-1  ARTIX-7 xc7a35tcsg325-1  # MPSoC xczu4eg-sfvc784-2-e  xczu3eg-sfvc784-1-e
```

High-Level Synthesis for Machine Learning

Python integration

hls4ml

```
hls_model = hls4ml.converters.keras_to_hls(cfg)
```

```
Interpreting Sequential
Topology:
Layer name: fc1_input_input, layer type: InputLayer, input shapes: [[None, 784]], output shape: [None, 784]
Layer name: fc1_input, layer type: QDense, input shapes: [[None, 784]], output shape: [None, 5]
Layer name: relu_input, layer type: Activation, input shapes: [[None, 5]], output shape: [None, 5]
Layer name: fc1, layer type: QDense, input shapes: [[None, 5]], output shape: [None, 7]
Layer name: relu1, layer type: Activation, input shapes: [[None, 7]], output shape: [None, 7]
Layer name: fc2, layer type: QDense, input shapes: [[None, 7]], output shape: [None, 10]
Layer name: relu2, layer type: Activation, input shapes: [[None, 10]], output shape: [None, 10]
Layer name: output, layer type: QDense, input shapes: [[None, 10]], output shape: [None, 2]
Layer name: sigmoid, layer type: Activation, input shapes: [[None, 2]], output shape: [None, 2]
