The Abdus Salam International Centre for Theoretical Physics

Workshop on Fully Programmable Systems-on-Chip for Scientific Applications

FPGA for Accelerating Machine Learning (ML) Algorithms

Romina Soledad Molina

MLab/STI Unit - ICTP

Doha, Qatar
2024

IAEA International Atomic Energy Agency *Atoms for Peace and Development*

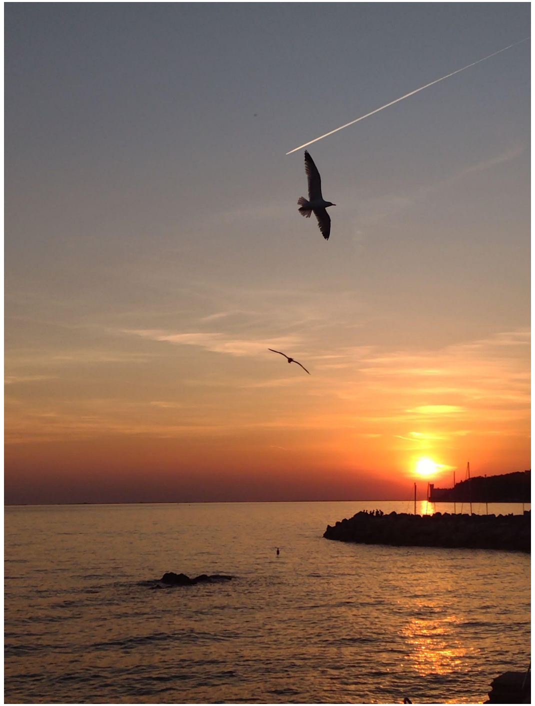
IUB Independent University, Bangladesh

جامعة قطر QATAR UNIVERSITY

Universiti Kebangsaan Malaysia *The National University of Malaysia*

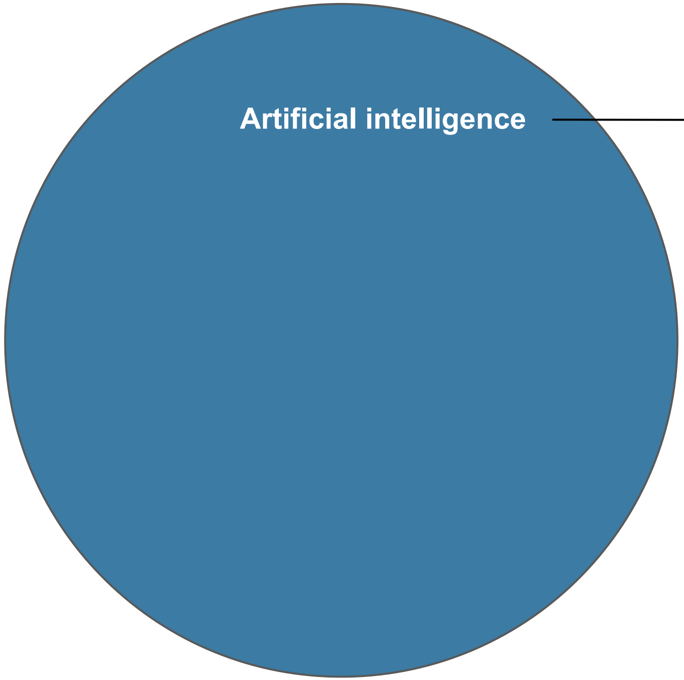
Outline

- Introduction.
- Remarks from SOTA.
- ML and model compression techniques.
- An end-to-end workflow to compress and deploy DNN on FPGA.
  - DNN training and compression.
  - Integration with a hardware synthesis tool for ML.
  - Hardware assessment framework.
- Applications.
- Final remarks.



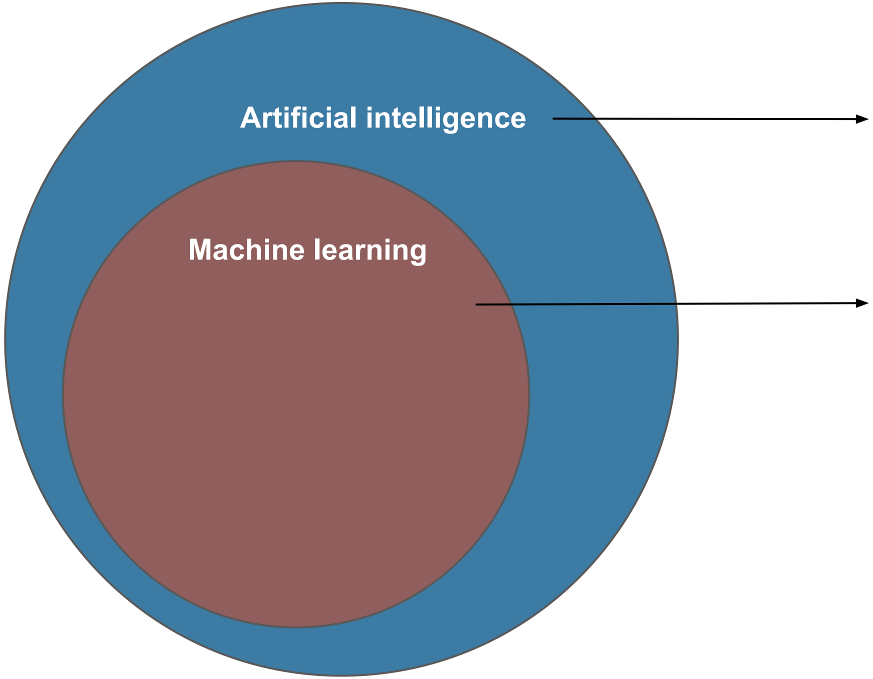
Introduction

Introduction



A machine/system capable of imitating human behavior

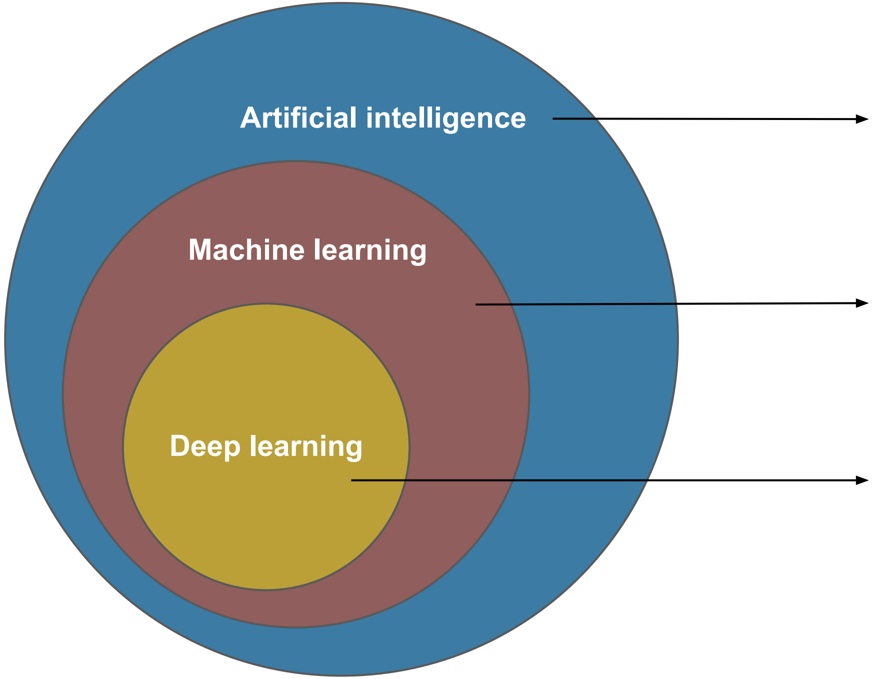
Introduction



A machine/system capable of imitating human behavior

Mathematics and statistics to learn from data.

Introduction



A machine/system capable of imitating human behavior

Mathematics and statistics to learn from data.

