1st Mesoamerican Workshop on Reconfigurable X-ray Scientific Instrumentation for Cultural Heritage

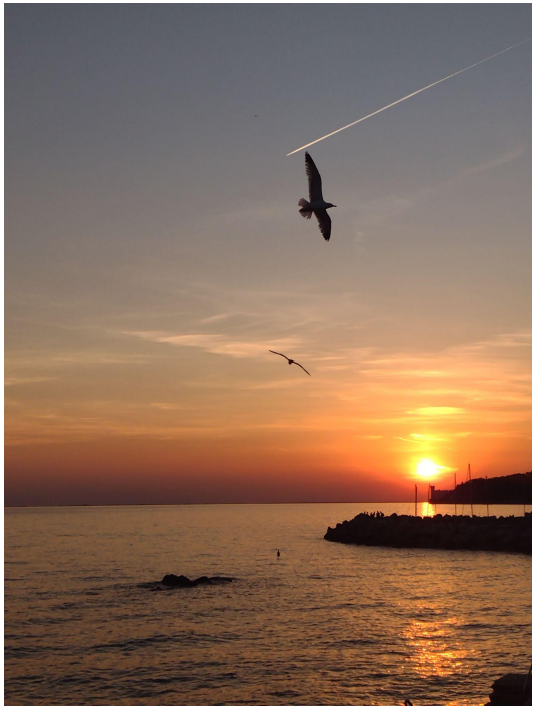
Machine learning and model compression for FPGA/SoC

Antigua Guatemala, June 2025

Romina Soledad Molina, Ph.D.

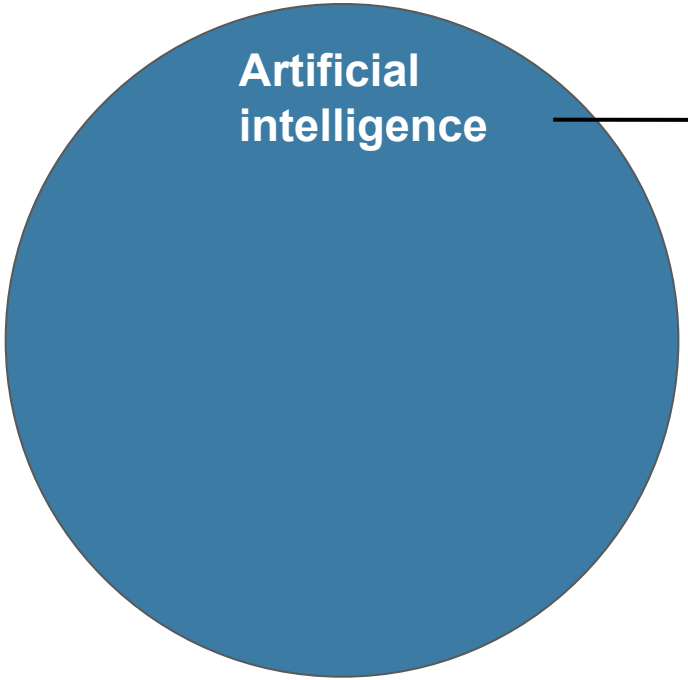
Outline

- Introduction.
- Machine Learning.
- Machine Learning and System-On-Chips based on FPGA.
- 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.
- ML and SoC-based FPGA for real-case applications.
- Final remarks.