Creating HLS model
```

High-Level Synthesis for Machine Learning

Python integration


```
hls_model.compile()
hls_model.build(csim=False, export=False)

# hls_model.build(csim=True, export=True, bitfile=True)
```

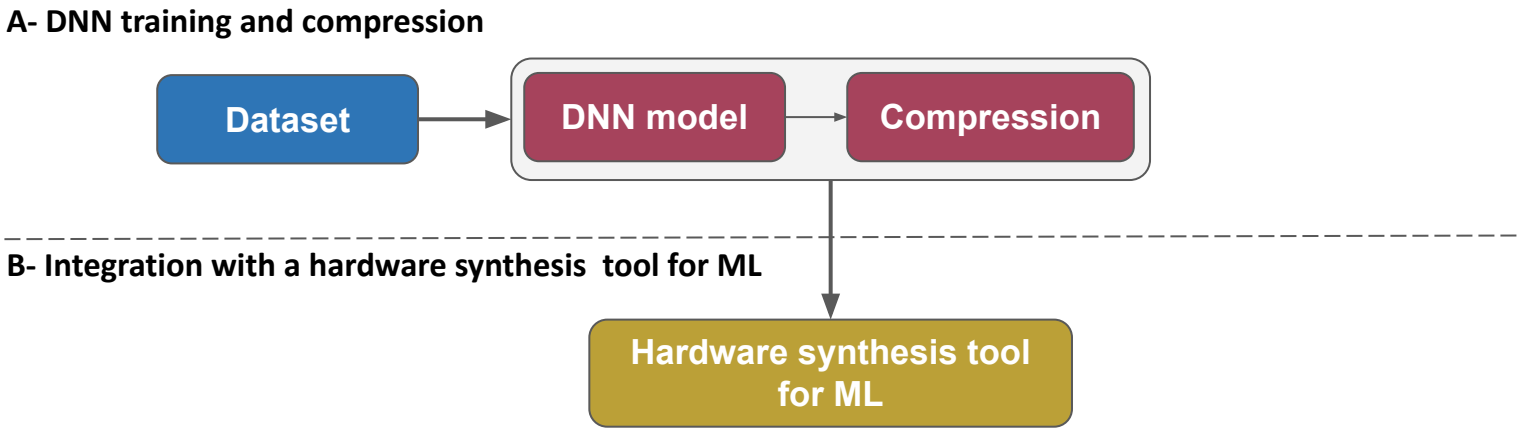
```
{'CSynthesisReport': {'TargetClockPeriod': '5.00',
  'EstimatedClockPeriod': '4.332',
  'BestLatency': '24',
  'WorstLatency': '24',
  'IntervalMin': '7',
  'IntervalMax': '7',
  'BRAM_18K': '1',
  'DSP': '4',
  'FF': '35096',
  'LUT': '51833',