Neural networks multi-layer

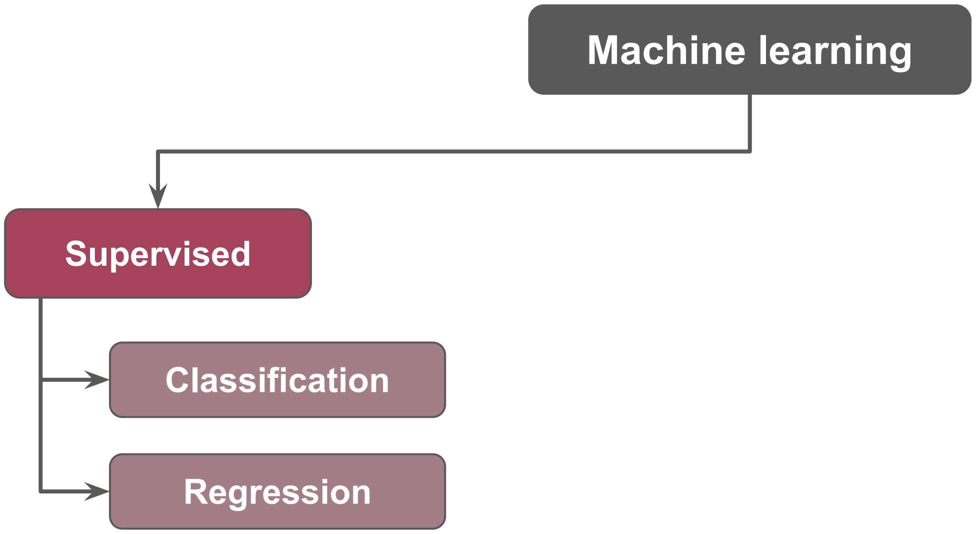
Machine learning

Classification

Machine learning

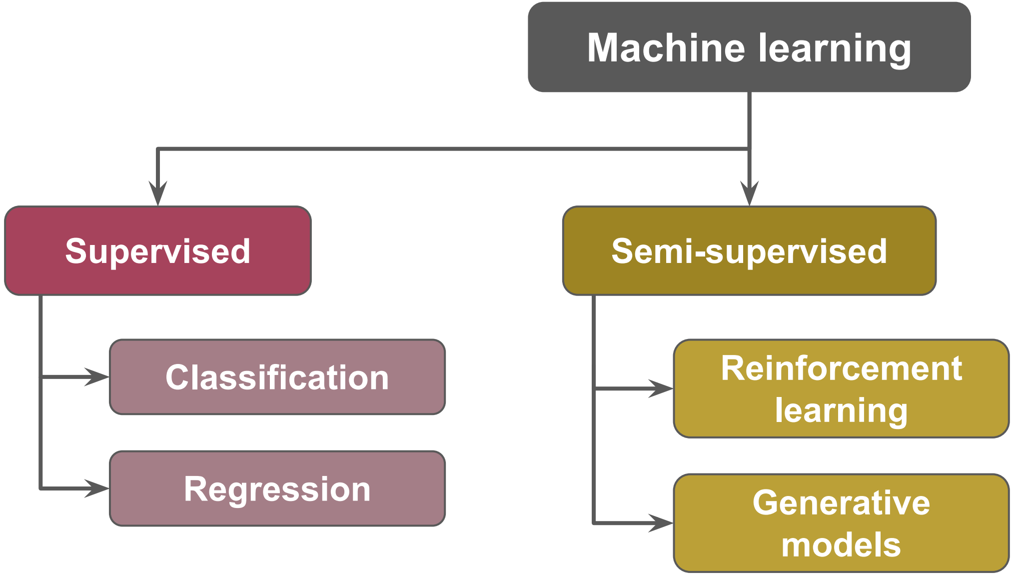
Machine learning

Classification



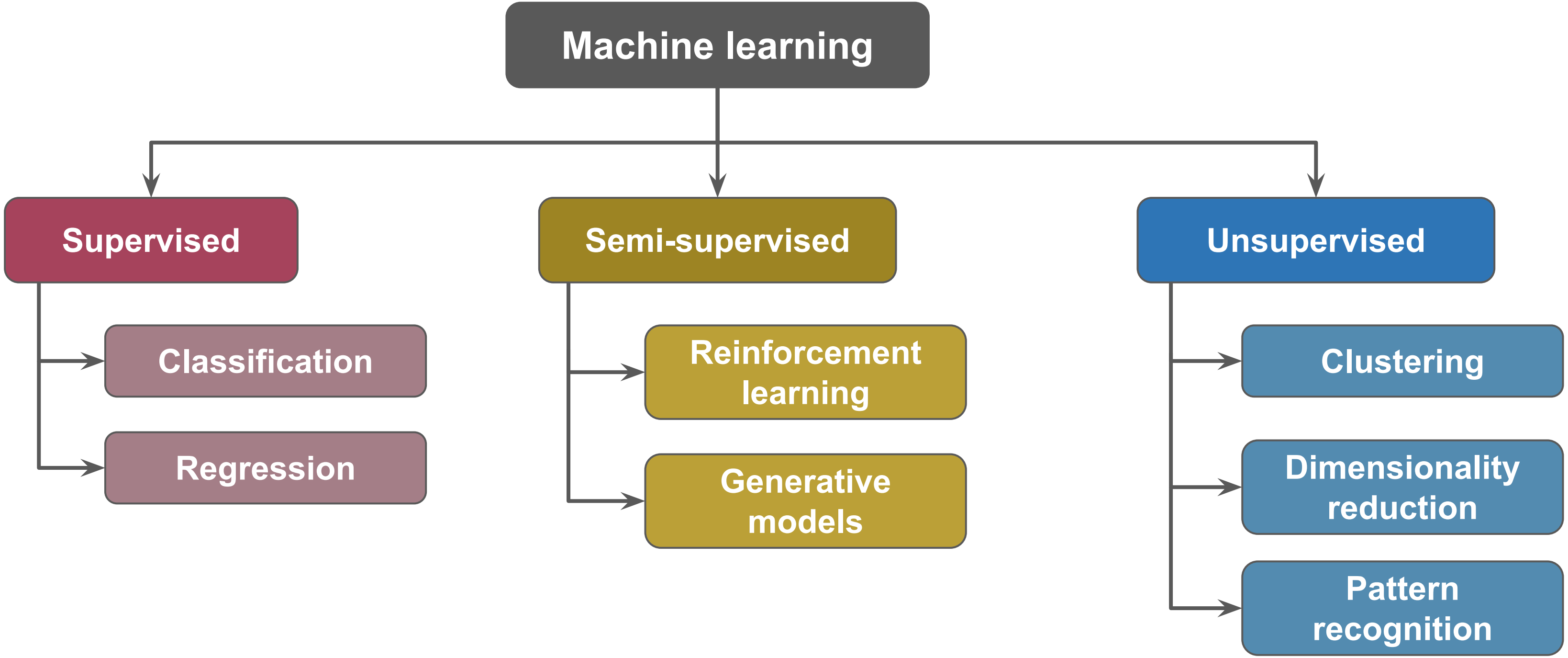
Machine learning

Classification

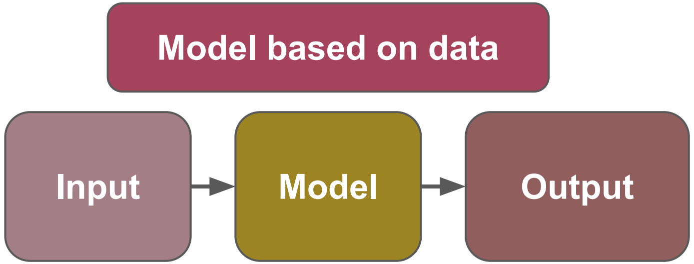


Machine learning

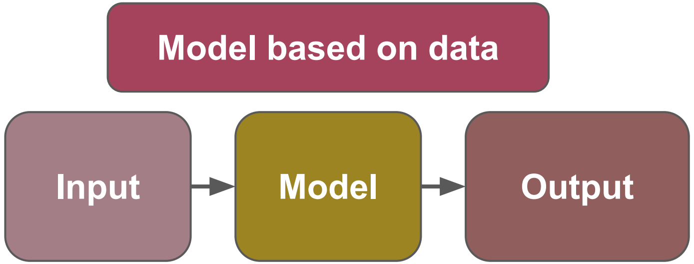
Classification

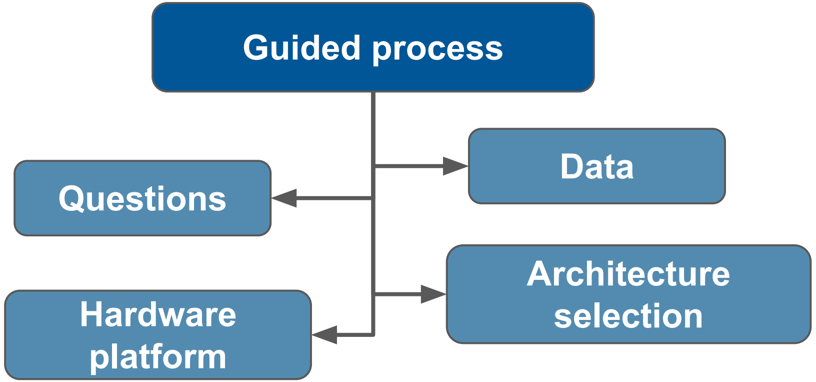


Machine learning

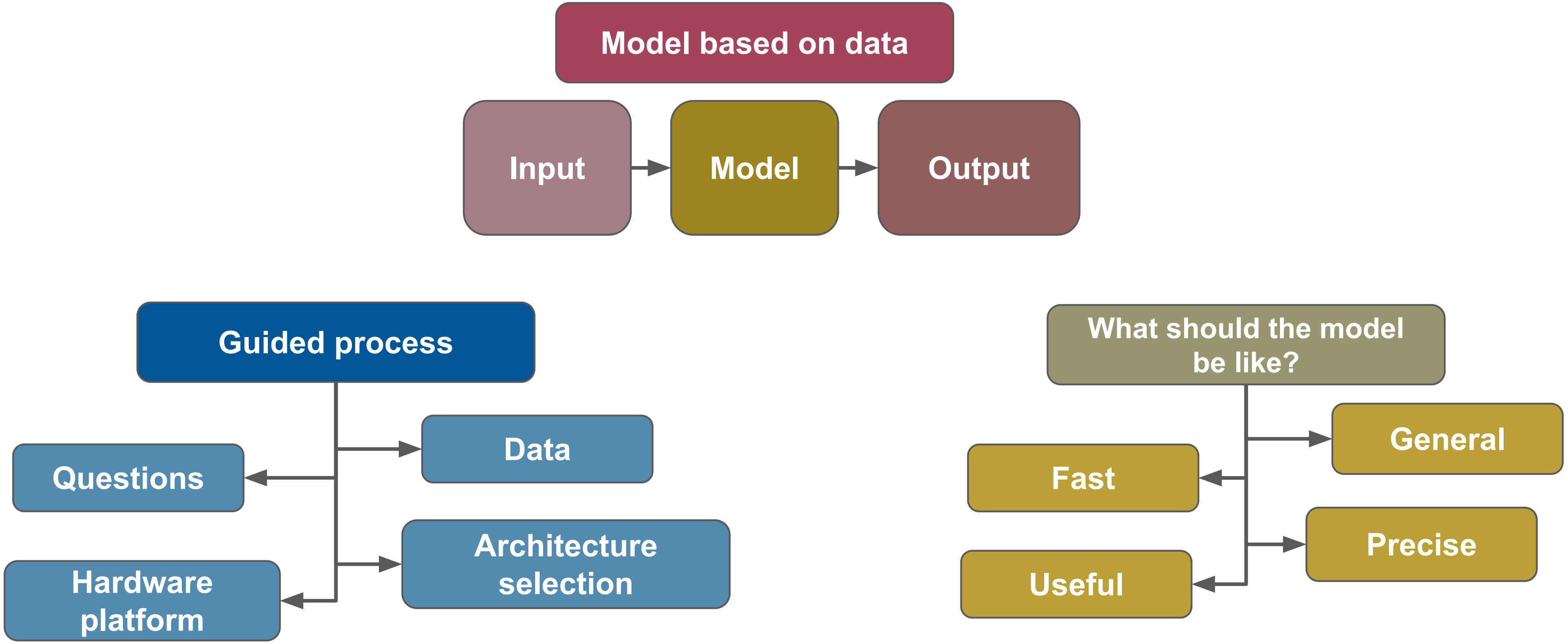


Machine learning





Machine learning

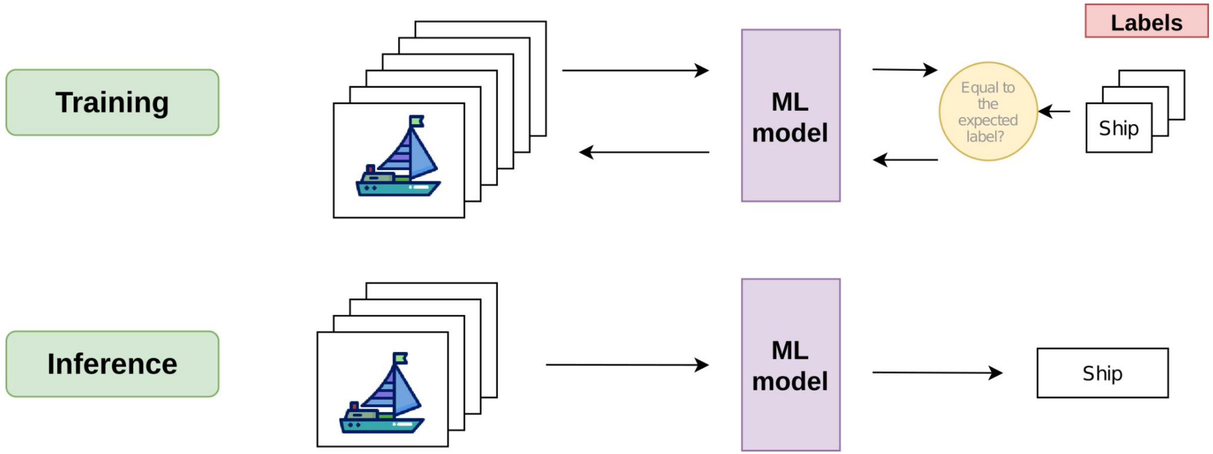


Machine learning

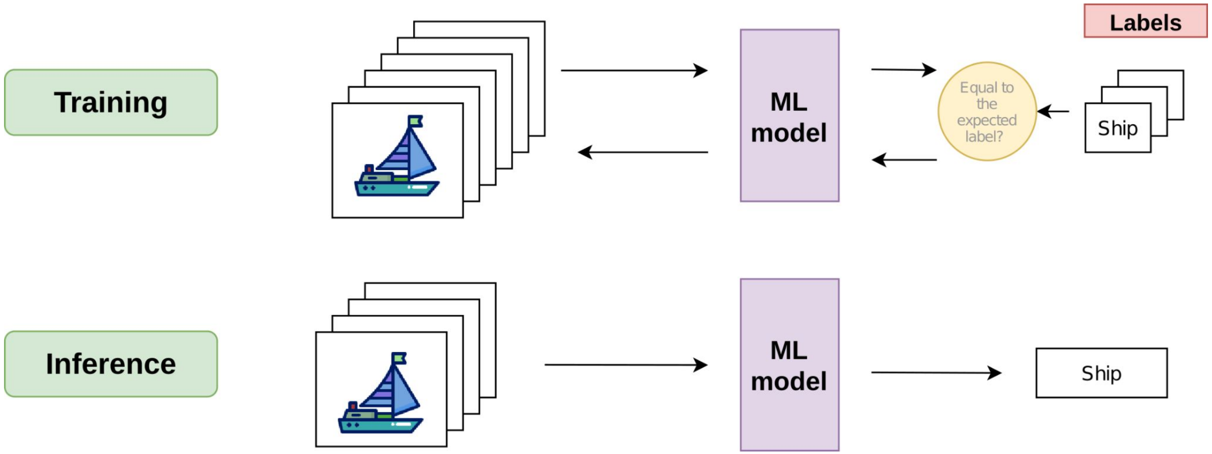
Generalization

Image from
Togootogtokh, E., & Amartuvshin, A. (2018). Deep Learning Approach for Very Similar Objects Recognition Application on Chihuahua and Muffin Problem. *ArXiv, abs/1801.09573*.

Machine learning for classification



Machine learning for classification



- In a classifier, **an input is mapped to a specific class**.

- Supervised training phase: **the network compares its current output with the desired output**. The difference between these two values is corrected using backpropagation.

Machine learning for classification

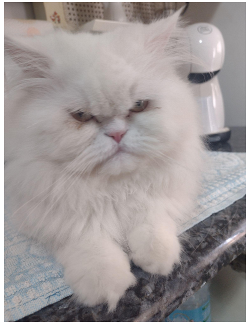
A classifier as example

Input → ML model → Prediction

Machine learning for classification

A classifier as example

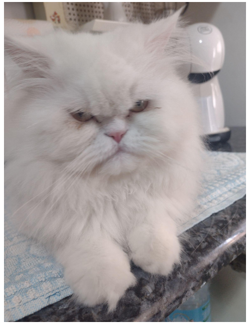
Input → ML model → Prediction



Machine learning for classification

A classifier as example

Input → ML model → Prediction

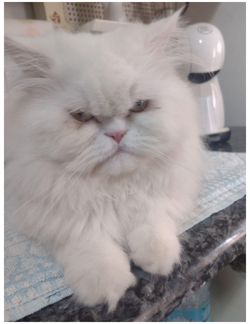


→ ML model

Machine learning for classification

A classifier as example

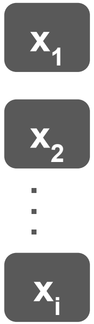
Input → ML model → Prediction



→ ML model → Cat

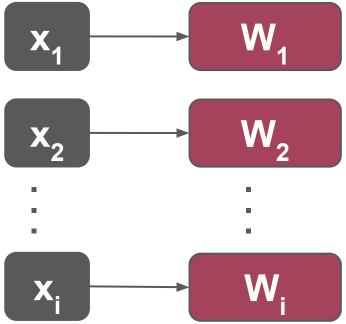
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



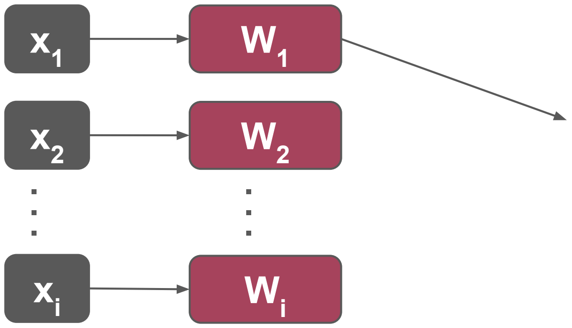
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



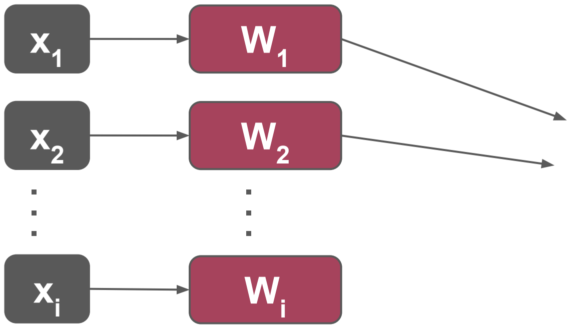
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



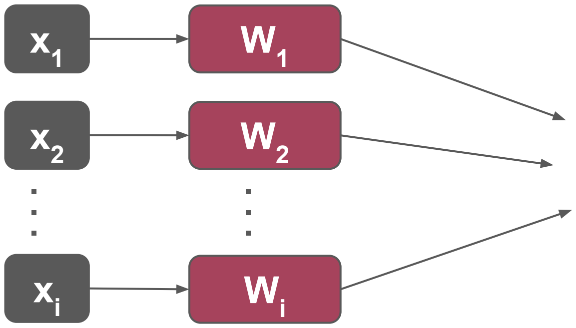
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



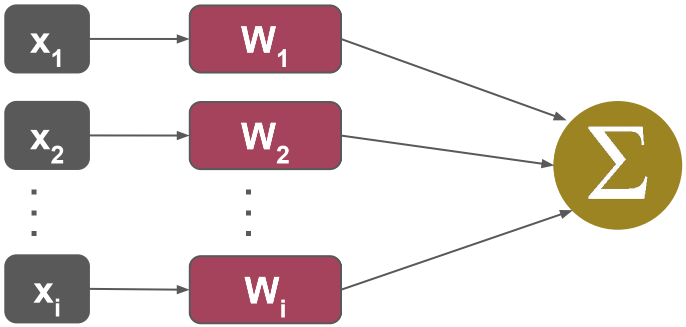
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