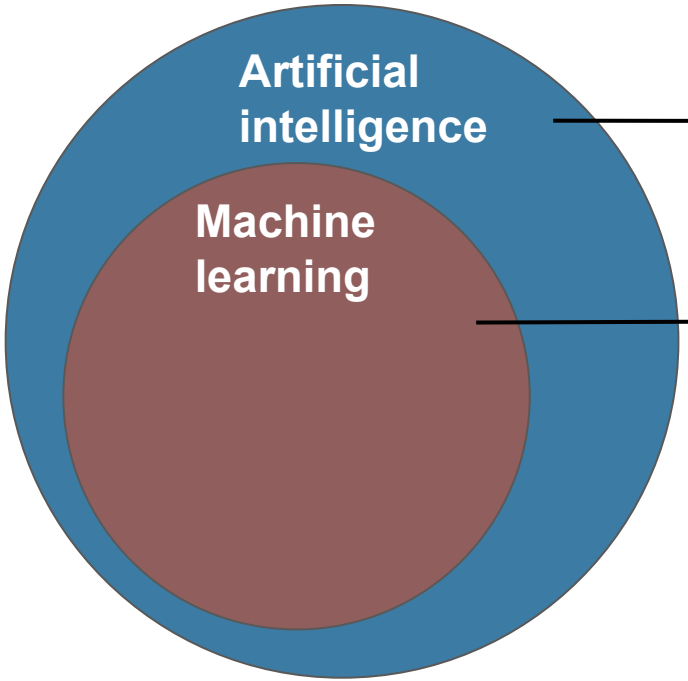
Introduction

Introduction



A machine/system capable of imitating human behavior

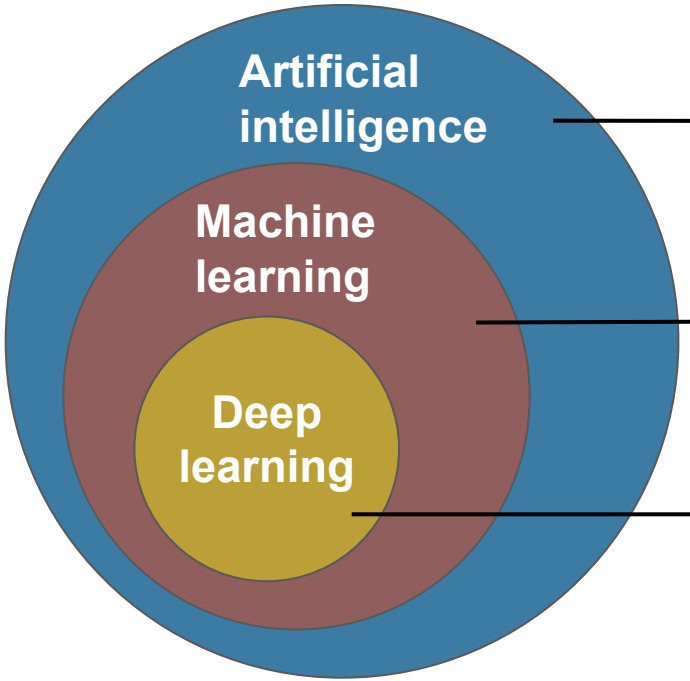
Introduction



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

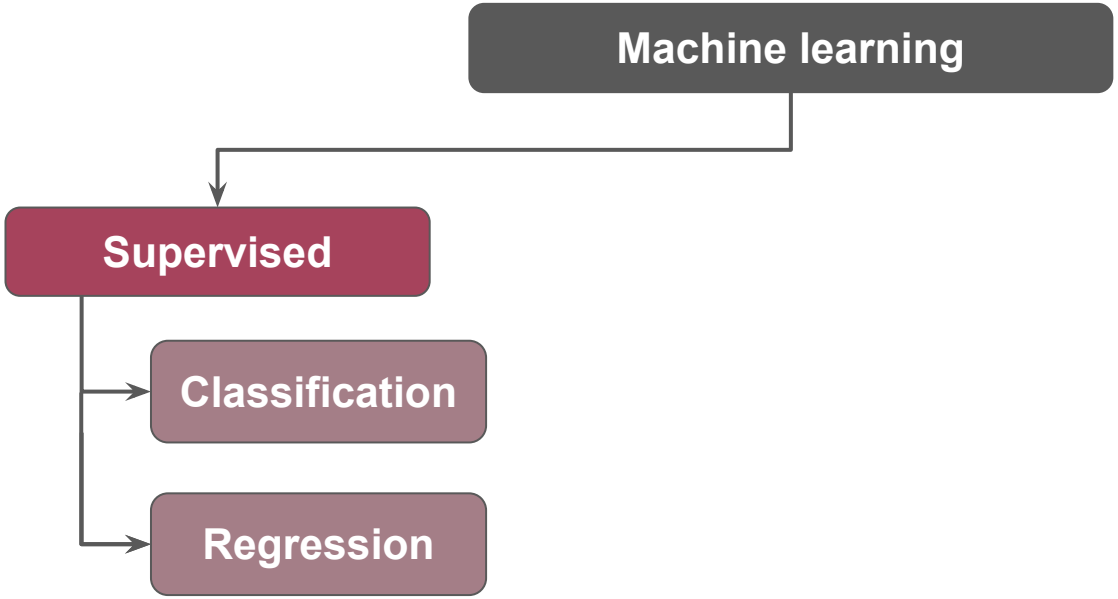
Machine learning

Classification

Machine learning

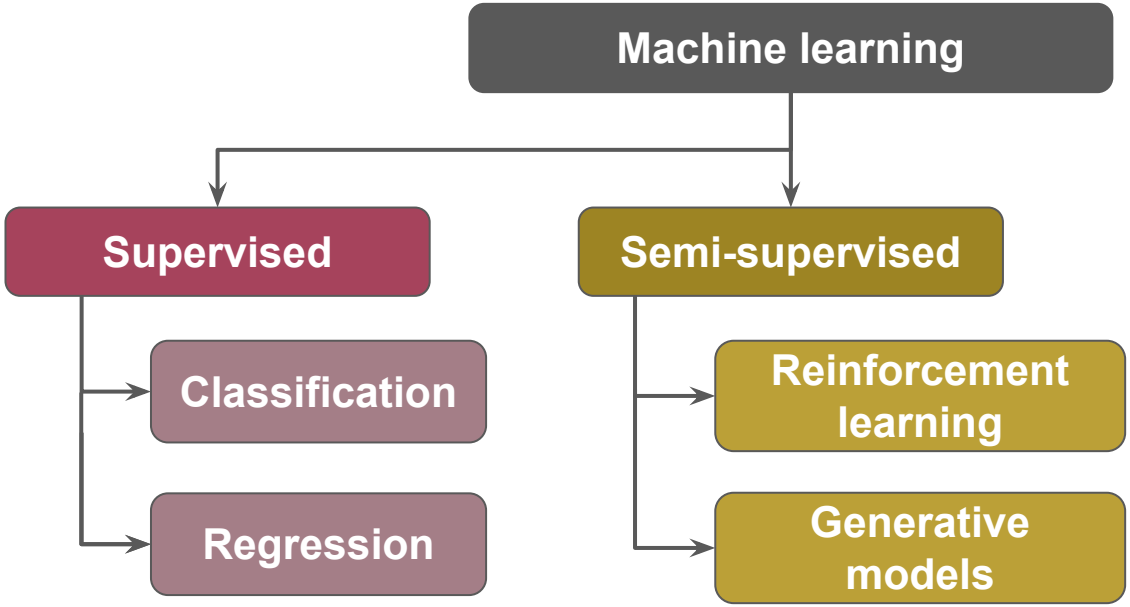
Machine learning

Classification



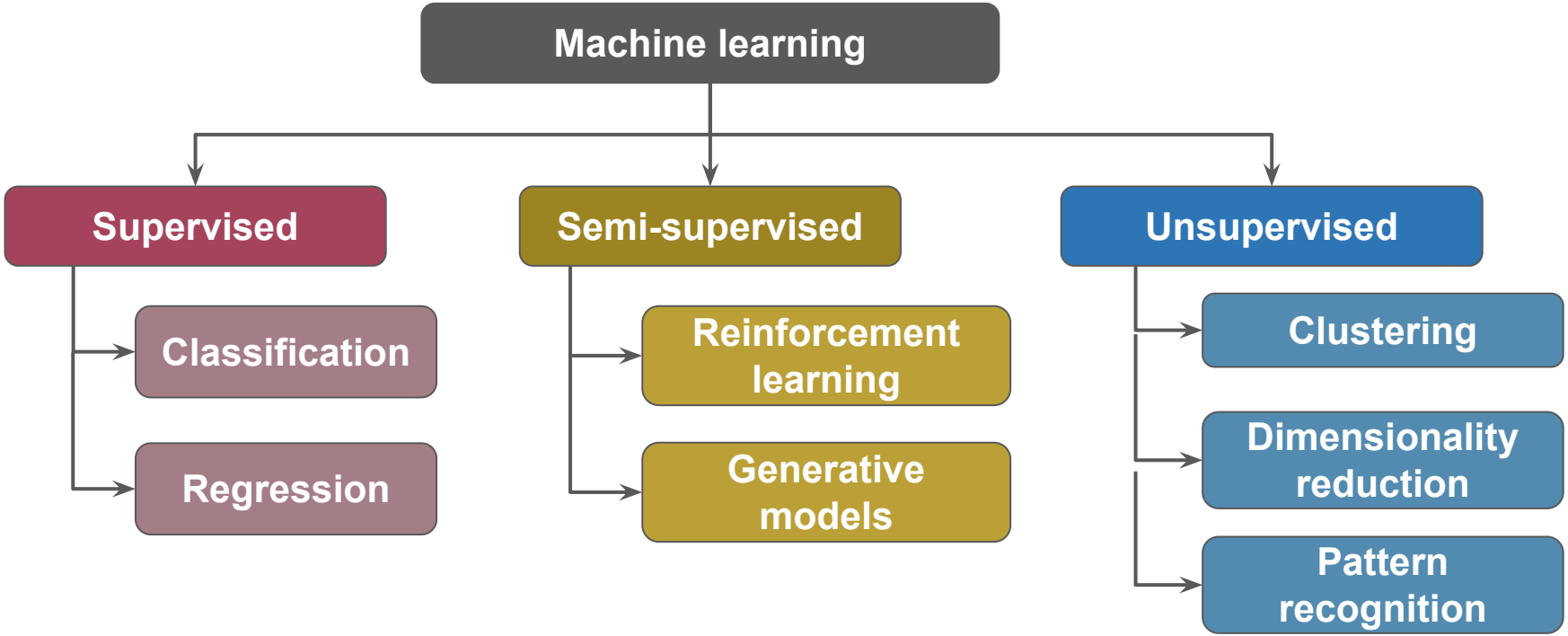