  'URAM': '0',
  'AvailableBRAM_18K': '280',
  'AvailableDSP': '220',
  'AvailableFF': '106400',
  'AvailableLUT': '53200',
  'AvailableURAM': '0'}}
```

Workflow for Deep Neural Network Deployment On Embedded Architectures

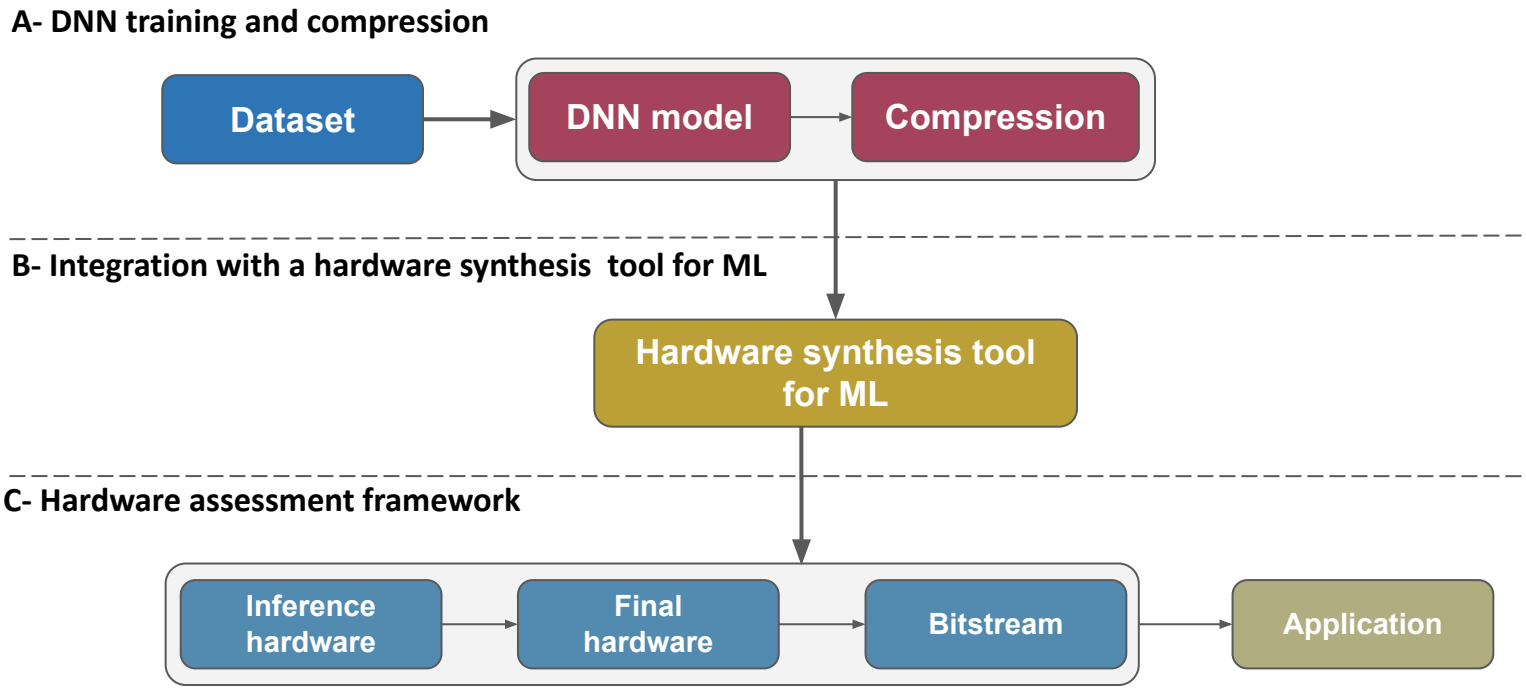
End-to-end workflow

A- DNN training and compression

End-to-end workflow



End-to-end workflow

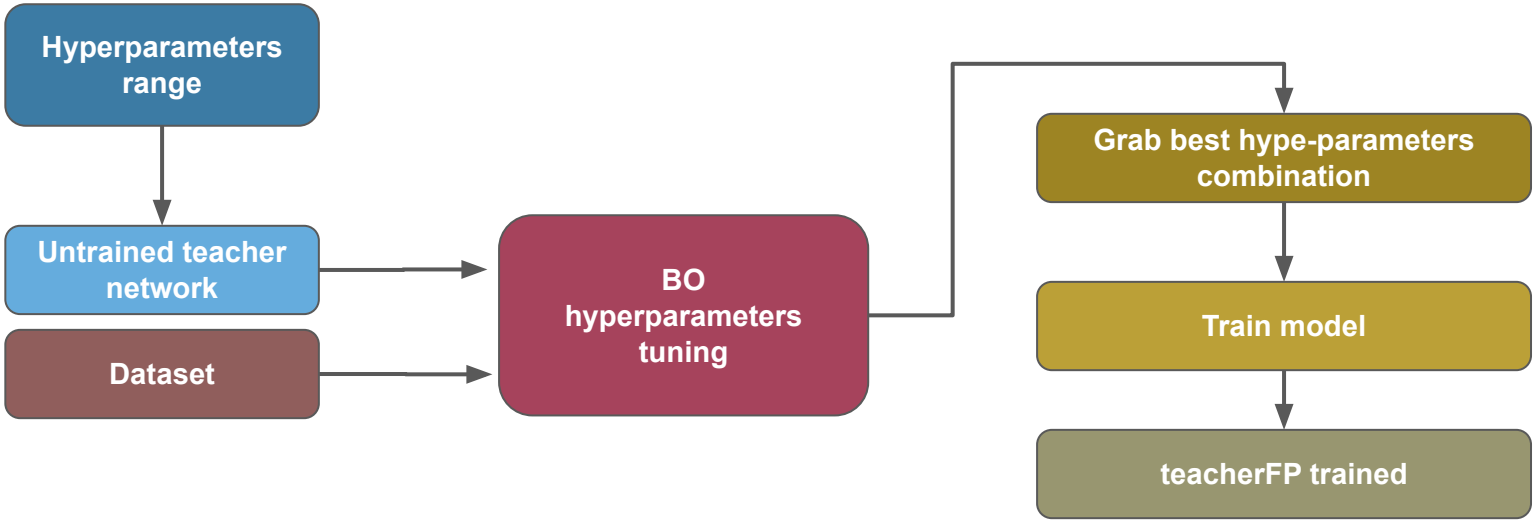


Available at https://github.com/RomiSolMolina/workflowCompressionML

A. DNN training and compression

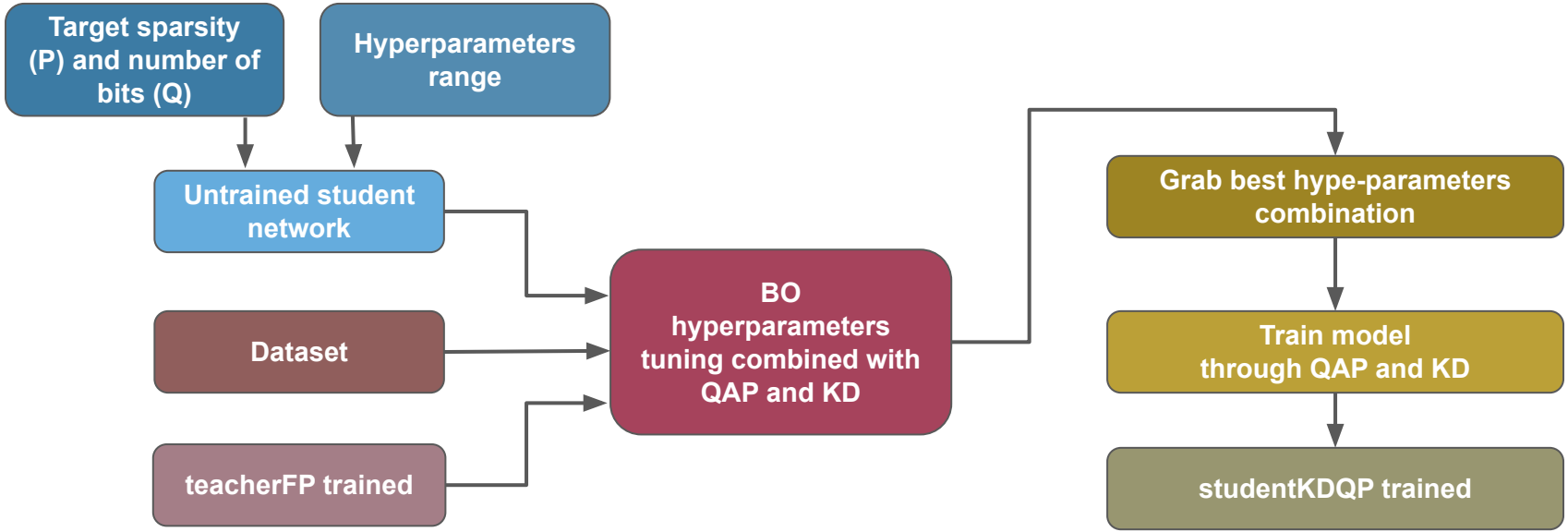