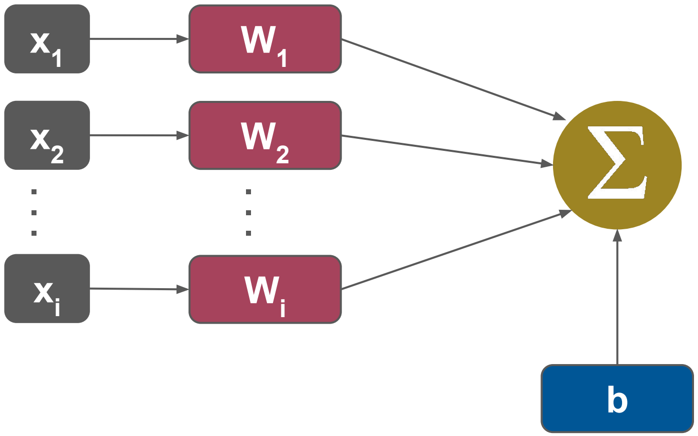
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



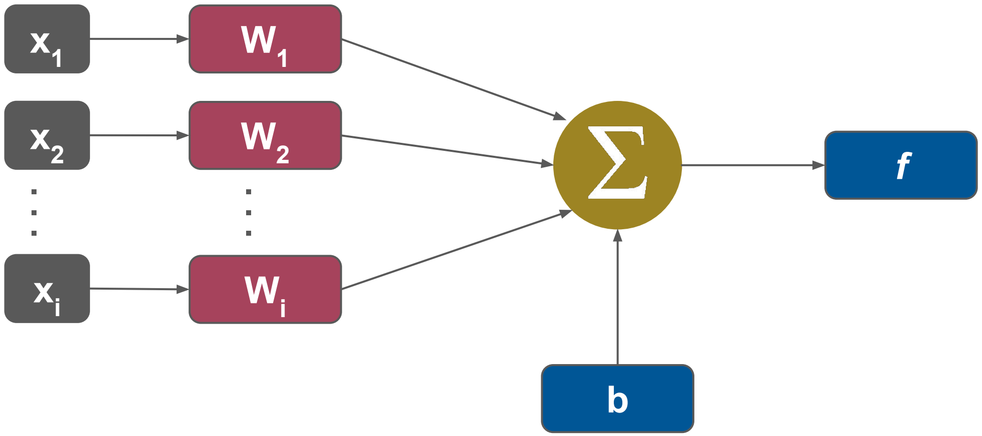
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



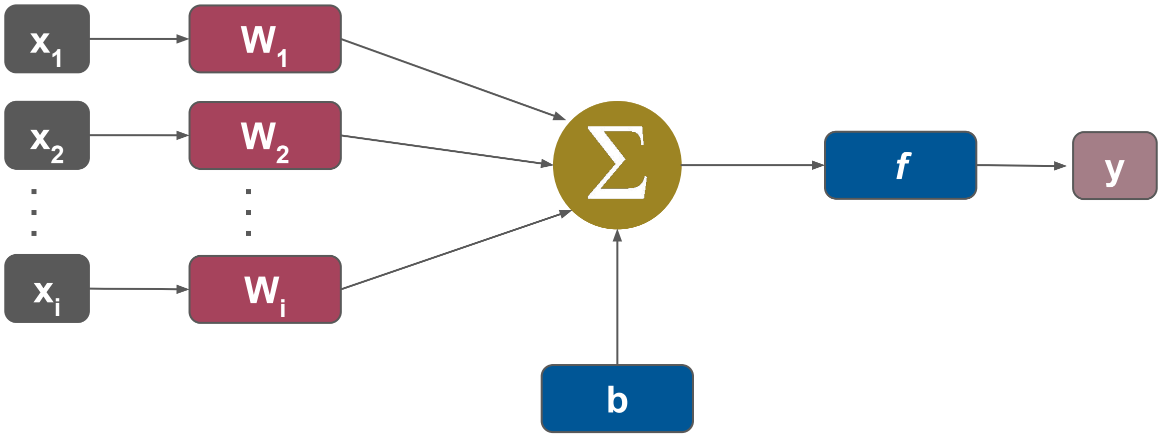
Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



Neural networks

An Artificial Neural Network (ANN) is composed of interconnected neurons (or nodes) arranged in multiple layers.



Introduction:
ML and SoC

Introduction

SoC-based FPGA

Introduction

Low latency

SoC-based FPGA

Introduction

Low latency

Low power consumption

SoC-based FPGA

Introduction

Low latency

Low power consumption

High parallelism

SoC-based FPGA

Introduction

Low latency

Low power consumption

High parallelism

SoC-based FPGA

Resource-constrained devices

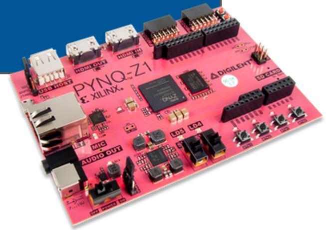
Introduction

Compression techniques

Original ML-based model

Compressed ML-based model

Embedded system (SoC-based FPGA)



Remarks from the SOTA

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

. Compression: focused on pruning and quantization.

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

. Compression: focused on pruning and quantization.

. Workflows addressing some parts of the development cycle.

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

. Compression: focused on pruning and quantization.

. Workflows addressing some parts of the development cycle.

. Off-chip memory transactions.

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

Memory footprint and latency

. Compression: focused on pruning and quantization.

. Workflows addressing some parts of the development cycle.

. Off-chip memory transactions.

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

. Compression: focused on pruning and quantization.

. Workflows addressing some parts of the development cycle.

. Off-chip memory transactions.

Memory footprint and latency

Ensemble of compression techniques

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

. Compression: focused on pruning and quantization.

. Workflows addressing some parts of the development cycle.

. Off-chip memory transactions.

Memory footprint and latency

Ensemble of compression techniques

On-chip memory deployment

Remarks from the SOTA

. Towards ML-based models implemented on resource-constrained devices.

. SOTA models: VGG16, MobileNet V2, BERT, U-Net, YOLO.

. Compression: focused on pruning and quantization.

. Workflows addressing some parts of the development cycle.

. Off-chip memory transactions.

Memory footprint and latency

Ensemble of compression techniques

On-chip memory deployment

Productivity

End-to-end workflow

ML and model compression techniques

ML and model compression techniques

for reconfigurable hardware accelerators

Ensemble of compression techniques - Exploration of the interplay between:

ML and model compression techniques
for reconfigurable hardware accelerators

Ensemble of compression techniques - Exploration of the interplay between:

Pruning	Quantization	Knowledge distillation

ML and model compression techniques
for reconfigurable hardware accelerators

Ensemble of compression techniques - Exploration of the interplay between:

Pruning	Quantization	Knowledge distillation

Remove neurons and connections.

ML and model compression techniques
for reconfigurable hardware accelerators

Ensemble of compression techniques - Exploration of the interplay between:

Pruning → **Remove neurons and connections.**

Quantization → **Selection of the number of bits to represent the weights and bias.**

Knowledge distillation

ML and model compression techniques
for reconfigurable hardware accelerators

Ensemble of compression techniques - Exploration of the interplay between:

Pruning	Quantization	Knowledge distillation
Remove neurons and connections.	Selection of the number of bits to represent the weights and bias.	Transfers the knowledge from a teacher network to a smaller and faster target network.

ML and model compression techniques
for reconfigurable hardware accelerators

Ensemble of compression techniques - Exploration of the interplay between:

Pruning → **Remove neurons and connections.**

Quantization → **Selection of the number of bits to represent the weights and bias.**

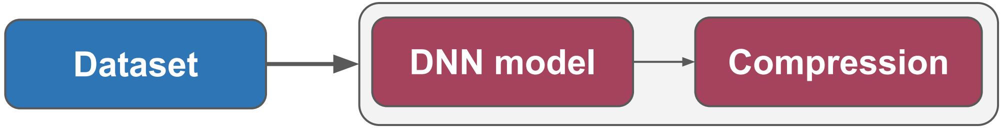
Knowledge distillation → **Transfers the knowledge from a teacher network to a smaller and faster target network.**

Fully on-chip deployment

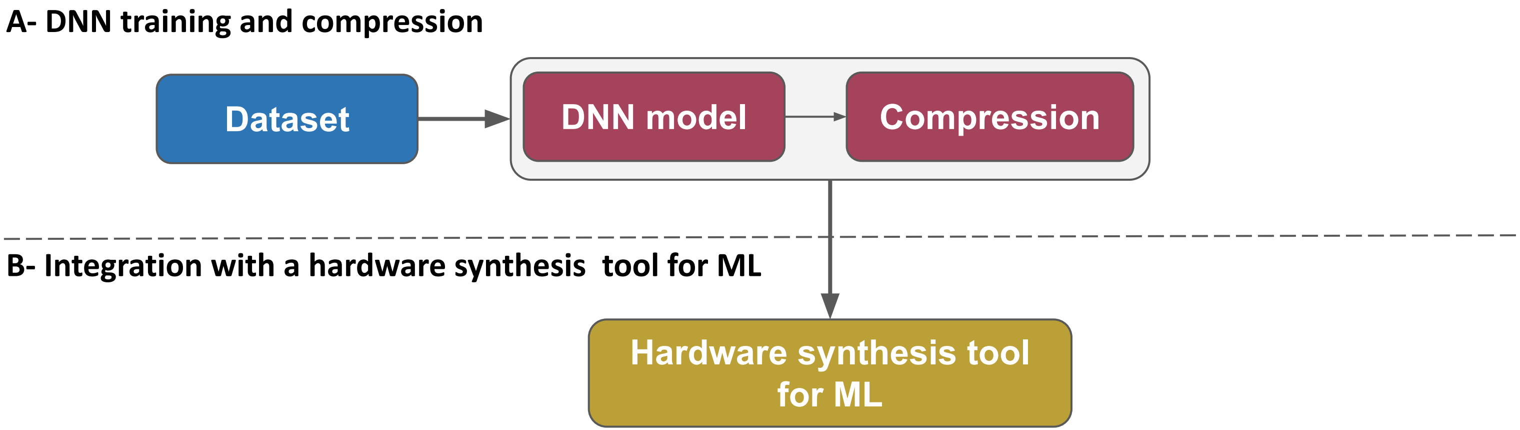
An end-to-end workflow to efficiently compress and deploy DNN on SoC/FPGA

End-to-end workflow

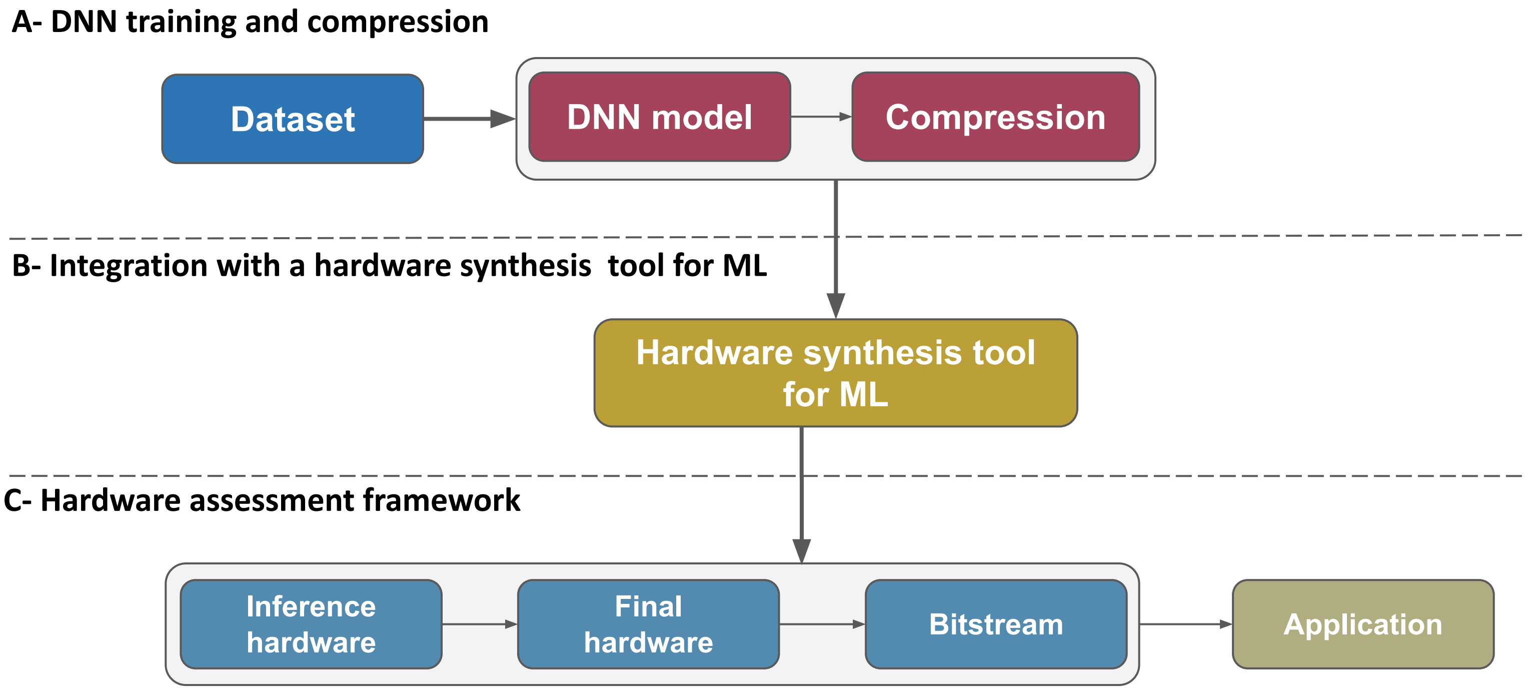
A- DNN training and compression



End-to-end workflow



End-to-end workflow

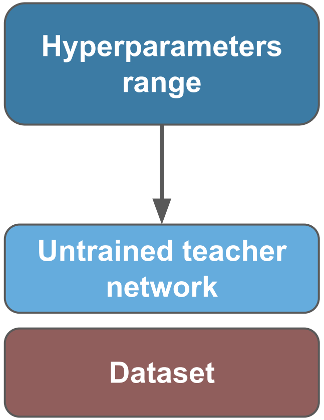


[Available at https://github.com/RomiSolMolina/workflowCompressionML](https://github.com/RomiSolMolina/workflowCompressionML)