Machine learning

Classification

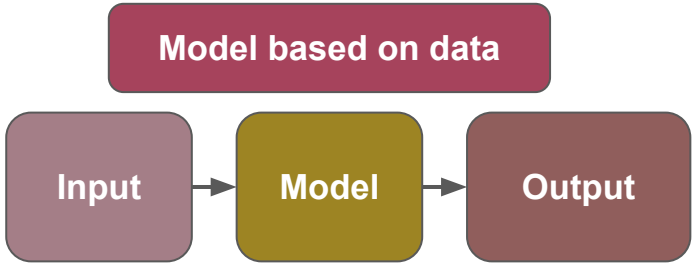


Machine learning

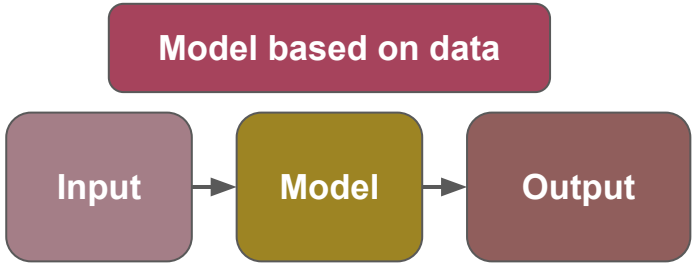
Classification

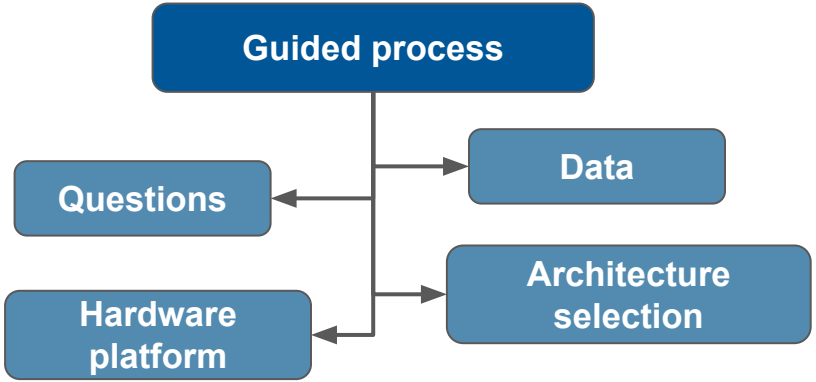


Machine learning

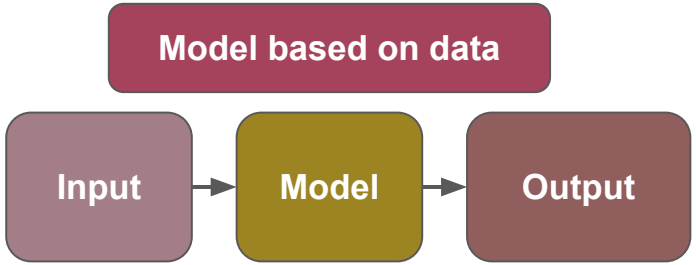


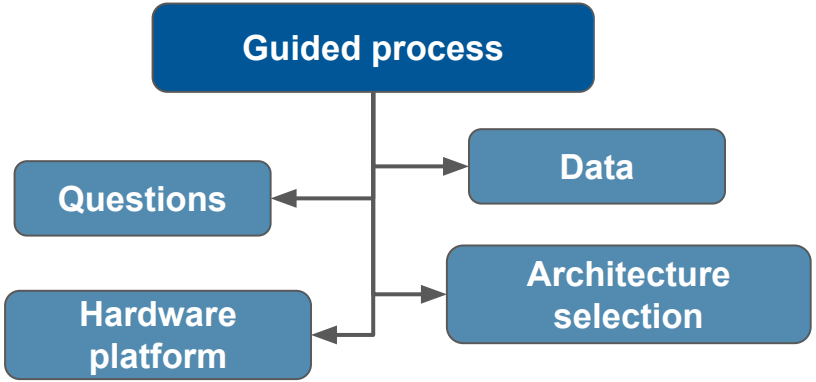
Machine learning

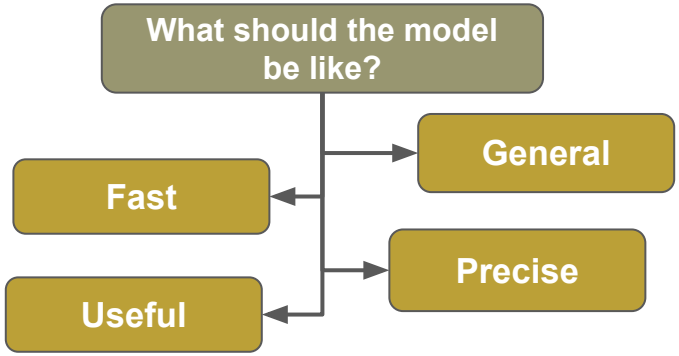




Machine learning





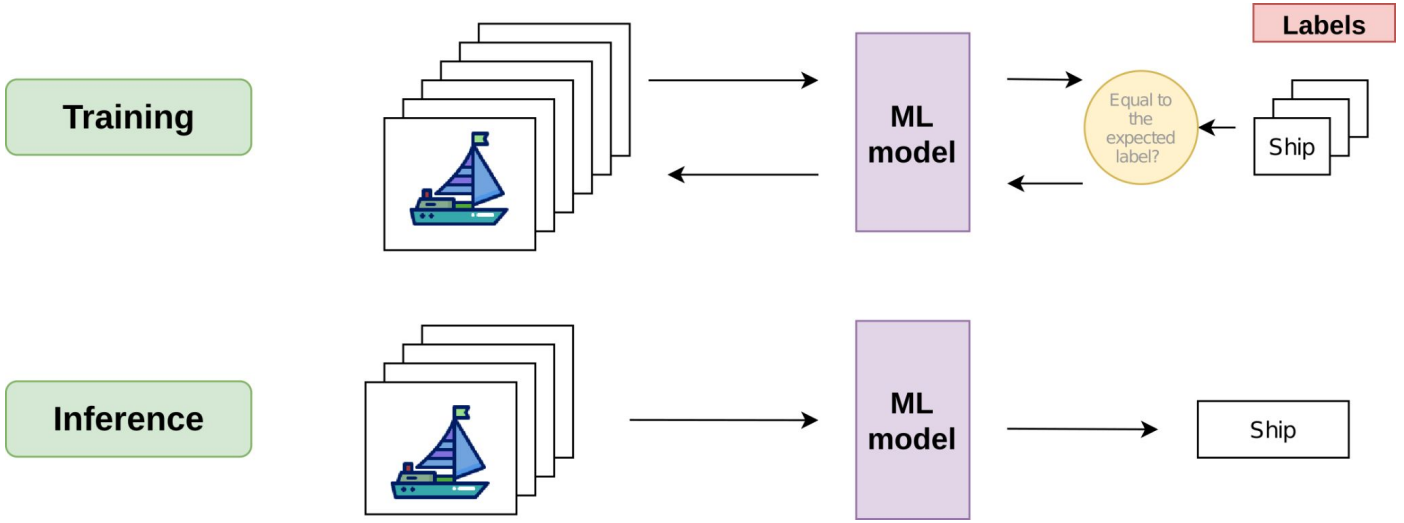