DNN training and compression

Stage 1 - Teacher training



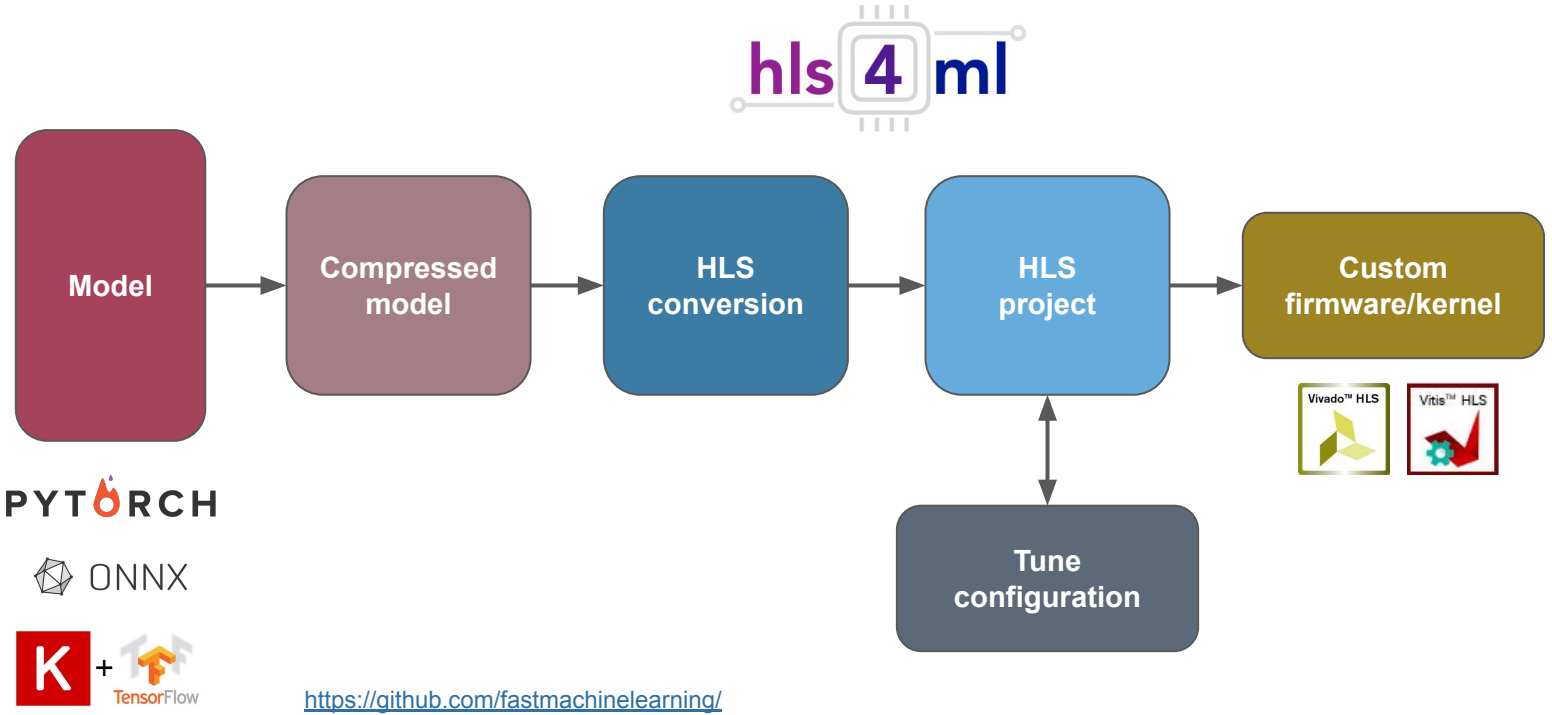
DNN training and compression

Stage 2 - Student training



B. Integration with a hardware synthesis tool for ML

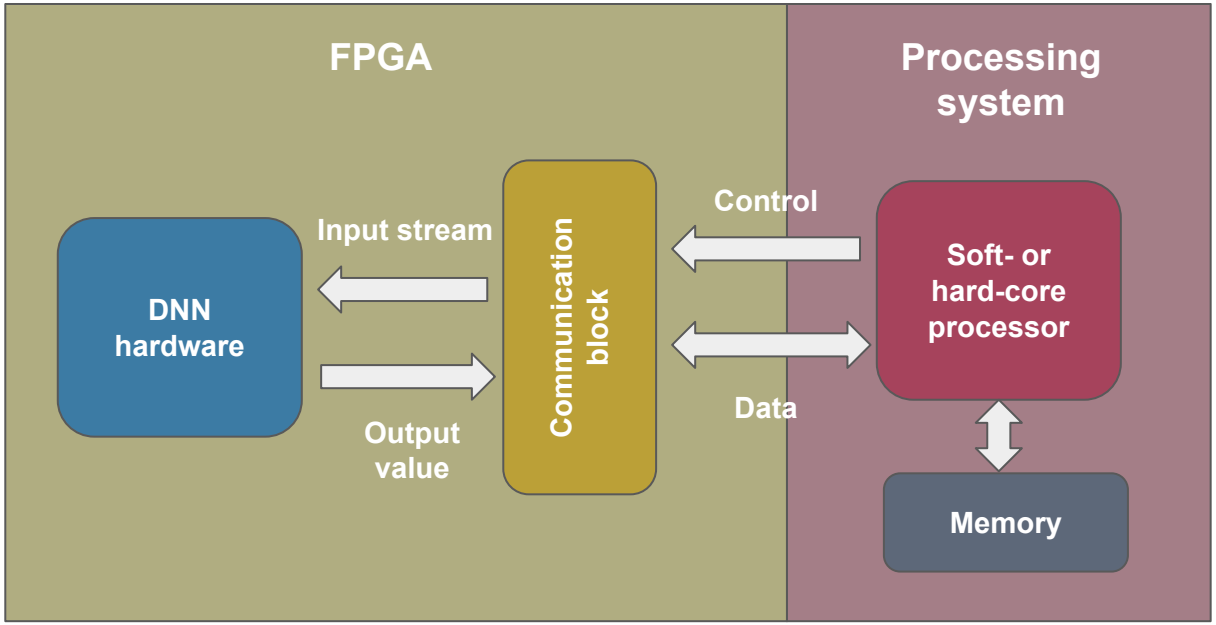
Integration with a hardware synthesis tool for ML



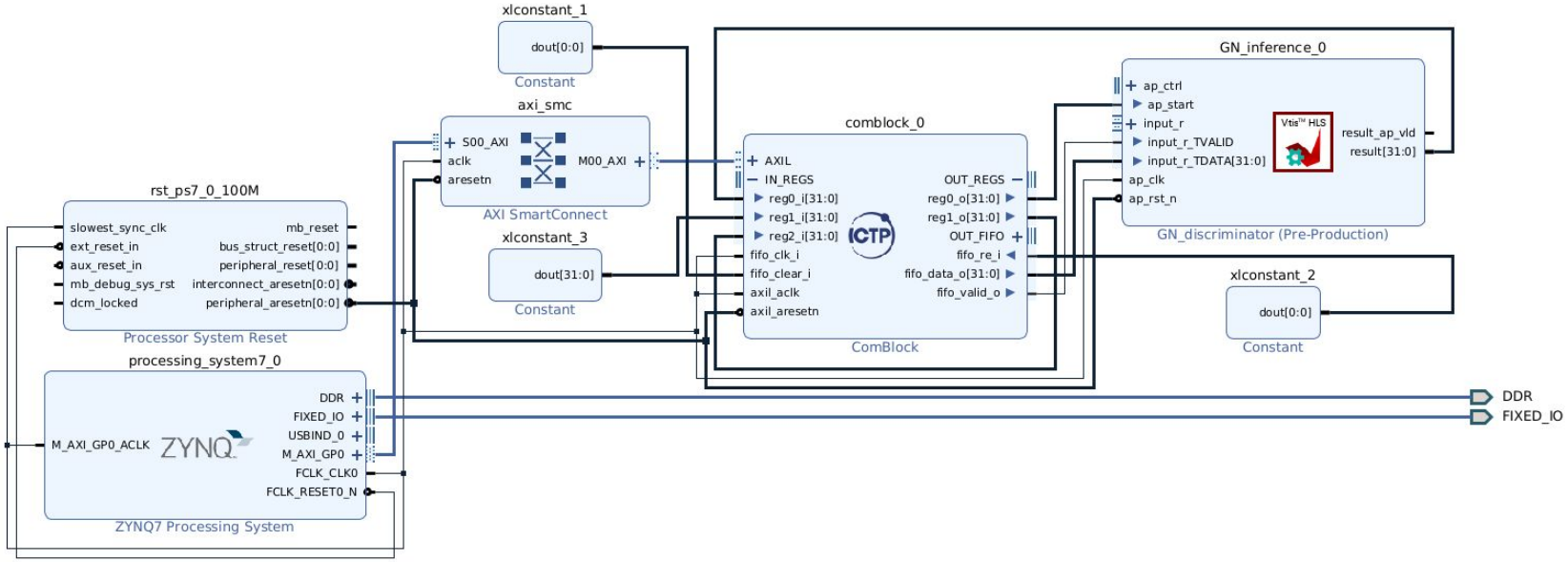
https://github.com/fastmachinelearning/

C. Hardware assessment framework

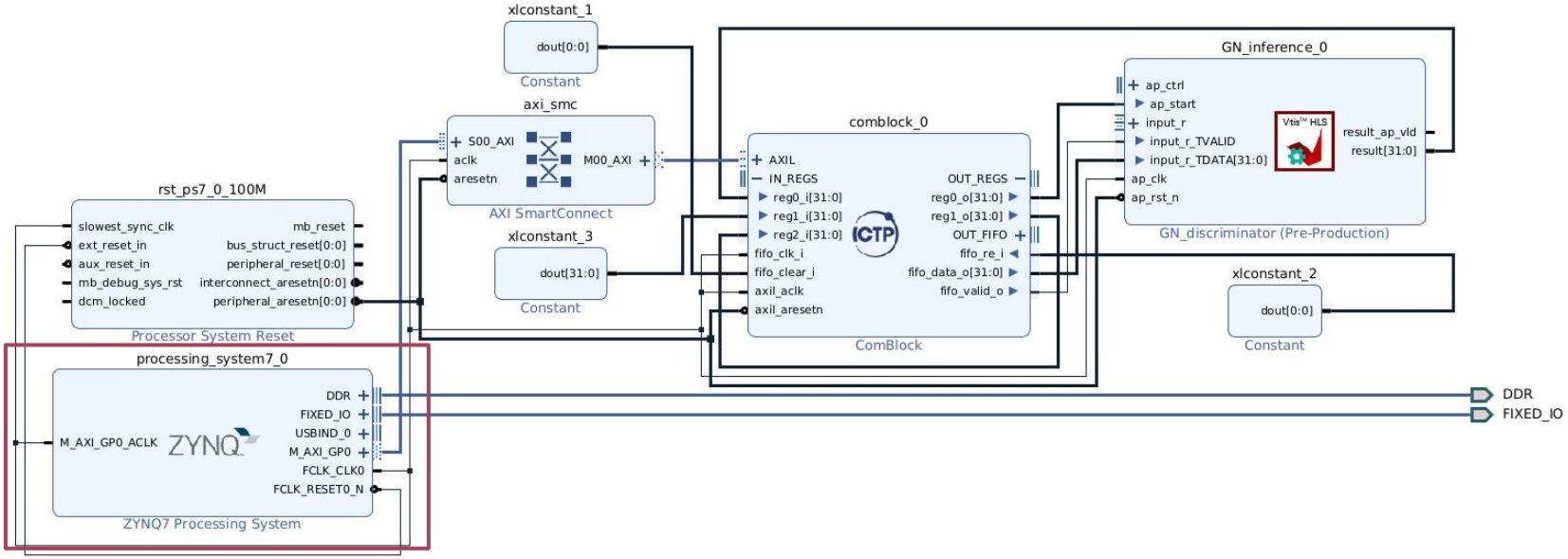
Hardware assessment framework



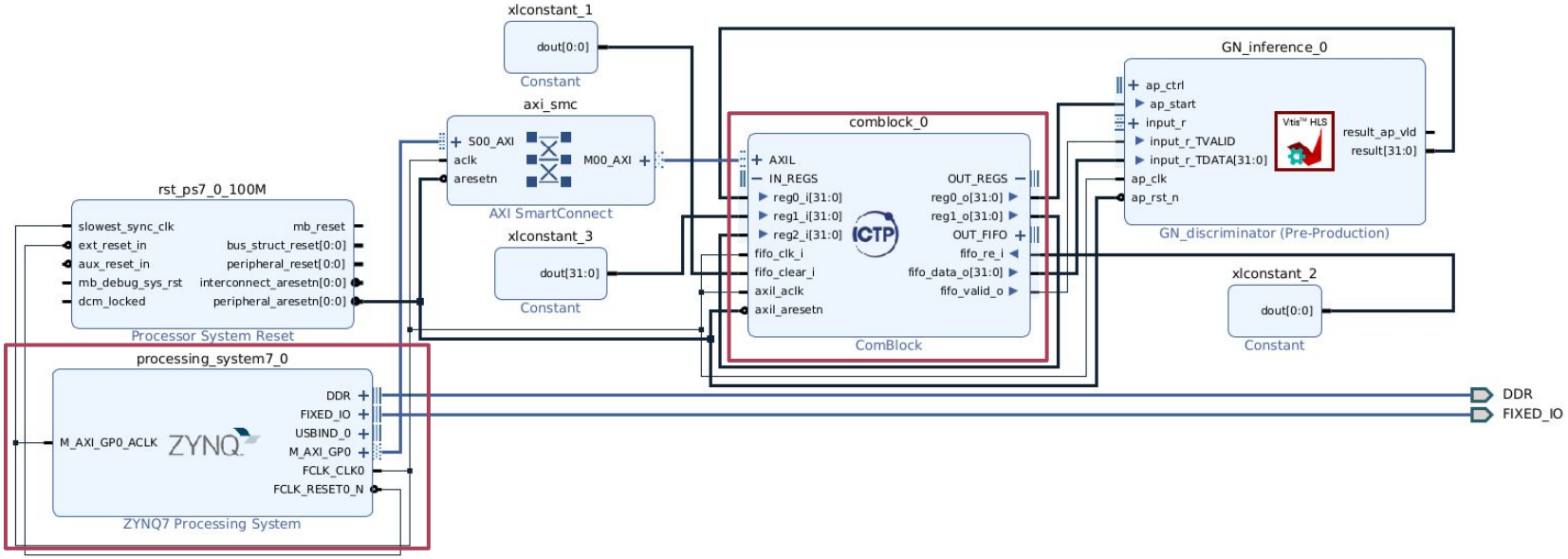