A. DNN training and compression

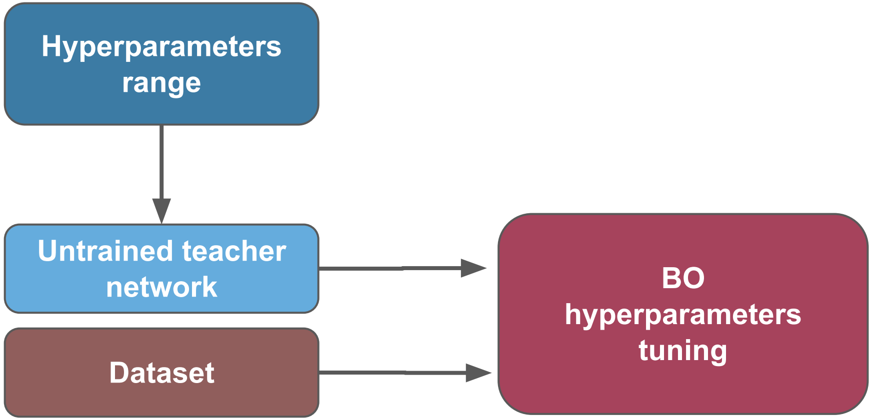
DNN training and compression

Stage 1 - Teacher training



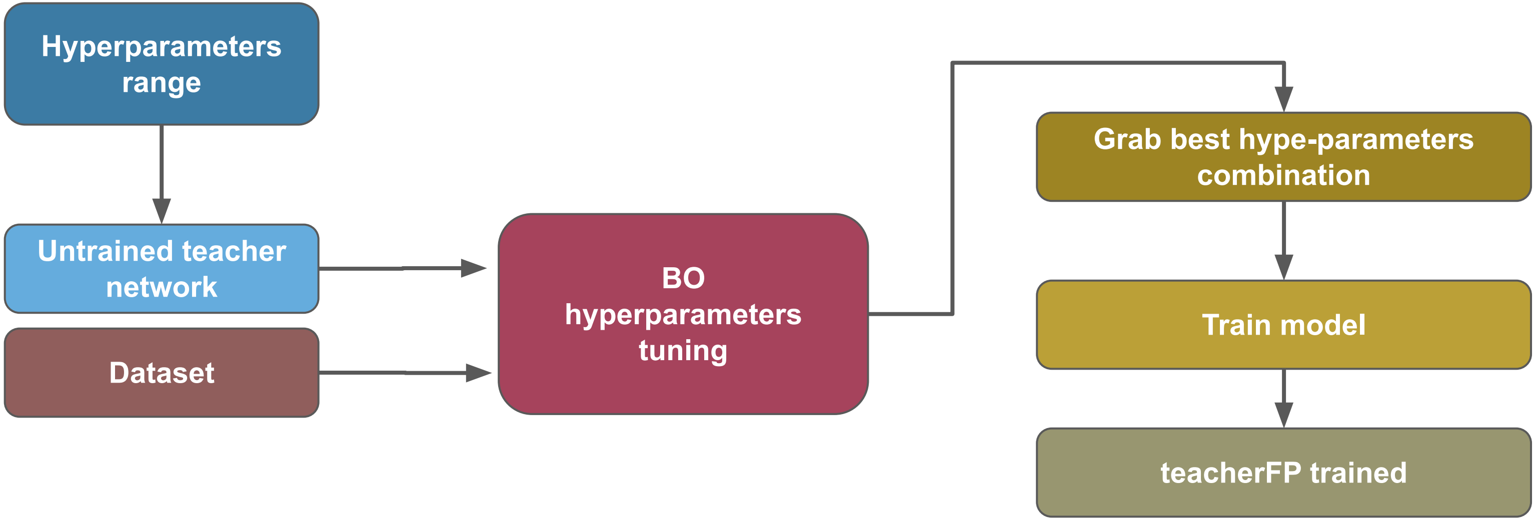
DNN training and compression

Stage 1 - Teacher training



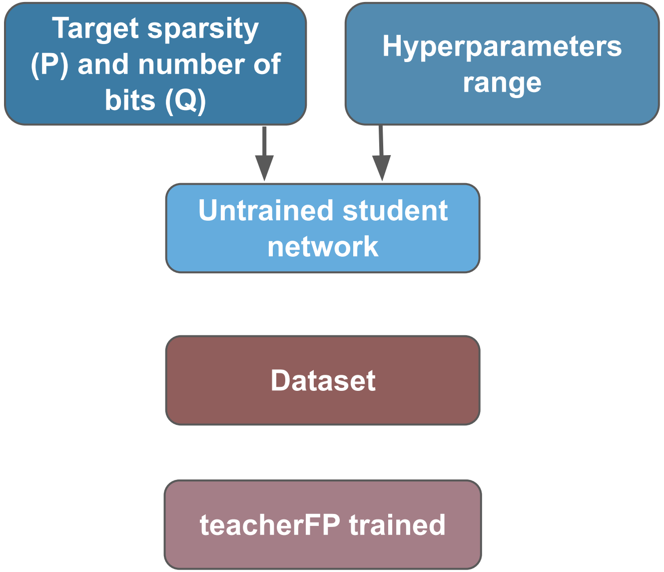
DNN training and compression

Stage 1 - Teacher training



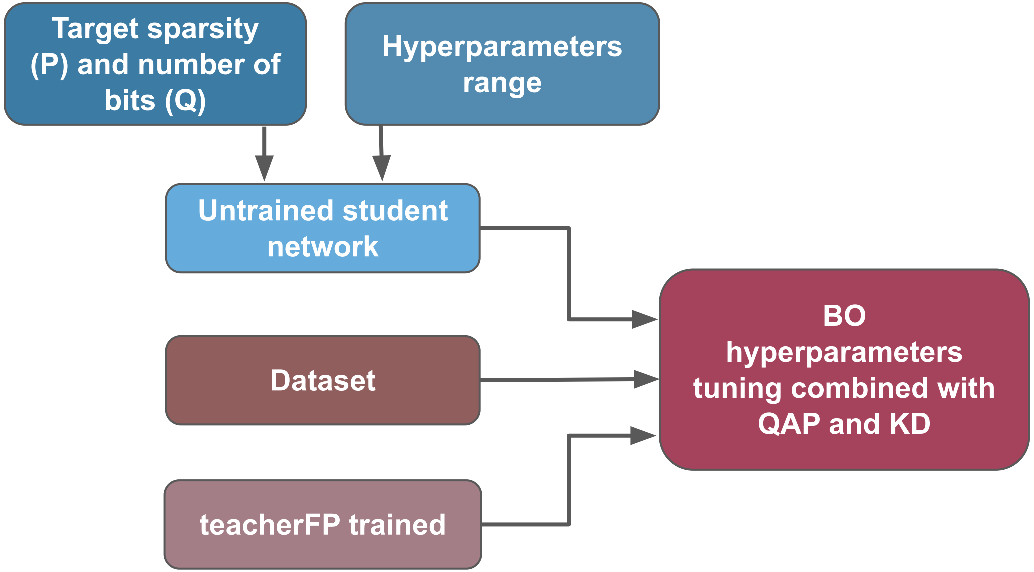
DNN training and compression

Stage 2 - Student training



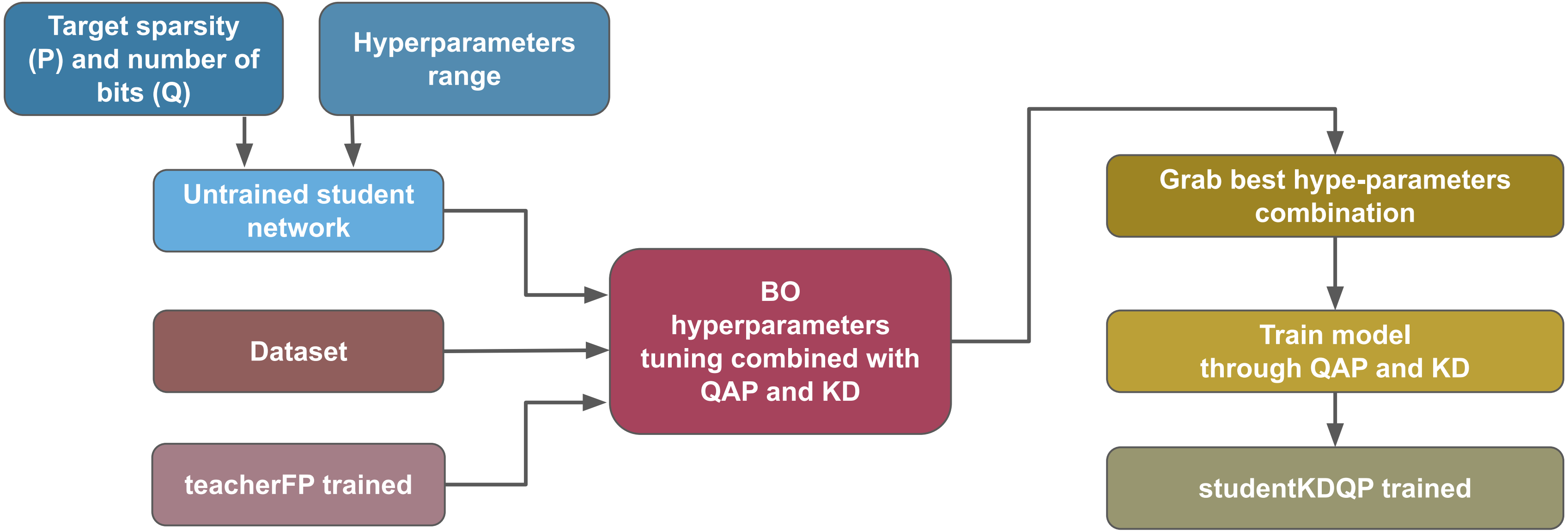
DNN training and compression

Stage 2 - Student 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

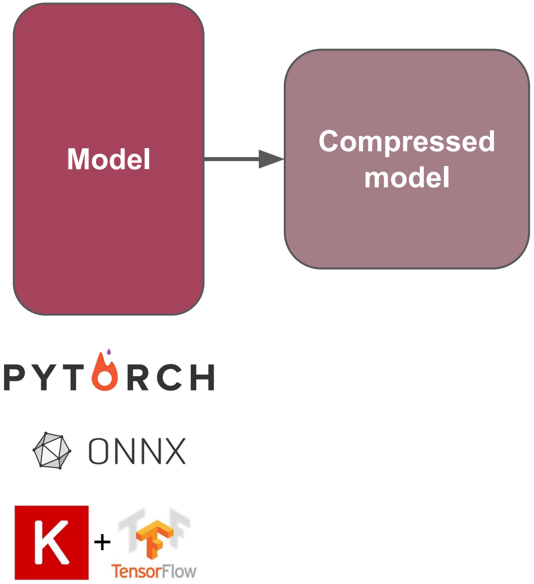
Model

PYTORCH

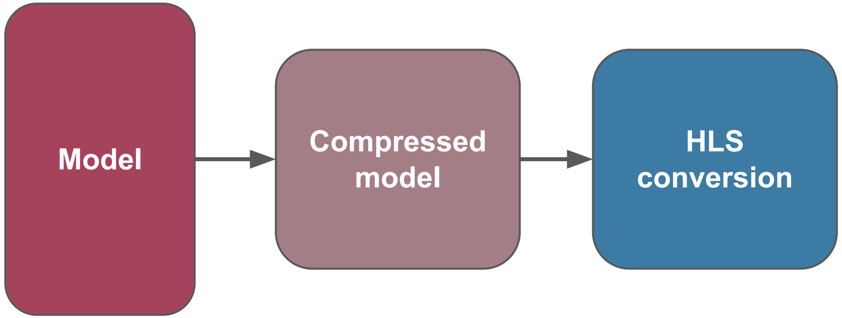
ONNX

K + TensorFlow

Integration with a hardware synthesis tool for ML



Integration with a hardware synthesis tool for ML

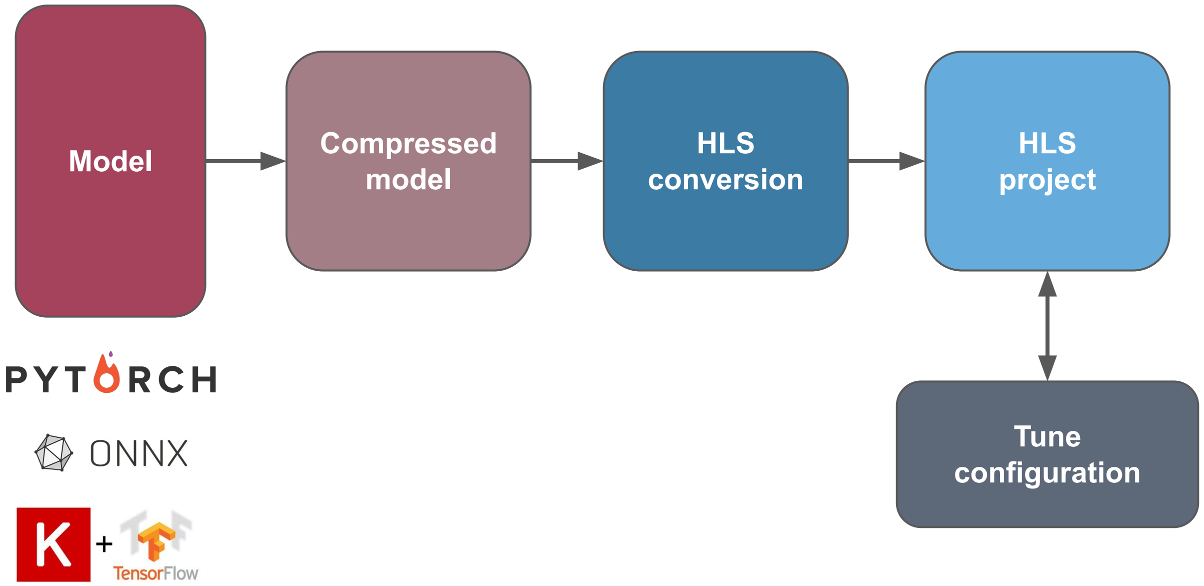


PYTORCH

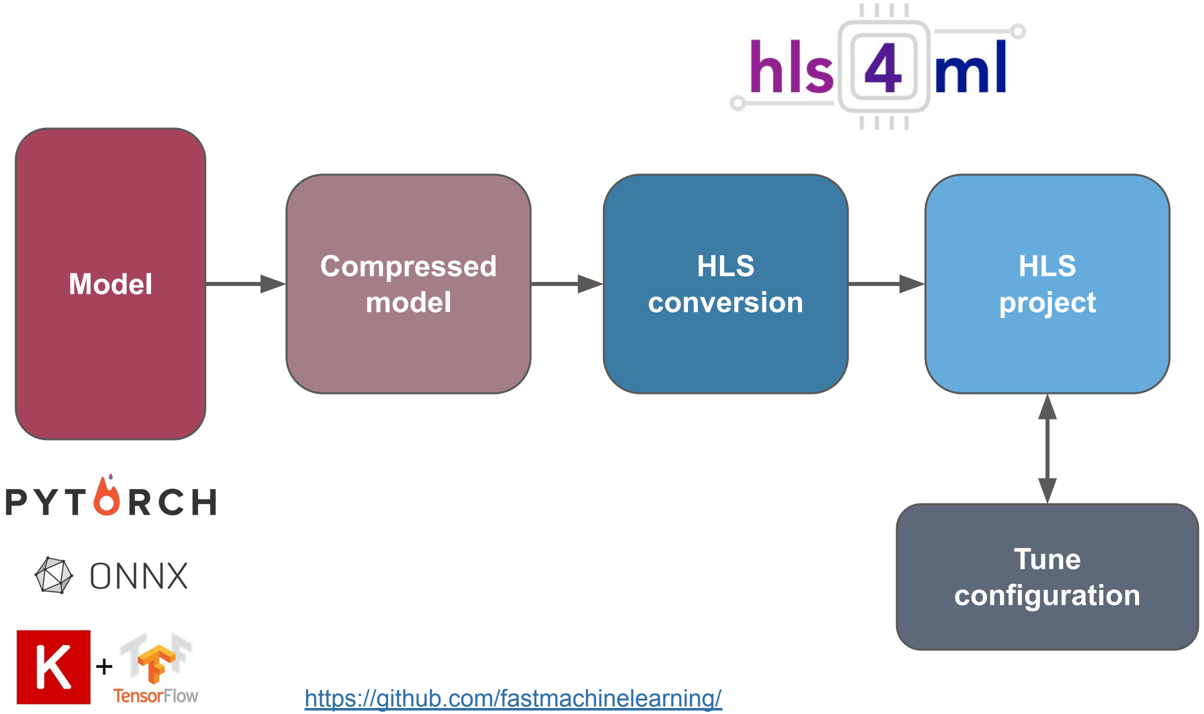
ONNX

K + TensorFlow

Integration with a hardware synthesis tool for ML



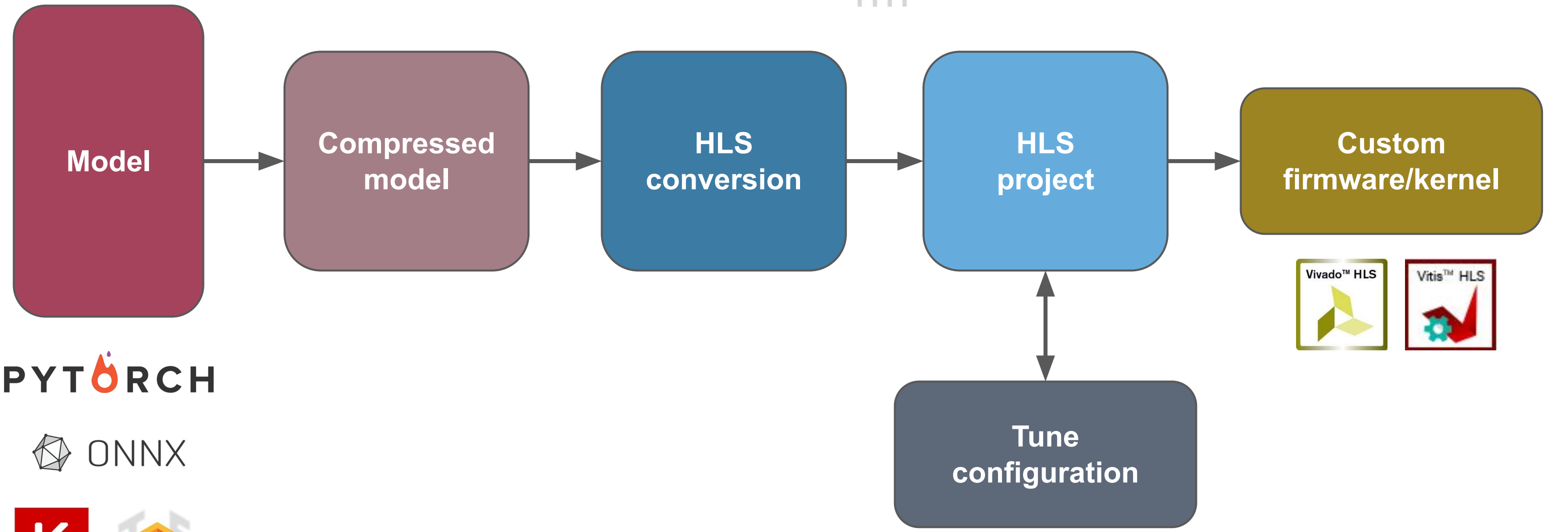
Integration with a hardware synthesis tool for ML



https://github.com/fastmachinelearning/

Integration with a hardware synthesis tool for ML

hls4ml



PYTORCH

ONNX

K + TensorFlow

https://github.com/fastmachinelearning/

Integration with a hardware synthesis tool for ML

hls4ml

ML framework support:

- (Q)Keras
- PyTorch (limited)
- (Q)ONNX (in development)

https://fastmachinelearning.org/hls4ml/

Integration with a hardware synthesis tool for ML

hls4ml

ML framework support:

- (Q)Keras
- PyTorch (limited)
- (Q)ONNX (in development)

Neural networks architectures:

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

https://fastmachinelearning.org/hls4ml/

Integration with a hardware synthesis tool for ML

hls4ml

ML framework support:

- (Q)Keras
- PyTorch (limited)
- (Q)ONNX (in development)

Neural networks architectures:

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

HLS backends:

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

https://fastmachinelearning.org/hls4ml/

Integration with a hardware synthesis tool for ML

Python integration

hls4ml

```python
from tensorflow.keras.models import load_model
from sklearn.metrics import accuracy_score
model = load_model('model_keras_MLP.h5')
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
fc1 (Dense)                  (None, 60)                3900
_________________________________________________________________
relu1 (Activation)           (None, 60)                0
_________________________________________________________________
fc0 (Dense)                  (None, 40)                2440
_________________________________________________________________
relu0 (Activation)           (None, 40)                0
_________________________________________________________________
fc2 (Dense)                  (None, 30)                1230
_________________________________________________________________
relu2 (Activation)           (None, 30)                0
_________________________________________________________________
```

Integration with a hardware synthesis tool for ML

Python integration

hls4ml

```
import hls4ml
import plotting

config = hls4ml.utils.config_from_keras_model(teacherMLP, granularity='model')
config['Model'] = {'Precision' : 'ap_fixed<24,16>', 'ReuseFactor' : 1, 'Strategy' : 'Latency'}