Machine learning

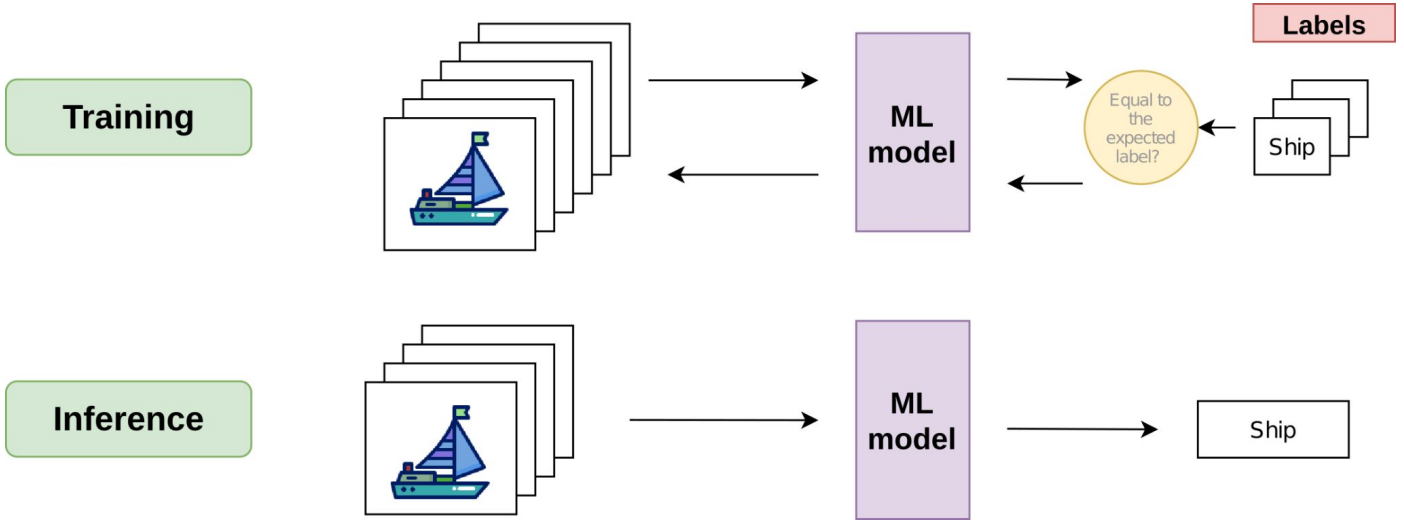
Generalization

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



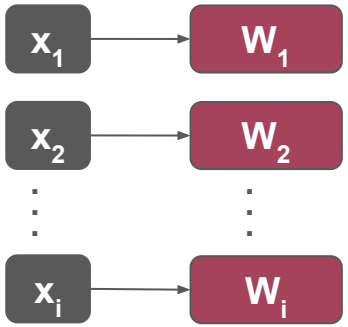
- In a classifier, **an input is mapped to a specific class**.
- Supervised training phase: **the network compares its current output with the desired output**. The difference between these two values is corrected using backpropagation.