Hardware assessment framework



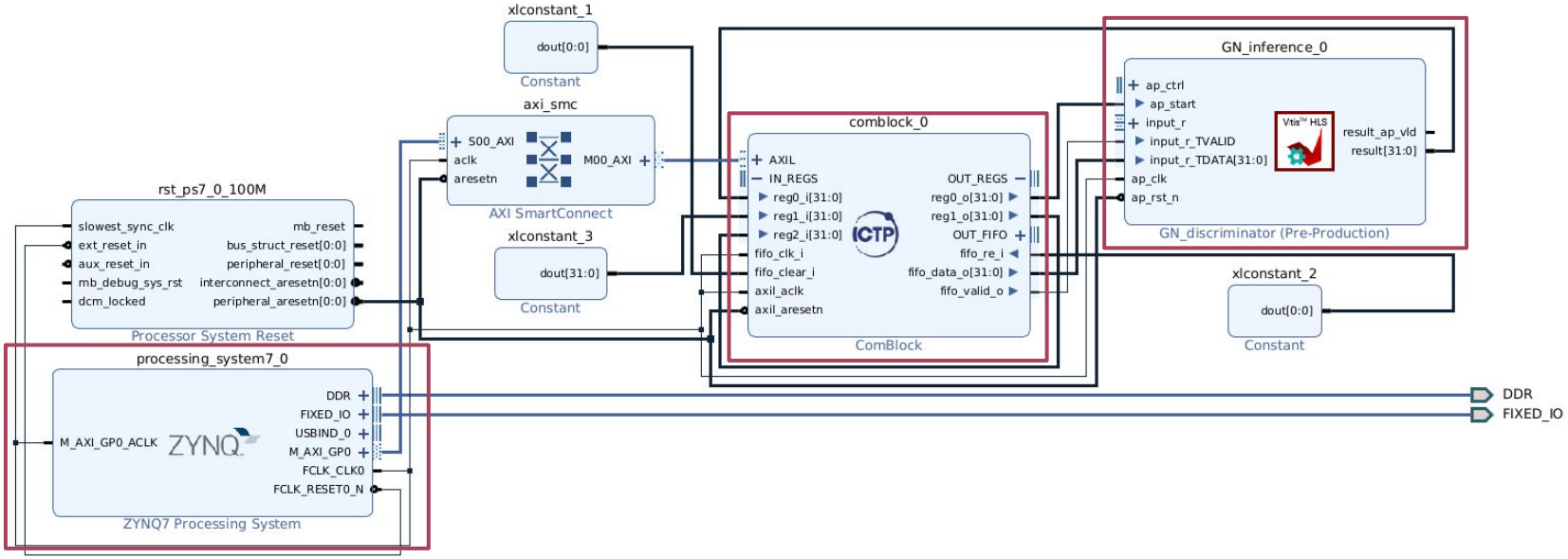
Hardware assessment framework



Hardware assessment framework



Hardware assessment framework



General step-by-step cheat sheet

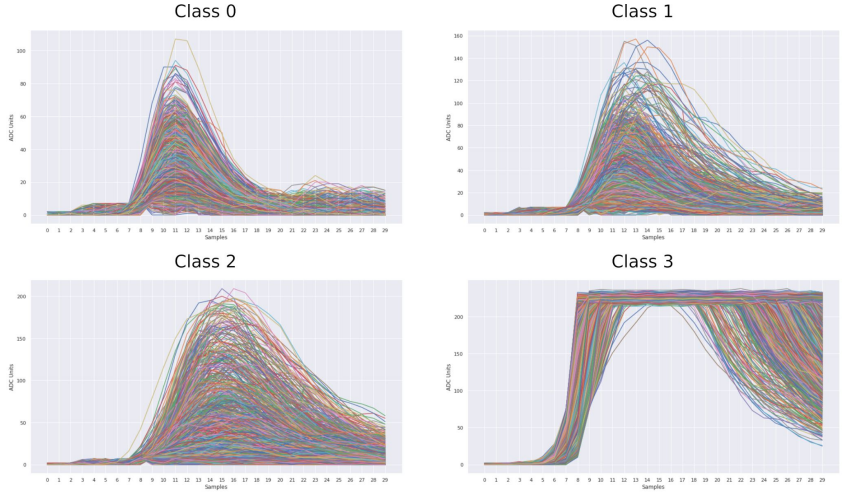
Bare-metal	PYNQ	Cluster
- Train the DNN model. - Compress the DNN model.		
- hsl4ml for HLS project creation for the specific FPGA part. - Export IP core.		
- Create hardware and export .xsa file.		
Create hardware platform.	Initialize PYNQ-based board.	Access to the HyperFPGA.