print("-----------------------------------")
plotting.print_dict(config)
print("-----------------------------------")
hls_model = hls4ml.converters.convert_from_keras_model(teacherMLP,
                                                       hls_config=config,
                                                       output_dir='model_3/hls_model'
                                                      )
hls_model.compile()
```

Integration with a hardware synthesis tool for ML

Python integration

hls4ml

```
import hls4ml
import plotting
hls4ml.model.optimizer.OutputRoundingSaturationMode.layers = ['Activation']
hls4ml.model.optimizer.OutputRoundingSaturationMode.rounding_mode = 'AP_RND'
hls4ml.model.optimizer.OutputRoundingSaturationMode.saturation_mode = 'AP_SAT'

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

config['Model'] = {'Precision' : 'ap_fixed<17,16>', 'ReuseFactor' : 1, 'Strategy' : 'Latency'}

config['LayerName']['fc1']['Precision']['weight'] = 'ap_fixed<9, 1>'

config['LayerName']['softmax']['Precision'] = 'ap_fixed<32,15>'

print("-----------------------------------")
plotting.print_dict(config)
print("-----------------------------------")
hls_model = hls4ml.converters.convert_from_keras_model(model,
                                                       hls_config=config,
                                                       output_dir='model_3/MLP_student_smr3765'
                                                      )
hls_model.compile()
```

Integration with a hardware synthesis tool for ML

QKeras for quantization-aware training

hls4ml

```
# MLP architecture
# Create the student QKERAS
studentQ_MLP = keras.Sequential(
    [
        Input(shape=(30,)),
        QDense(20, name='fc1',
                  kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu1'),
        QDense(10, name='fc2',
                  kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu2'),
        QDense(10, name='fc6',
                  kernel_quantizer=quantized_bits(9,1,alpha=1), bias_quantizer=quantized_bits(23,15,alpha=1)),
        QActivation(activation=quantized_relu(16,15), name='relu3'),

        QDense(4, name='output',
                  kernel_quantizer=quantized_bits(32,15,alpha=1), bias_quantizer=quantized_bits(32,15,alpha=1)),
        Activation(activation='softmax', name='softmax')


    ],
    name="student",
)

print_qstats(studentQ_MLP)
```

Integration with a hardware synthesis tool for ML

High-level synthesis project generated through hls4ml

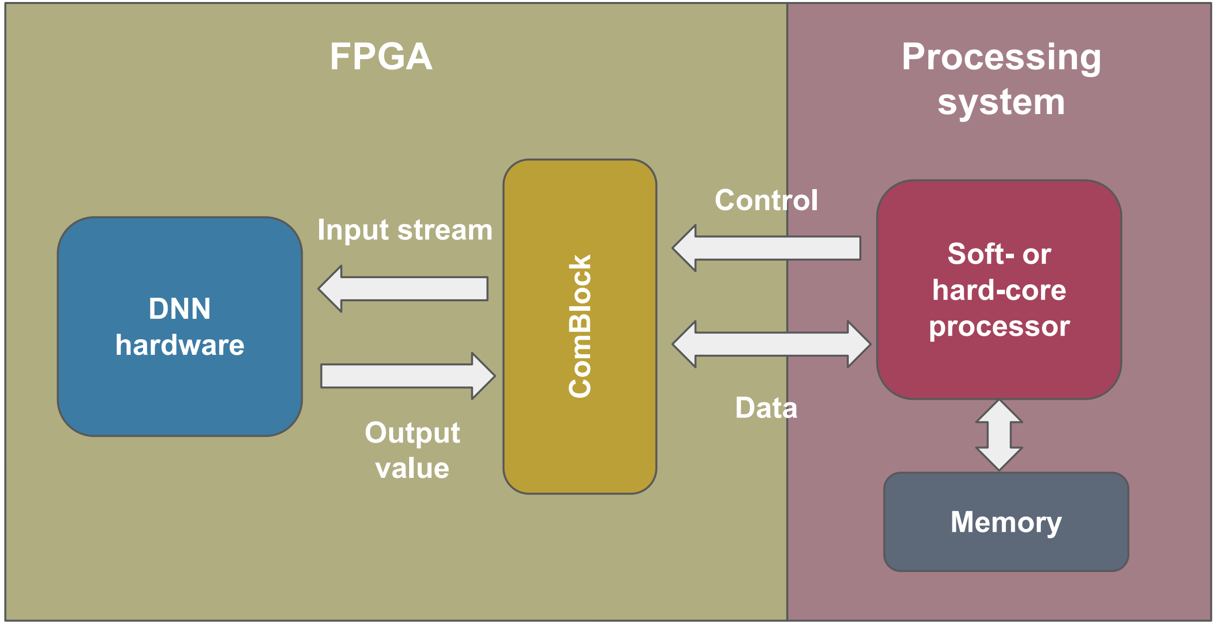
hls4ml

esis Summary(solution1) | myproject.cpp

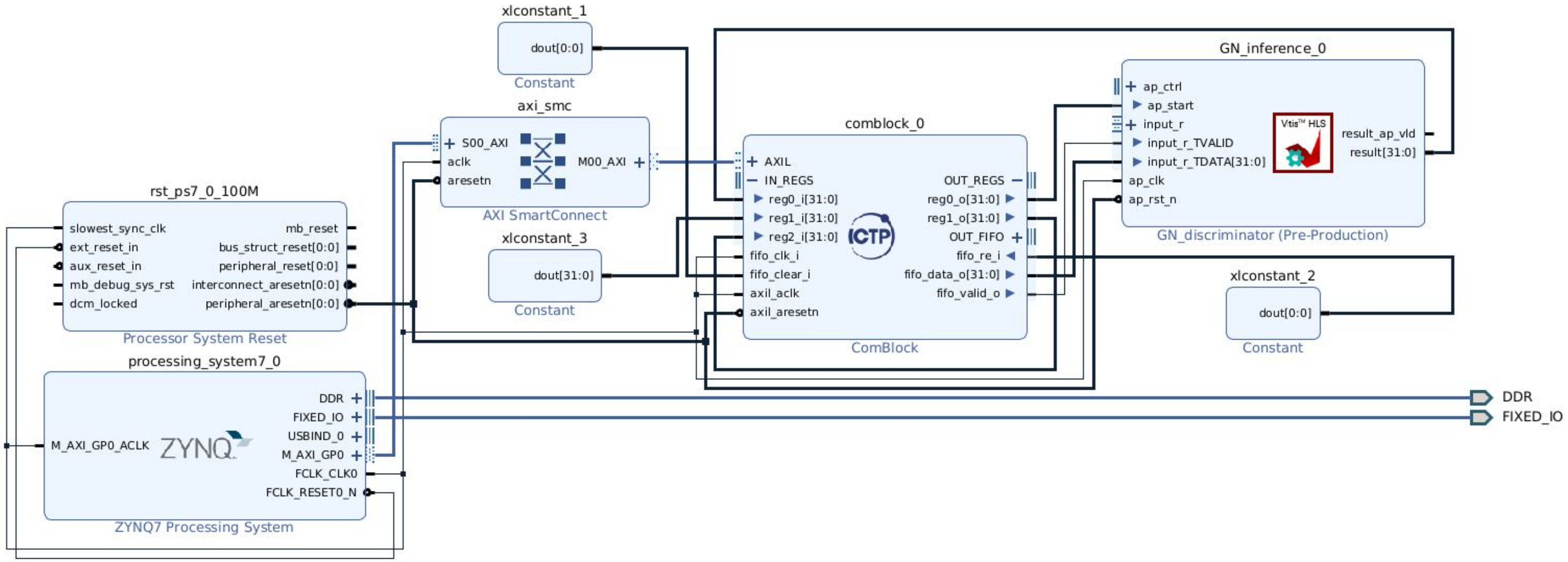
```
layer8_t layer8_out[N_LAYER_8];
#pragma HLS ARRAY_PARTITION variable=layer8_out complete dim=0
nnet::dense<layer6_t, layer8_t, config8>(layer6_out, layer8_out, w8, b8); // fc3

layer9_t layer9_out[N_LAYER_8];
#pragma HLS ARRAY_PARTITION variable=layer9_out complete dim=0
nnet::linear<layer8_t, layer9_t, linear_config9>(layer8_out, layer9_out); // fc3_linear

layer11_t layer11_out[N_LAYER_11];
#pragma HLS ARRAY_PARTITION variable=layer11_out complete dim=0
nnet::dense<layer9_t, layer11_t, config11>(layer9_out, layer11_out, w11, b11); // fc4

layer12_t layer12_out[N_LAYER_11];
#pragma HLS ARRAY_PARTITION variable=layer12_out complete dim=0
nnet::linear<layer11_t, layer12_t, linear_config12>(layer11_out, layer12_out); // fc4_linear
```