Neural networks

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

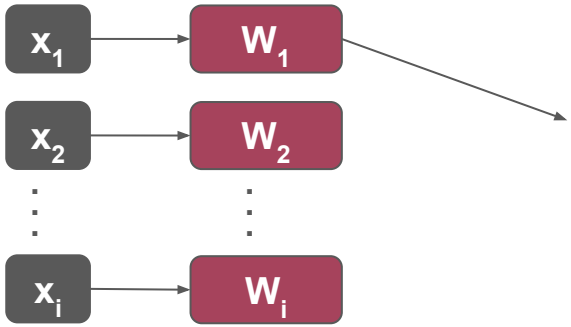
Neural networks

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



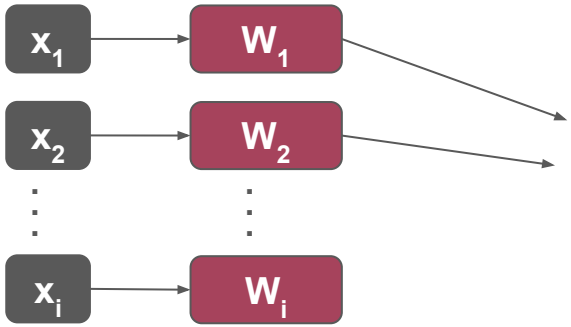
Neural networks

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



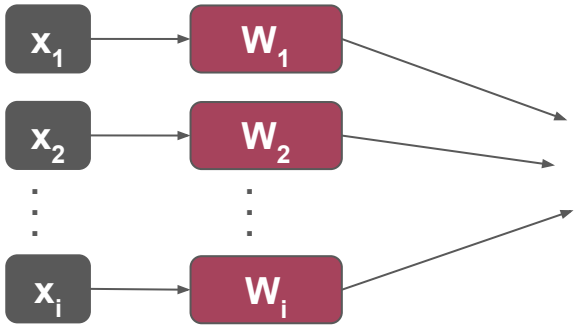
Neural networks

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



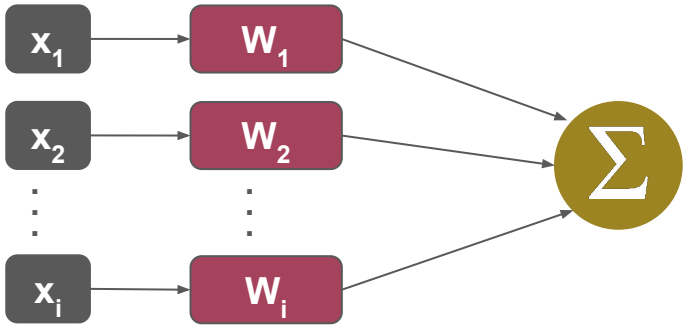
Neural networks

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



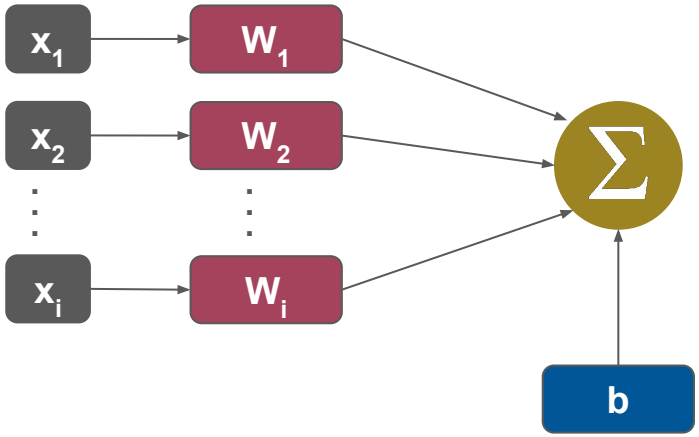
Neural networks

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



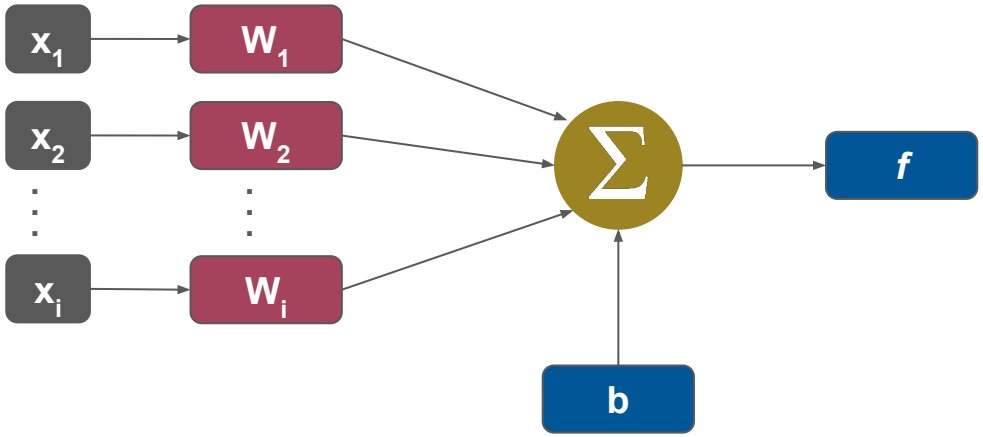
Neural networks

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



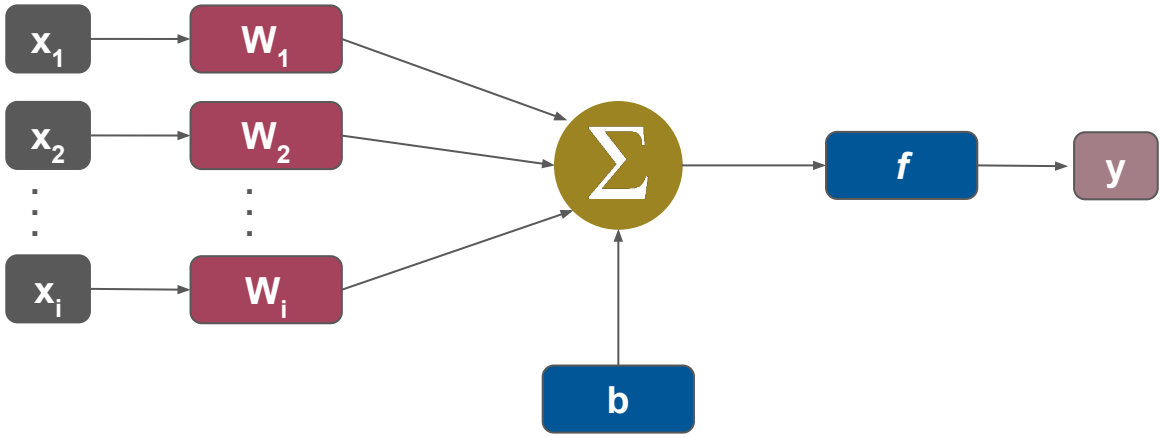
Neural networks

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.



Machine Learning and System-On-Chips based on FPGA

Machine Learning and System-On-Chips based on FPGA

FPGA / SoC-based on FPGA

Machine Learning and System-On-Chips based on FPGA

Low latency

FPGA / SoC-based on FPGA

Machine Learning and System-On-Chips based on FPGA

Low latency

Energy Efficiency

FPGA / SoC-based on FPGA

Machine Learning and System-On-Chips based on FPGA

Low latency

Energy Efficiency

High parallelism

FPGA / SoC-based on FPGA

Machine Learning and System-On-Chips based on FPGA

- Low latency
- Energy Efficiency
- High parallelism
- Scalability

FPGA / SoC-based on FPGA

Machine Learning and System-On-Chips based on FPGA

- Low latency
- Energy Efficiency
- High parallelism
- Scalability
- Customizable AI Acceleration

FPGA / SoC-based on FPGA

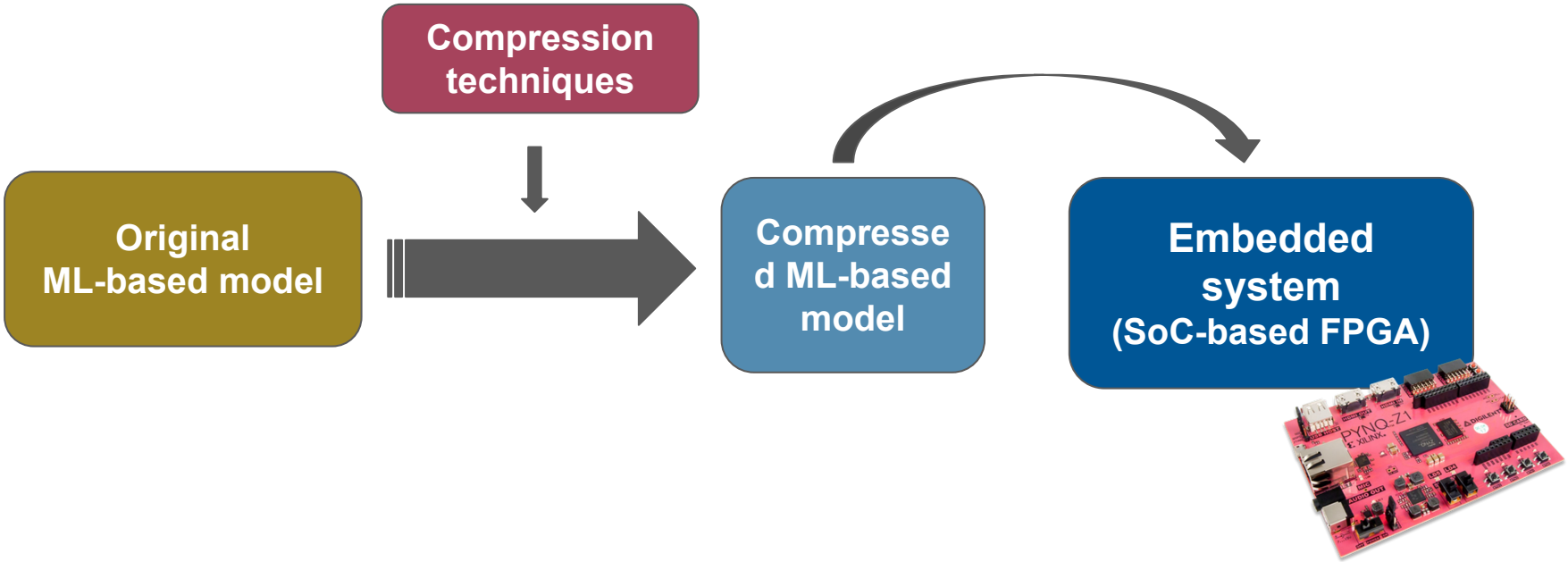
Machine Learning and System-On-Chips based on FPGA

- Low latency
- Energy Efficiency
- High parallelism
- Scalability
- Customizable AI Acceleration

FPGA / SoC-based on FPGA

Resource-constrained devices

Machine Learning and System-On-Chips based on FPGA



Machine Learning and System-On-Chips based on FPGA

- **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.

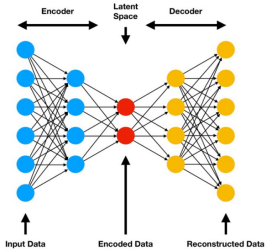
Machine Learning and System-On-Chips based on FPGA

- **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.