Create C application (interact with the communication block and the IP core).	Upload .xsa file.	Upload .xsa file.
Initialize serial communication.	Python code to interact with the ML-based hardware.	Python code to interact with the ML-based hardware.
Program FPGA, load .elf, and check output in the serial terminal.	Run the application.	Run the application.

Some results…

Results

Applications

- **1D-MLP** is focused on 1-D signals: a pulse shape discriminator (PSD) based on a Multi-Layer Perceptron (MLP), to be used for event recognition in cosmic rays studies.



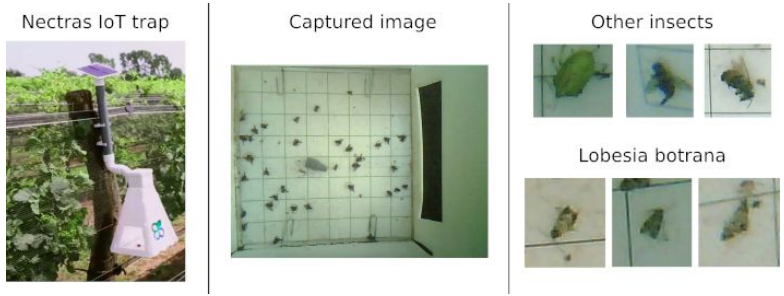
Molina, R. S., Morales, I. R., Crespo, M. L., Costa, V. G., Carrato, S., & Ramponi, G. (2023). An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA. *IEEE Embedded Systems Letters*, *16*(3), 255-258.

Results

Applications

- Application in the field of object classification in 2D-images; its aim is moth classification in the context of pest detection. The solution **2D-CNN** is based on ad-hoc CNN trained with a dataset obtained from in-field traps through an IoT system. This application was further developed in (2D-VGG16) using a larger pre-trained teacher network (VGG16), and a public dataset.

Ad-hoc CNN



2D-VGG16

Molina, R. S., Morales, I. R., Crespo, M. L., Costa, V. G., Carrato, S., & Ramponi, G. (2023). An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA. *IEEE Embedded Systems Letters*, *16*(3), 255-258.

Results

TABLE I

EVALUATION. THE ACRONYMS IN THE TABLE ARE P: PARAMETERS, CR: COMPRESSION RATIO, NL: NUMBER OF LAYERS (CONSIDERING ONLY CONVOLUTIONAL AND FULLY-CONNECTED LAYERS), OA: OVERALL ACCURACY, SP: SPARSITY, L: LATENCY WITHOUT DATA TRANSFER.

	Teacher model		Compressed student models						Hardware implementation of the compressed student models					
Application	**P**	**OA[%]**	**P**	**CR**	**NL**	**OA[%]**	**Bits**	**SP[%]**	**FPGA**	**BRAM [%]**	**DSP[%]**	**FF[%]**	**LUT[%]**	**L [clk]**
1D-MLP	16,352	99.7	529	30.91	1	98.96	8	20.00	Artix-7	0.00	14.00	2.75	26.27	10
2D-CNN	723,499	99.19	3,699	195.59	11	94.11	8	50.00	ZCU102	5.70	10.90	6.00	18.79	17,411
2D-VGG16	14,818,695	98.87	3,207	4,898.50	16	96.67	8	60.00	ZCU102	10.84	17.54	6.02	15.40	18,005

Molina, R. S., Morales, I. R., Crespo, M. L., Costa, V. G., Carrato, S., & Ramponi, G. (2023). An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA. *IEEE Embedded Systems Letters*, *16*(3), 255-258.

Results

TABLE I

EVALUATION. THE ACRONYMS IN THE TABLE ARE P: PARAMETERS, CR: COMPRESSION RATIO, NL: NUMBER OF LAYERS (CONSIDERING ONLY CONVOLUTIONAL AND FULLY-CONNECTED LAYERS), OA: OVERALL ACCURACY, SP: SPARSITY, L: LATENCY WITHOUT DATA TRANSFER.

	Teacher model		Compressed student models						Hardware implementation of the compressed student models					
Application	**P**	**OA[%]**	**P**	**CR**	**NL**	**OA[%]**	**Bits**	**SP[%]**	**FPGA**	**BRAM [%]**	**DSP[%]**	**FF[%]**	**LUT[%]**	**L [clk]**
1D-MLP	16,352	99.7	529	30.91	1	98.96	8	20.00	Artix-7	0.00	14.00	2.75	26.27	10
2D-CNN	723,499	99.19	3,699	195.59	11	94.11	8	50.00	ZCU102	5.70	10.90	6.00	18.79	17,411
2D-VGG16	14,818,695	98.87	3,207	4,898.50	16	96.67	8	60.00	ZCU102	10.84	17.54	6.02	15.40	18,005

Molina, R. S., Morales, I. R., Crespo, M. L., Costa, V. G., Carrato, S., & Ramponi, G. (2023). An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA. *IEEE Embedded Systems Letters*, *16*(3), 255-258.

Results

TABLE I

EVALUATION. THE ACRONYMS IN THE TABLE ARE P: PARAMETERS, CR: COMPRESSION RATIO, NL: NUMBER OF LAYERS (CONSIDERING ONLY CONVOLUTIONAL AND FULLY-CONNECTED LAYERS), OA: OVERALL ACCURACY, SP: SPARSITY, L: LATENCY WITHOUT DATA TRANSFER.