C. Hardware assessment framework

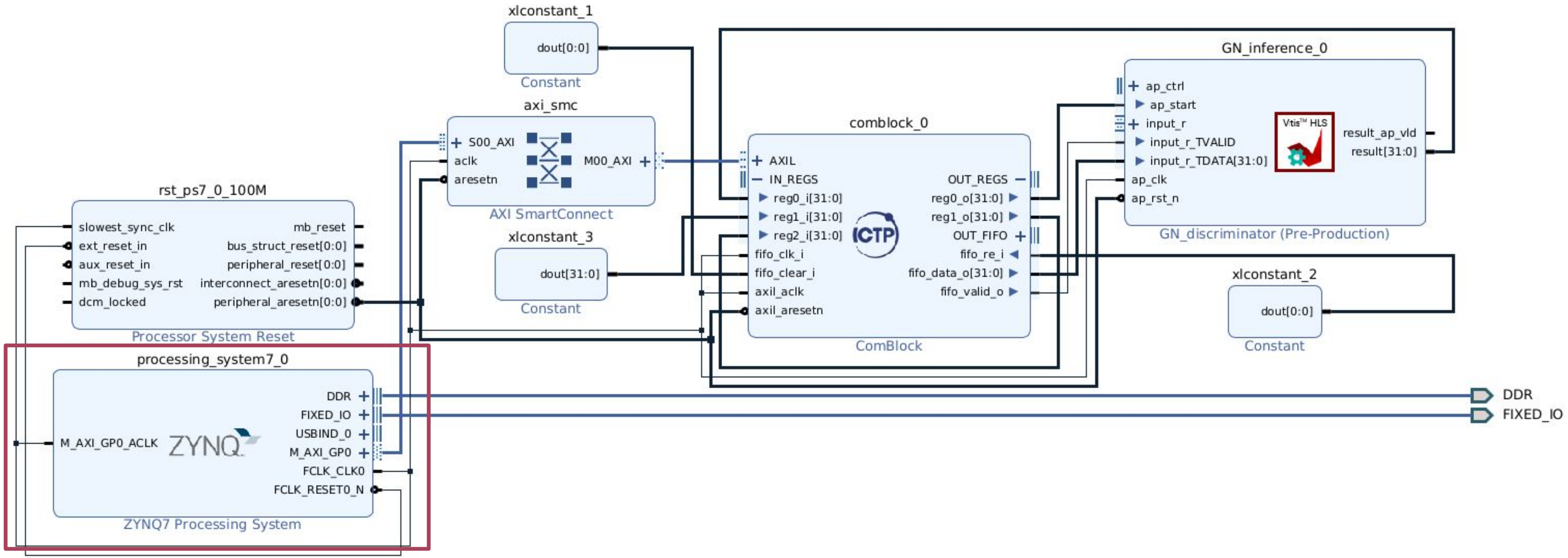
Hardware assessment framework



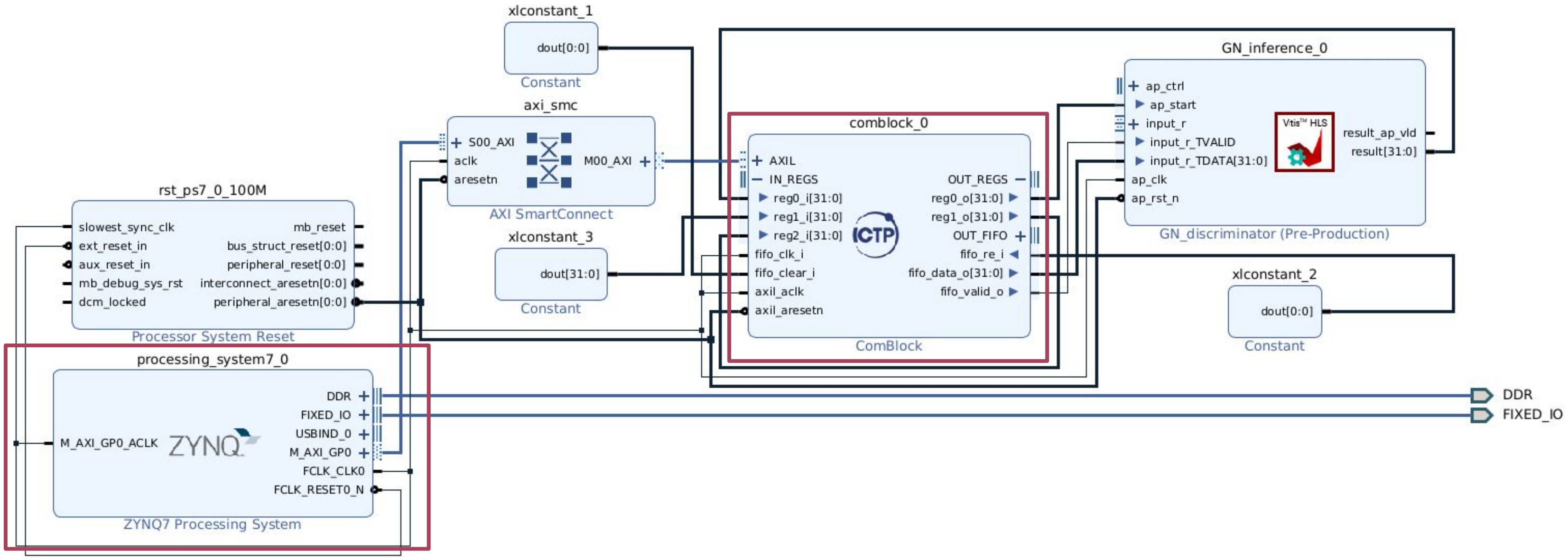
Hardware assessment framework



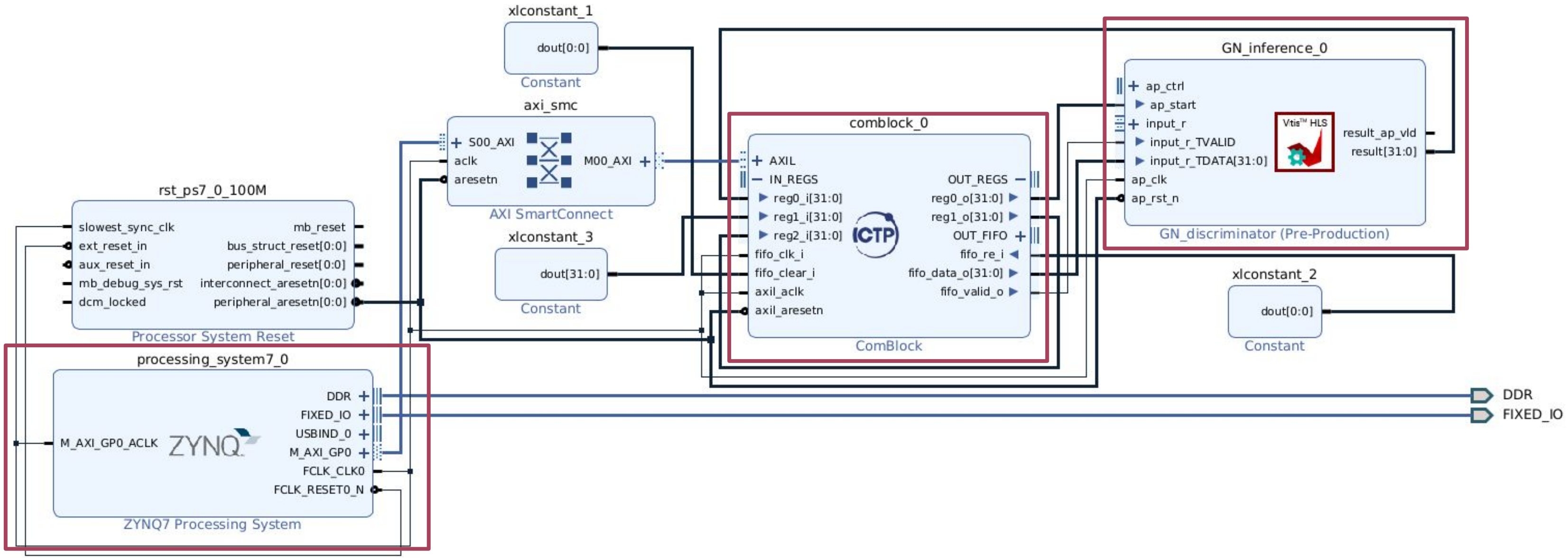
Hardware assessment framework



Hardware assessment framework



Hardware assessment framework



Applications

ML and model compression techniques for SoC/FPGA

Applications

- Gamma/Neutron discrimination [submitted TNS].
- Pest classification in fruit crops [9, 11].
- Pulse shape discriminator for cosmic rays studies [8, 11].
- Volcanic seismic event detection [12].
- Object detection for adverse weather conditions, particularly haze and fog [India - on-going].
- Water quality monitoring applied to Dunav river [Serbia - Remarkable / on-going].

Gamma/Neutron discrimination

ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

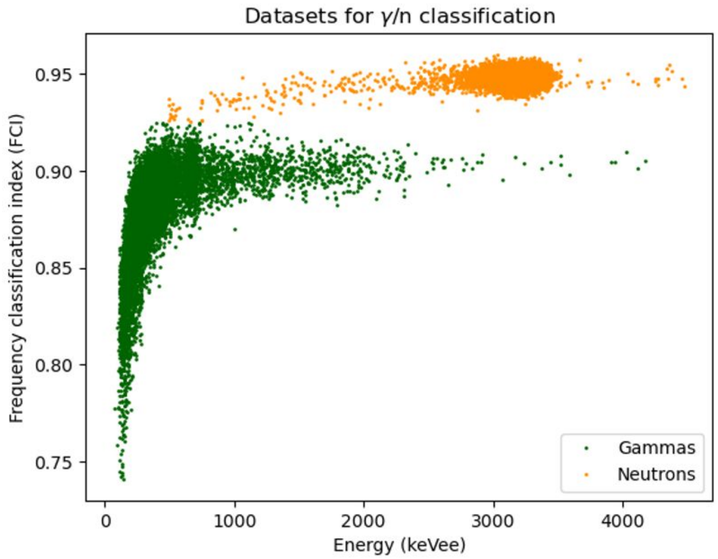
- Tagged dataset of **gamma and neutron events** from Deuterium-Deuterium (DD) and Deuterium-Tritium (DT) generators.
- The dataset was recorded at the **Neutron Science Facility (NSF) of the Nuclear Science and Instrumentation Laboratory (NSIL), IAEA.**
- The detector is based on a small **CLYC** (Cs2LiYCl6:Ce) crystal (0.5 in diameter by 30 mm length) coupled to a 4-element SiPM array.
- The data were **sampled at 4 GSPS with 10-bits resolution** using a CAEN DT5761 digitizer.
- **The total gamma and neutron events in this dataset are 10,913 and 27,696, respectively.**

ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency



Morales, I. R., Crespo, M. L., Bogovac, M., Cicuttin, A., Kanaki, K., & Carrato, S. (2023). Gamma/neutron classification with SiPM CLYC detectors using frequency-domain analysis for embedded real-time applications. Nuclear Engineering and Technology.

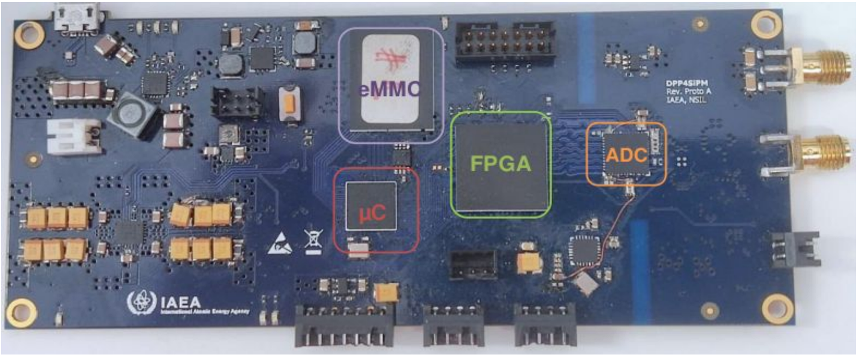
Dataset from https://doi.org/10.5281/zenodo.8037059

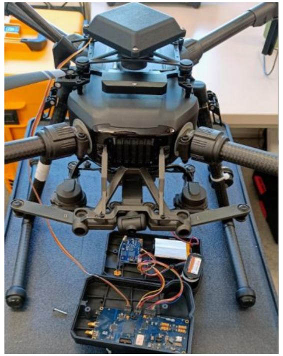
ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency



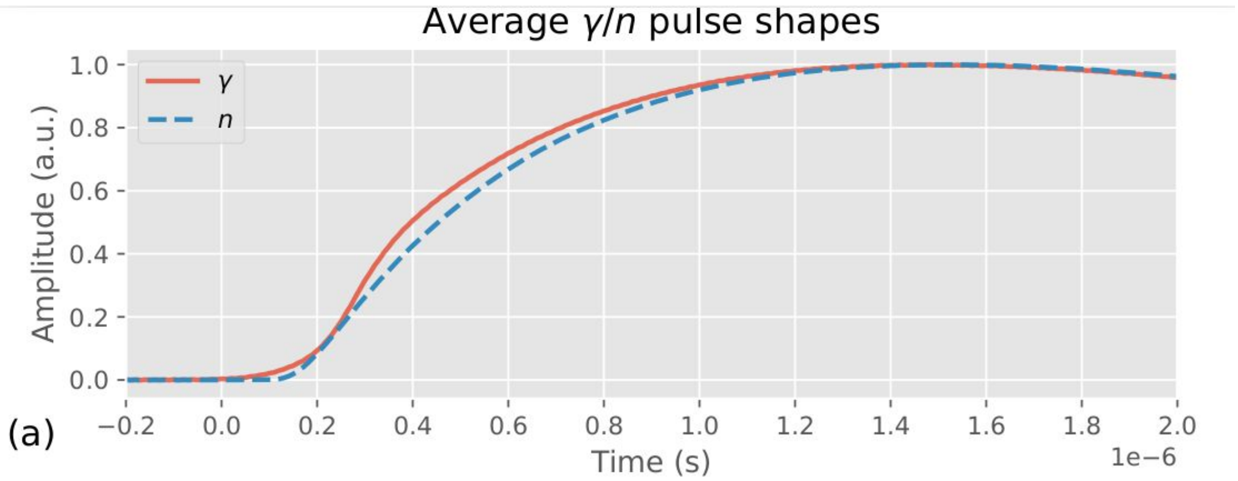


ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

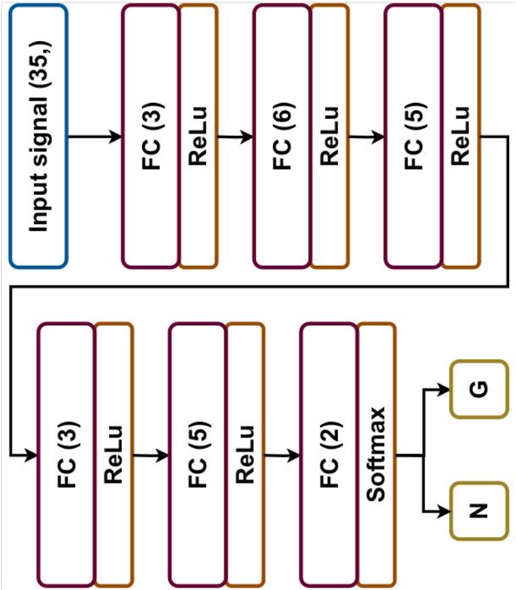


ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

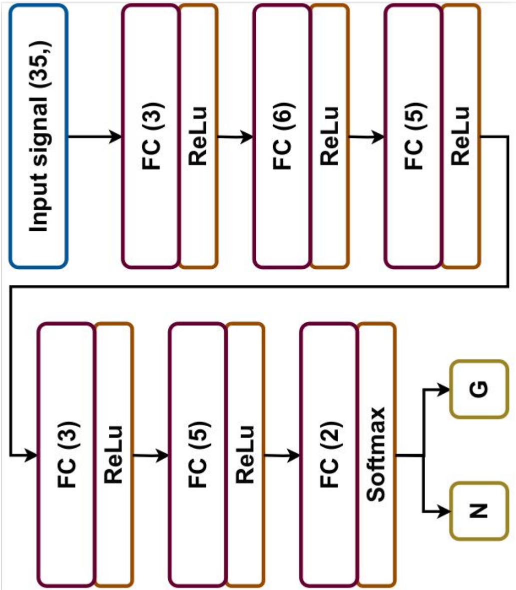


ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency



Teacher architecture with **2,623 parameters** distributed in 6 hidden layers (MLP).

Compressed architecture with **217 parameters**, distributed in 6 hidden layers (MLP).

Input size reduction:
35 samples of the leading edge of the pulse.

ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

- **Overall accuracy**
  - Teacher architecture (original): **99.00%**
  - Student architecture (compressed): **98.20%**

ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

- **Overall accuracy**
  - Teacher architecture (original): **99.00%**
  - Student architecture (compressed): **98.20%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - Artix-7 platform: **below 35%**

ML and model compression techniques for SoC/FPGA

Applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

- **Overall accuracy**
  - Teacher architecture (original): **99.00%**
  - Student architecture (compressed): **98.20%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - Artix-7 platform: **below 35%**

- **SoC latency**
  - Zedboard platform: **45 clk cycles (@200MHz)**

Image classification based on CNN

ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

Precision agriculture on the edge

ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

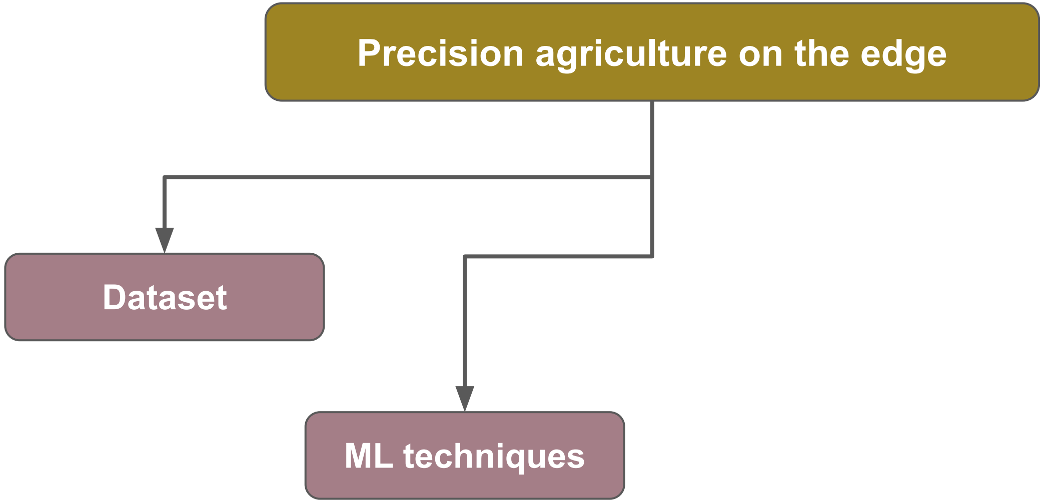
Precision agriculture on the edge

Dataset

ML and model compression techniques for SoC/FPGA

Applications

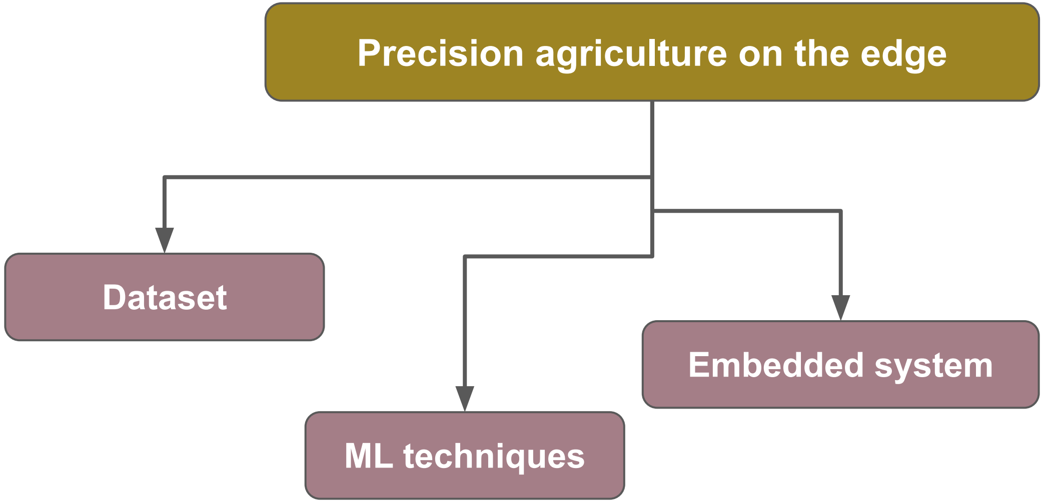
Pest classification in fruit crops



ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops



ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

Precision agriculture on the edge

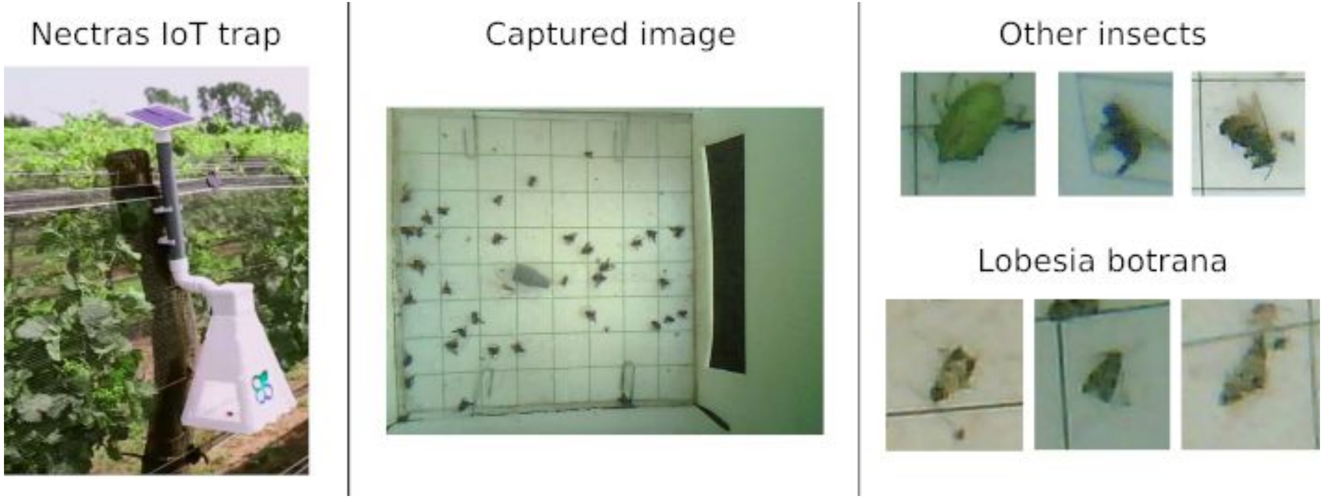
- Dataset
- ML techniques
- Embedded system
- Memory footprint

ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

Precision agriculture on the edge

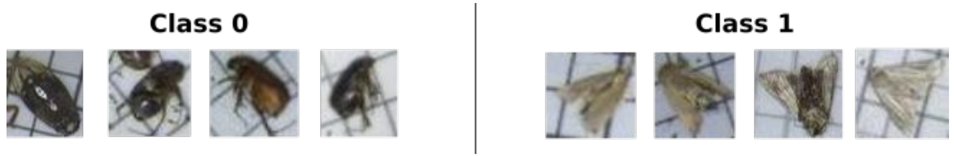


ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

Pest24 [6]



A standard dataset available in the literature for training, granting a stable and effective performance.

ML and model compression techniques for SoC/FPGA

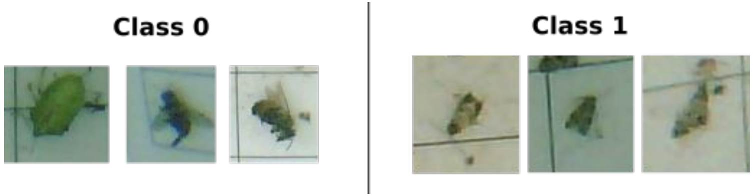
Applications

Pest classification in fruit crops

Pest24 [6]

A standard dataset available in the literature for training, granting a stable and effective performance.

Argentina

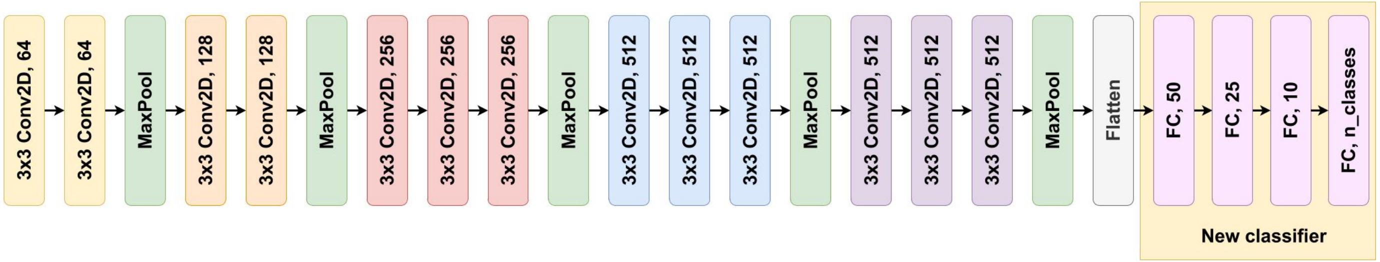


Images provided by the system in Argentina.

ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

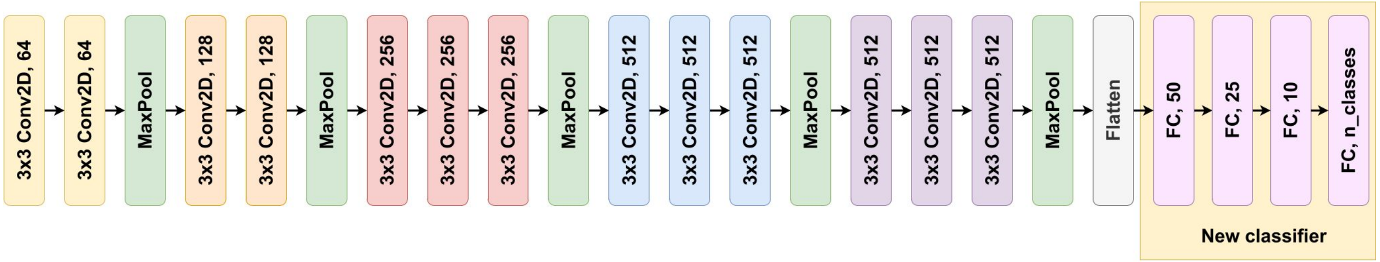


Teacher architecture based on VGG16 and obtained through transfer learning
– 14,818,706 parameters –

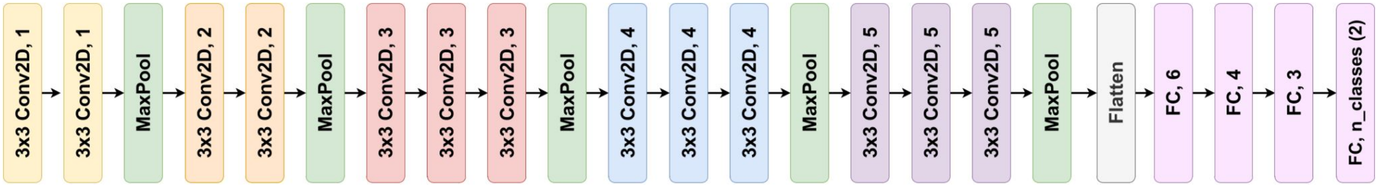
ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops



Teacher architecture based on VGG16 and obtained through transfer learning
– 14,818,706 parameters –