ML and model compression techniques

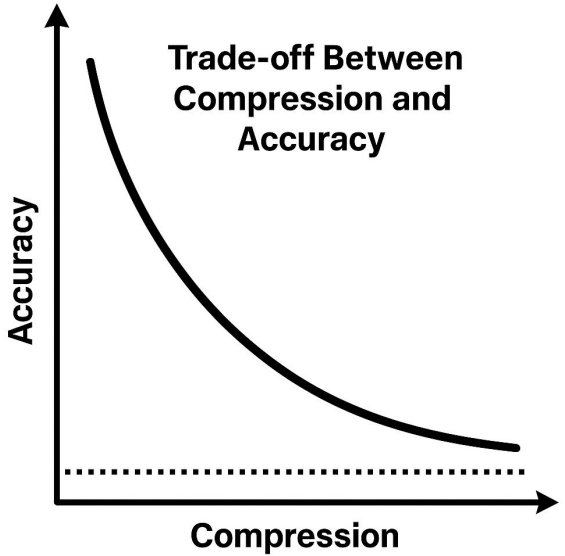
ML and model compression techniques

- **Compression,** in the context of Machine Learning, involves techniques aiming to reduce the size of models or datasets while preserving performance.
    - **Model Compression:** Reducing the size of machine learning models (such as neural networks) without significantly affecting their accuracy.
    - **Data compression:** Reducing the size of the data used for training, validation, and testing. Example: Autoencoders.



ML and model compression techniques

- **Trade-off Between Compression and Accuracy**
  - Balance between reducing the size of the model or data and maintaining its performance.



ML and model compression techniques

- **Problem:**
  - **Very large models → high memory consumption and inference time.**

The most accurate models are large and expensive in terms of memory and processing time.

ML and model compression techniques

The most accurate models (such as deep neural networks) are large and expensive in terms of memory and processing time.

Model	Parameters (millions)	Disk size (MB)
ResNet-50	~25.6M	~98 MB
BERT-base	~110M	~440 MB
BERT-large	~340M	~1.3 GB
VGG-16	~138M	~528 MB
VGG-19	~144M	~548 MB
YOLOv3	~62M	~236 MB
SpineNet-49S (small)	~11M	~45 MB
MobileNetV3-Large	~5.4M	~20 MB

ML and model compression techniques

- **Problem:**
  - **Very large models → high memory consumption and inference time.**

- **Solution:**
  - **Compression**

ML and model compression techniques

Distilled versions

Model	Parameters (millions)	Disk size (MB)	Accuracy
DistilBERT	66M (↓40%)	~250 MB	97% of BERT
SqueezeNet	1.2M (↓99%)	~4.8 MB (↓99%)	Similar to VGG-16
YOLOv8-Nano	3.2M (↓92%)	~8 MB (↓90%)	Good trade-off with YOLOv8-L
YOLO-Fastest	0.2M (↓99.5%)	~1 MB (↓99%)	Lower accuracy

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

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.

ML and model compression techniques

for reconfigurable hardware accelerators

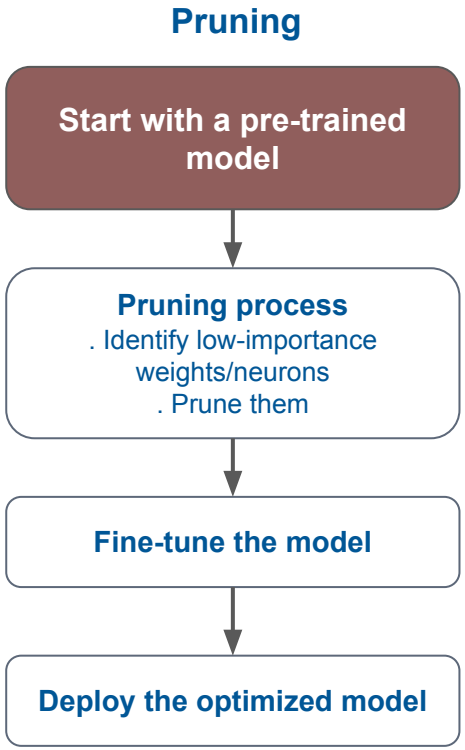
Ensemble of compression techniques - Exploration of the interplay between:

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.

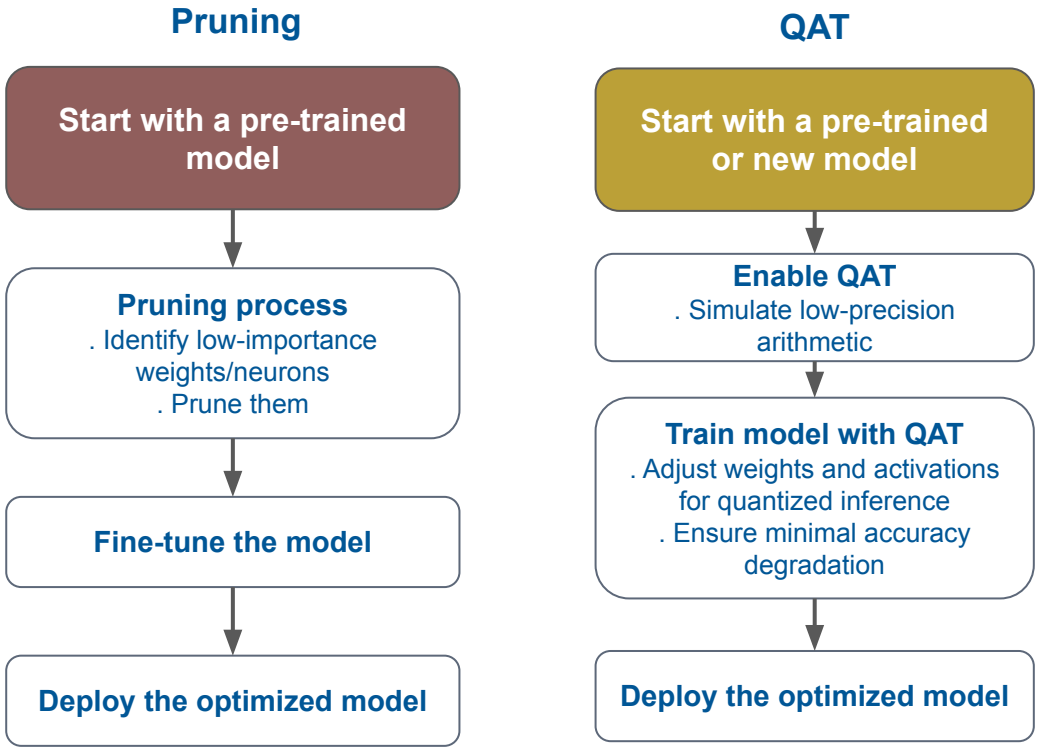
Fully on-chip deployment

How do we combine compression techniques?