	Teacher model		Compressed student models						Hardware implementation of the compressed student models					
Application	**P**	**OA[%]**	**P**	**CR**	**NL**	**OA[%]**	**Bits**	**SP[%]**	**FPGA**	**BRAM [%]**	**DSP[%]**	**FF[%]**	**LUT[%]**	**L [clk]**
1D-MLP	16,352	99.7	529	30.91	1	98.96	8	20.00	Artix-7	0.00	14.00	2.75	26.27	10
2D-CNN	723,499	99.19	3,699	195.59	11	94.11	8	50.00	ZCU102	5.70	10.90	6.00	18.79	17,411
2D-VGG16	14,818,695	98.87	3,207	4,898.50	16	96.67	8	60.00	ZCU102	10.84	17.54	6.02	15.40	18,005

Molina, R. S., Morales, I. R., Crespo, M. L., Costa, V. G., Carrato, S., & Ramponi, G. (2023). An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA. *IEEE Embedded Systems Letters*, *16*(3), 255-258.

Joint ICTP-IAEA School on Detector Signal Processing and Machine Learning for Scientific Instrumentation and Reconfigurable Computing

Using FPGAs to Accelerate Machine Learning Algorithms

smr4110 | Trieste, Italy - 2025

Romina Soledad Molina, Ph.D.

IAEA International Atomic Energy Agency
Atoms for Peace and Development

The Abdus Salam International Centre for Theoretical Physics

Complementary Frameworks for AI on FPGAs

Complementary Frameworks for AI on FPGAs

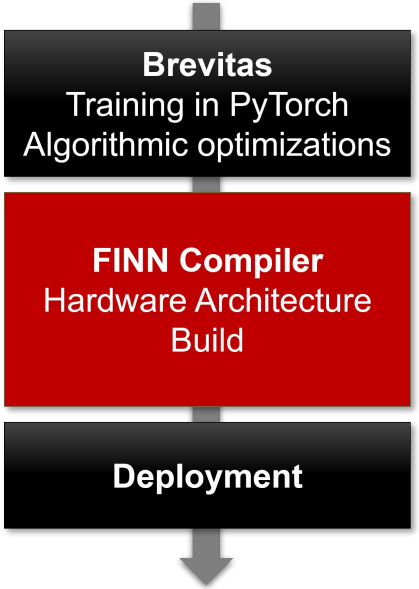
PYNQ · FINN · Vitis AI

- **PYNQ**
  - Stands for Python Productivity for Zynq.
  - Open-source framework from Xilinx.
  - Provides libraries and Jupyter notebooks for quick prototyping.
  - Focus: Ease of use & education.

Complementary Frameworks for AI on FPGAs

PYNQ · FINN · Vitis AI

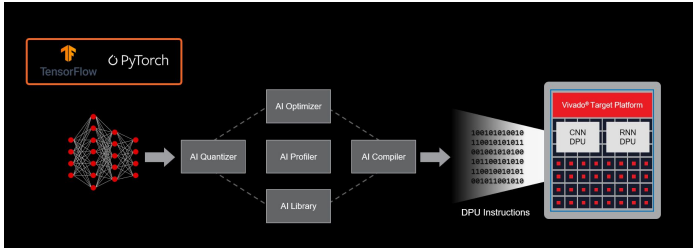
- **FINN (Overview)**
  - Developed by Xilinx Research Labs.
  - Specializes in quantized neural networks (QNNs).
  - Generates dataflow-style FPGA accelerators.
  - Optimized for ultra-low latency & high throughput.
  - Focus: Efficient deep learning inference.



Complementary Frameworks for AI on FPGAs

PYNQ · FINN · Vitis AI

- **Vitis AI (Overview)**
  - Vitis AI (Overview)
  - Xilinx’s official AI development platform
  - Supports TensorFlow, PyTorch, Caffe models
  - Provides optimized DPU (Deep Processing Unit) for FPGAs/SoCs
  - Toolchain includes model quantization, pruning, and compilation
  - Focus: Production-ready AI acceleration



XILINX VITIS™ AI

Complementary Frameworks for AI on FPGAs

PYNQ · FINN · Vitis AI

- **How They Complement Each Other**
  - **PYNQ** → Rapid prototyping & education
  - **FINN** → Research & deployment of QNNs
  - **Vitis AI** → Industrial-grade AI acceleration
  - **hls4ml** → Converts ML models into FPGA firmware via HLS

Complementary Frameworks for AI on FPGAs

PYNQ · FINN · Vitis AI

- **How They Complement Each Other**
  - **PYNQ** → Rapid prototyping & education
  - **FINN** → Research & deployment of QNNs
  - **Vitis AI** → Industrial-grade AI acceleration
  - **hls4ml** → Converts ML models into FPGA firmware via HLS

Together: learning → prototyping → research → production pipeline

Complementary Frameworks for AI on FPGAs

PYNQ · FINN · Vitis AI

Key Takeaways

PYNQ: Python-friendly FPGA development

FINN: QNN accelerators, low latency

Vitis AI: Full-stack AI deployment

hls4ml: Bridges ML frameworks with FPGA design

Complementary Frameworks for AI on FPGAs

PYNQ · FINN · Vitis AI

Key Takeaways

PYNQ: Python-friendly FPGA development

FINN: QNN accelerators, low latency

Vitis AI: Full-stack AI deployment

hls4ml: Bridges ML frameworks with FPGA design

Complementary roles, not competitors
Enable innovation across education, research, prototyping, and industry