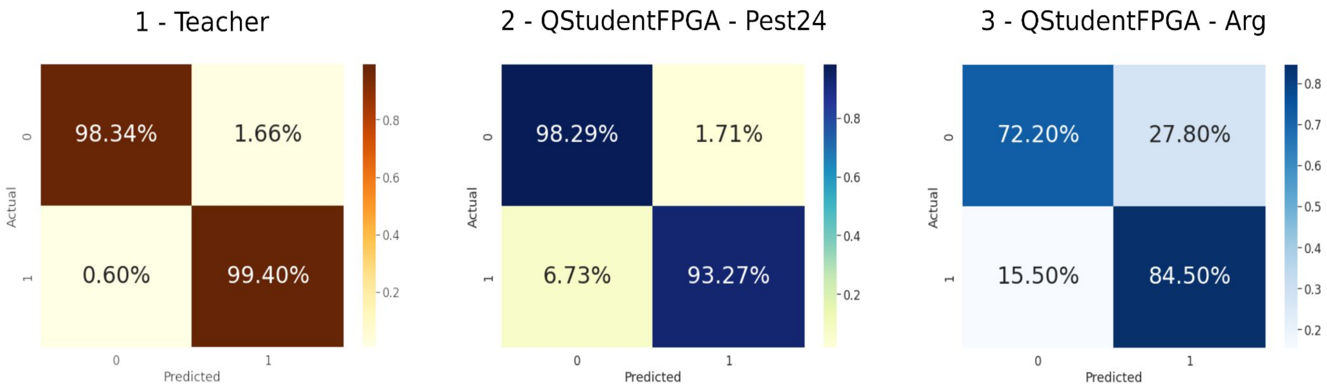


Compressed architecture.
Compression ratio: 7,409*x* – in number of parameters –

ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops



ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

- **Overall accuracy**
  - Teacher architecture: **98.87%**
  - Student architecture: **95.78%**

ML and model compression techniques for SoC/FPGA

Applications

Pest classification in fruit crops

- **Overall accuracy**
  - Teacher architecture: **98.87%**
  - Student architecture: **95.78%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - KRIA platform: **below 21%**
  - PYNQ-Z1 platform: **below 63%**

ML and model compression techniques for SoC/FPGA

Applications

Object detection
[work in progress]

Image from Cityscapes dataset.
Cordts, M., Omran, M., Ramos, S., Rehfeld, T., Enzweiler, M., Benenson, R., ... & Schiele, B. (2016). The cityscapes dataset for semantic urban scene understanding. In *Proceedings of the IEEE conference on computer vision and pattern recognition* (pp. 3213-3223).

ML and model compression techniques for SoC/FPGA

Applications

Object detection
[work in progress]

Image from Cityscapes dataset.

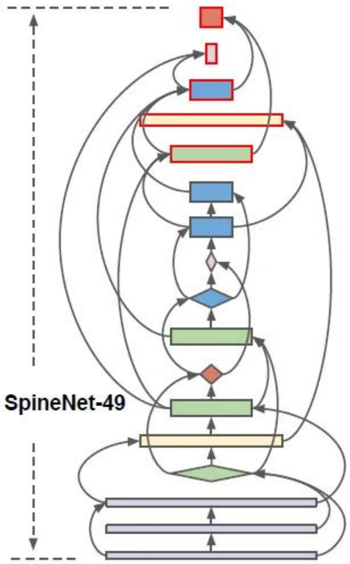


Image from Du, X., Lin, T. Y., Jin, P., Ghiasi, G., Tan, M., Cui, Y., ... & Song, X. (2020). Spinenet: Learning scale-permuted backbone for recognition and localization. In Proceedings of the IEEE/CVF conference on computer vision and pattern recognition (pp. 11592-11601).

Final remarks

- The proposed workflow successfully generates compressed models, leading to a **fully on-chip memory-mapped implementation** on the FPGA.

- The **integration of KD** into the ensemble of compression techniques contributes to achieving a balanced student model in terms of size, computational efficiency, and accuracy.

- The workflow addresses the entire development cycle: **from the ML-based architecture training to the hardware deployment**, overcoming the limitations outlined in previous works.

- Addition of FPGA metric estimation in the training stage to adapt the ML-based model to the hardware architecture.

The Abdus Salam International Centre for Theoretical Physics

Workshop on Fully Programmable Systems-on-Chip for Scientific Applications

IAEA International Atomic Energy Agency Atoms for Peace and Development

IUB Independent University, Bangladesh

جامعة قطر QATAR UNIVERSITY

Universiti Kebangsaan Malaysia The National University of Malaysia

Thank you!!

Doha, Qatar
2024

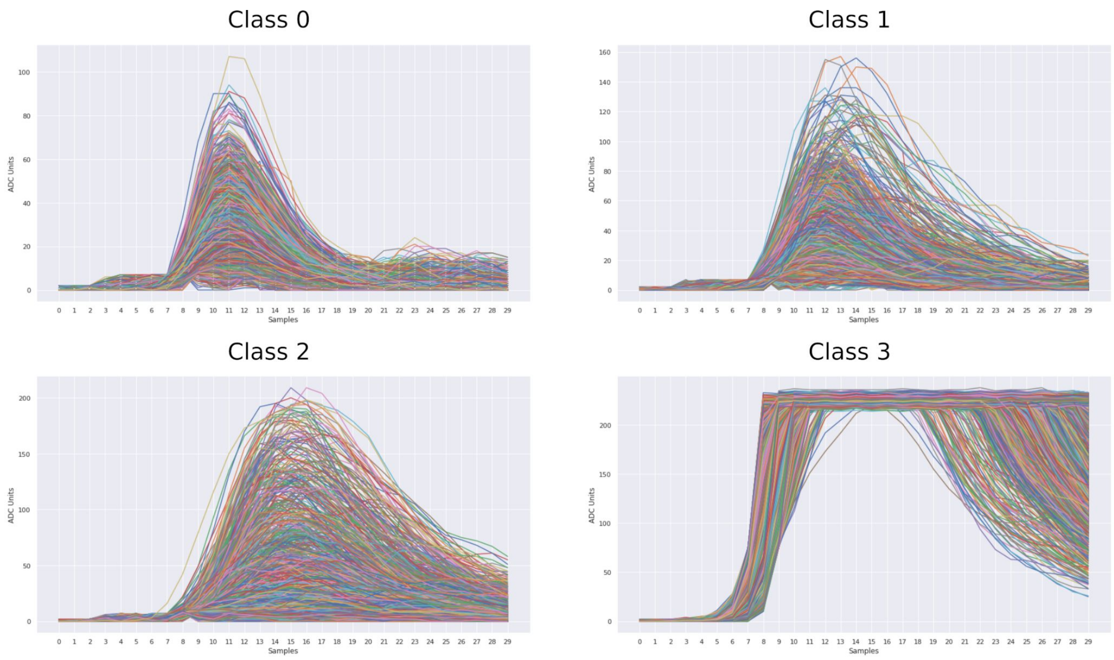
Back up slides

Pulse shape discriminator for cosmic rays studies

ML and model compression techniques for SoC/FPGA

Applications

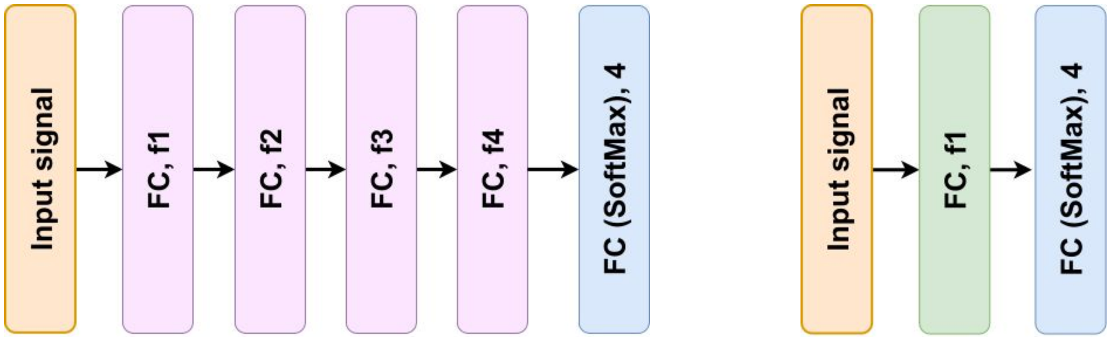
Pulse shape discriminator for cosmic rays studies



ML and model compression techniques for SoC/FPGA

Applications

Pulse shape discriminator for cosmic rays studies



Left. Teacher architecture based on MLP - **16,352** parameters.

Right: Distilled architecture - **529** parameters
Compression ratio: 30.91*x.*

ML and model compression techniques for SoC/FPGA

Applications

Pulse shape discriminator for cosmic rays studies

- **Overall accuracy**
  - Teacher architecture: **99.70%**
  - Student architecture: **98.96%**
    - **8-bit fixed point**
    - **Target sparsity: 20%**

ML and model compression techniques for SoC/FPGA

Applications

Pulse shape discriminator for cosmic rays studies

- **Overall accuracy**
  - Teacher architecture: **99.70%**
  - Student architecture: **98.96%**
    - **8-bit fixed point**
    - **Target sparsity: 20%**

- **SoC memory footprint in terms of resource utilization @200MHz**
  - Artix-7: **below 27%**

ML and model compression techniques for SoC/FPGA

Applications

Pulse shape discriminator for cosmic rays studies

- **Overall accuracy**
  - Teacher architecture: **99.70%**
  - Student architecture: **98.96%**
    - **8-bit fixed point**
    - **Target sparsity: 20%**

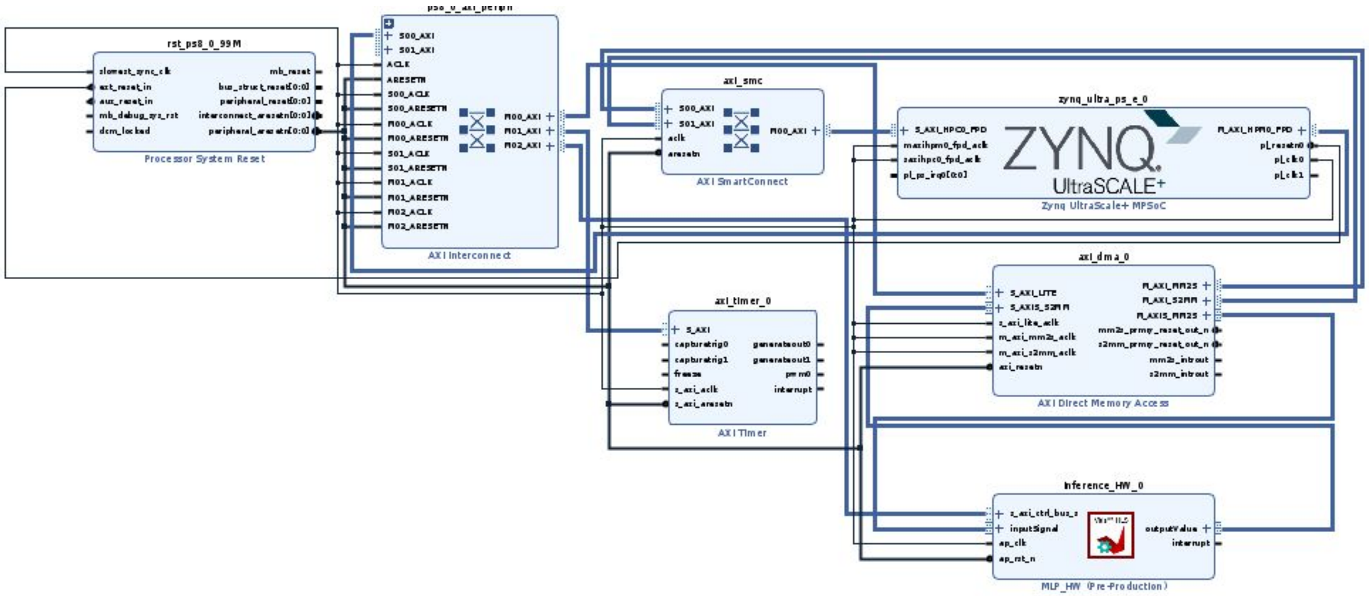
- **SoC memory footprint in terms of resource utilization @200MHz**
  - Artix-7: **below 27%**

- **SoC latency @200MHz**
  - Artix-7: **10 clock cycles**

ML and model compression techniques for SoC/FPGA

Applications - Connections example

Pulse shape discriminator for cosmic rays studies



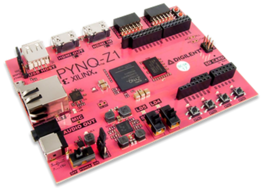
ML and model compression techniques for SoC/FPGA

Applications

Pulse shape discriminator for cosmic rays studies

Pulse shape discriminator for cosmic rays

```
In [ ]: from pynq import Overlay

In [ ]: ol = Overlay("hw/inference_PYNQ.bit")

In [ ]: ol.ip_dict

In [ ]: dma = ol.axi_dma_0
        dma_send = ol.axi_dma_0.sendchannel
        dma_recv = ol.axi_dma_0.recvchannel

In [ ]: from pynq import allocate
        import numpy as np

        data_size = 30
        input_buffer = allocate(shape=(data_size,), dtype=np.uint32)

In [ ]: x3 = [0, 2, 0, 0, 0, 0, 2, 14, 68, 231, 232, 232, 232, 230, 232,
              231, 233, 232, 231, 231, 232, 232, 231, 230, 232, 231, 232, 231, 231, 230]
        for i in range(0, data_size):
            input_buffer[i] = x3[i]

In [ ]: import matplotlib.pyplot as plt
        plt.figure(figsize=(15,7))
        plt.xlabel('Samples', fontsize=11)
        plt.ylabel('Amplitude', fontsize=11)
        plt.grid(True, alpha=1.0)
        plt.plot(x3, 'o', label="Signal 1", color='navy', markersize=7, lw=1)
```

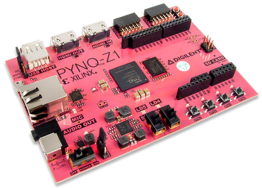


ML and model compression techniques for SoC/FPGA

Applications

Pulse shape discriminator for cosmic rays studies

```
In [ ]: hls_ip = ol.inference_HW_0

In [ ]: hls_ip.register_map

In [ ]: # Initialize HLS IP core

        CONTROL_REGISTER = 0x0
        hls_ip.write(CONTROL_REGISTER, 0x81) # 0x81 will set bit 0

In [ ]: hls_ip.register_map

In [ ]: # Start the DMA transfer
        dma_send.transfer(input_buffer)

In [ ]: output_buffer = allocate(shape=(4,), dtype=np.uint32)

In [ ]: dma_recv.transfer(output_buffer)

In [ ]: for i in range(4):
            print((output_buffer[i]))
```



PYNQ