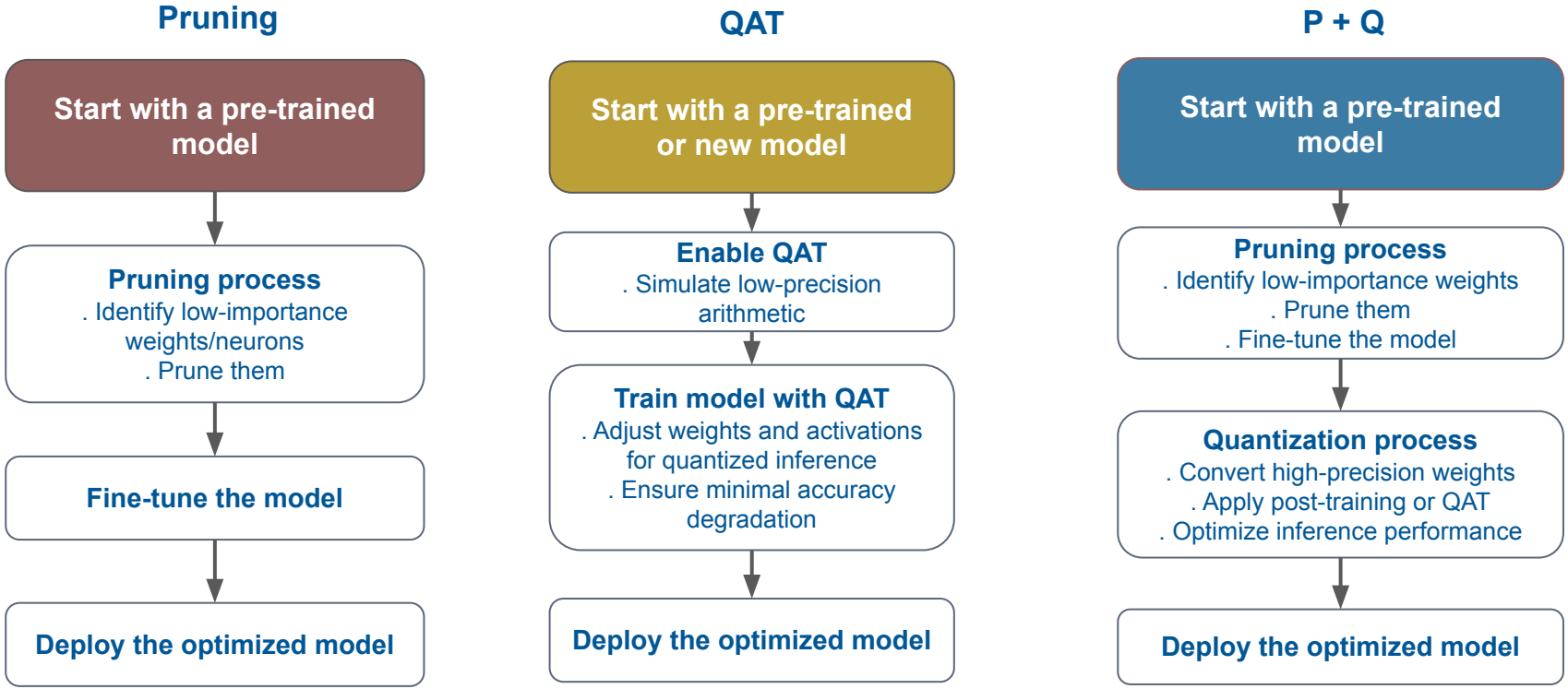
How do we combine compression techniques?



How do we combine compression techniques?



How do we combine compression techniques?



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

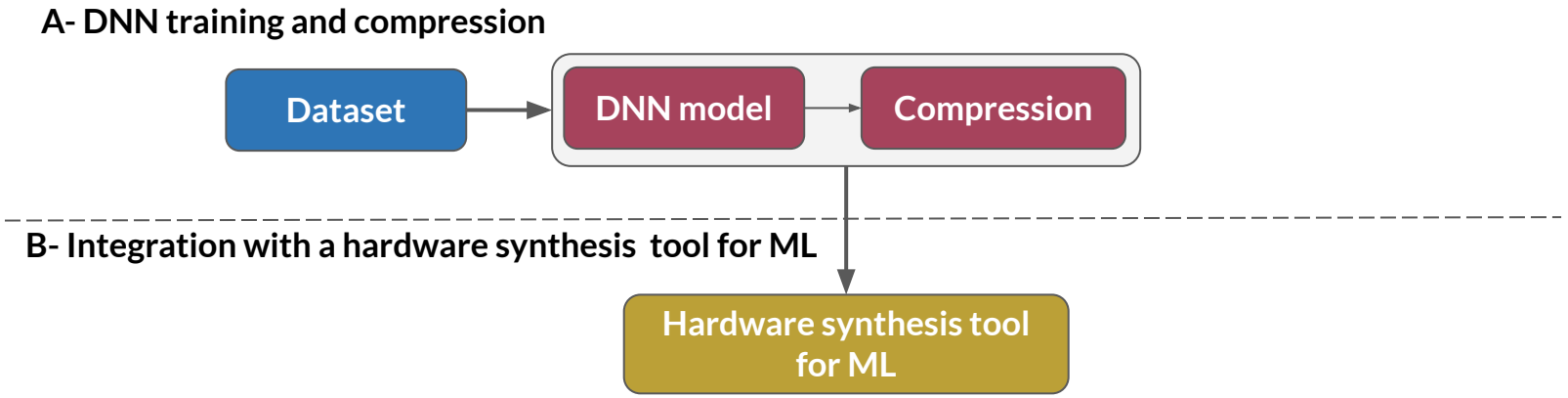
End-to-end workflow

A- DNN training and compression

Publication:
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.

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

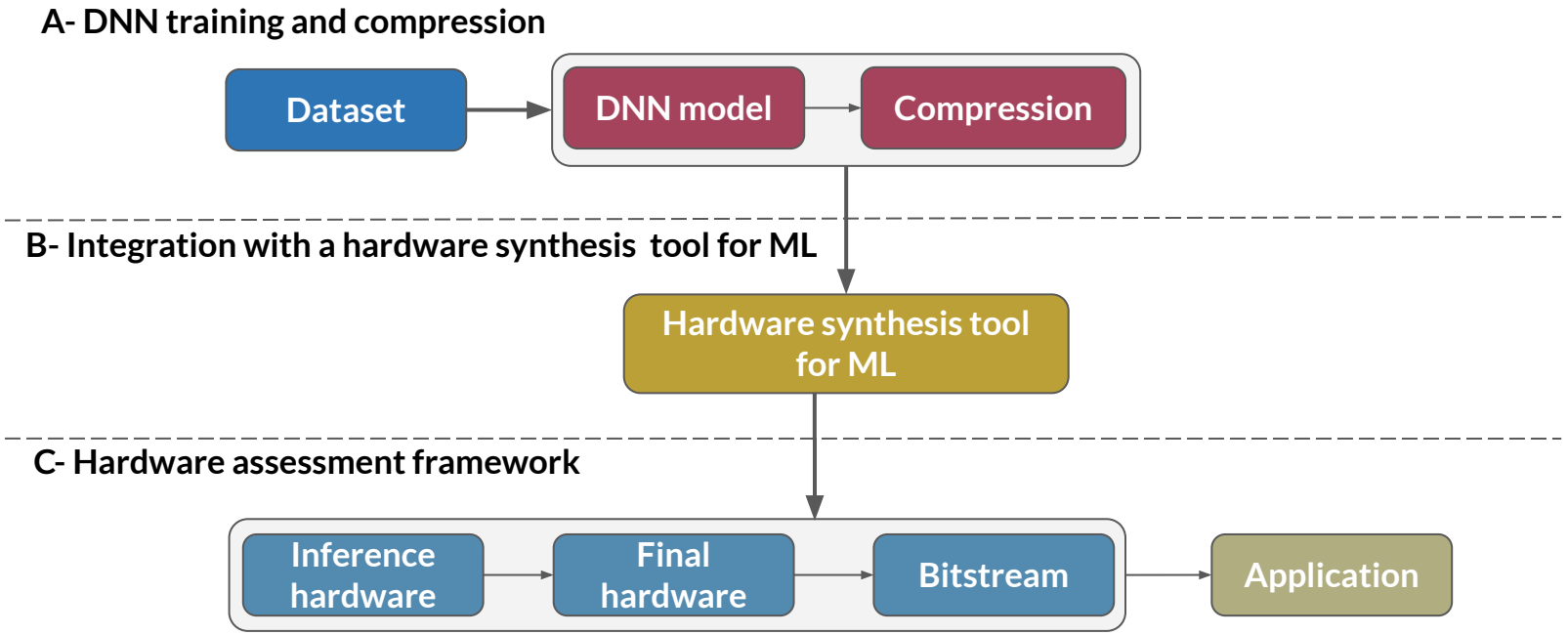
End-to-end workflow



Publication:
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.

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

End-to-end workflow



Publication:

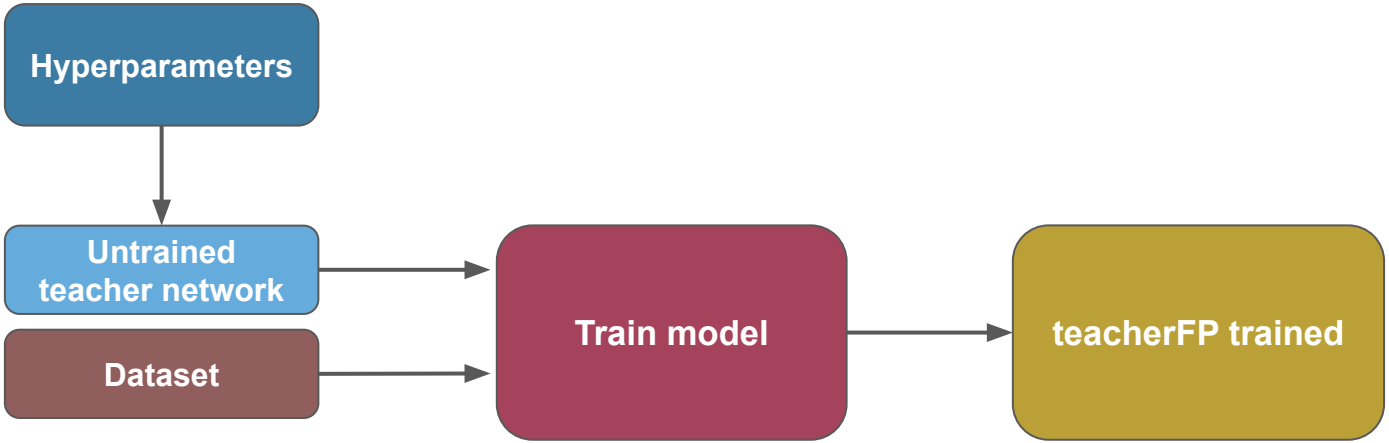
Molina, R. S., Morales, I. R., Crespo, M. L., Costa, V. G., Carrato, S., & Ramponi, G. (2023). "An end-to-end workflow to efficiently compress and deploy DNN classifiers on SoC/FPGA". *IEEE Embedded Systems Letters*, *16*(3), 255-258.

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

A. DNN training and compression

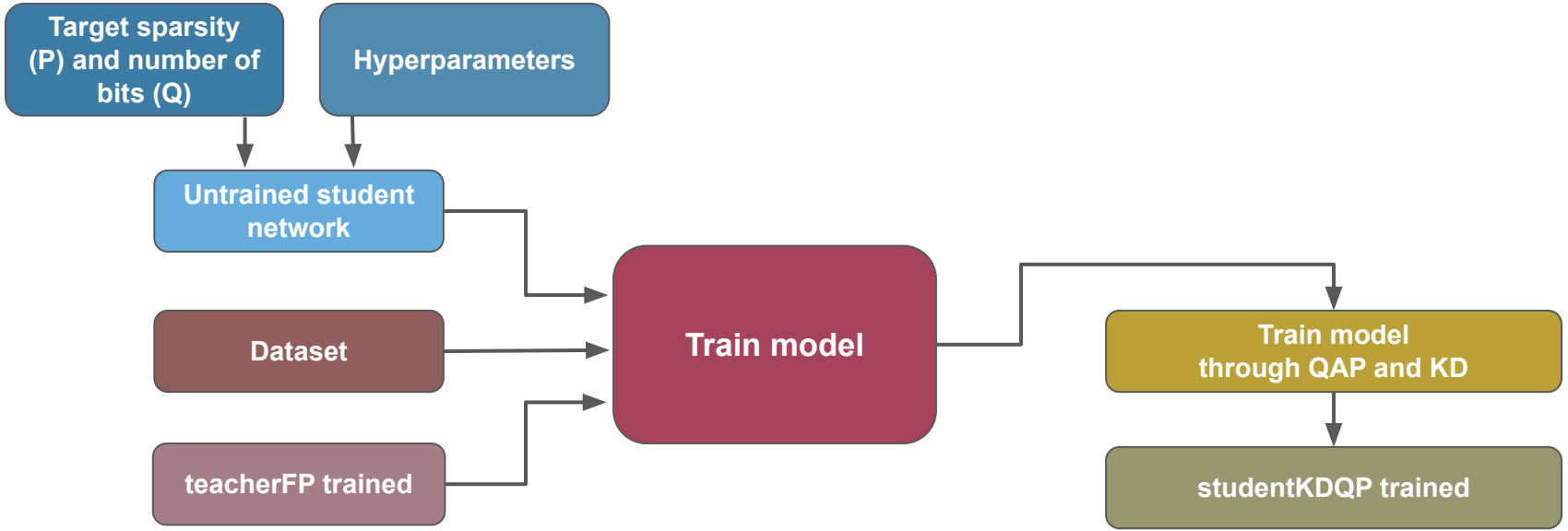
DNN training and compression

Stage 1 - Teacher training



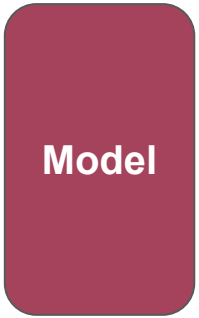
DNN training and compression

Stage 2 - Student training



B. Integration with a hardware synthesis tool for ML

Integration with a hardware synthesis tool for ML



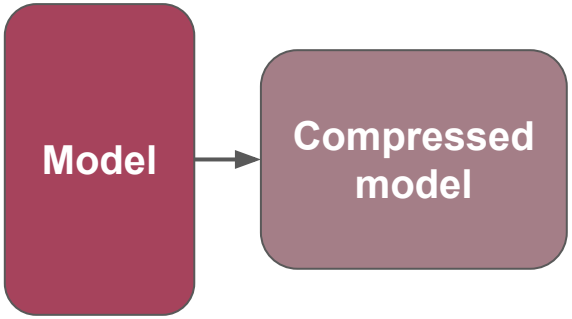
PYTORCH

ONNX

K + TensorFlow

https://github.com/fastmachinelearning/

Integration with a hardware synthesis tool for ML



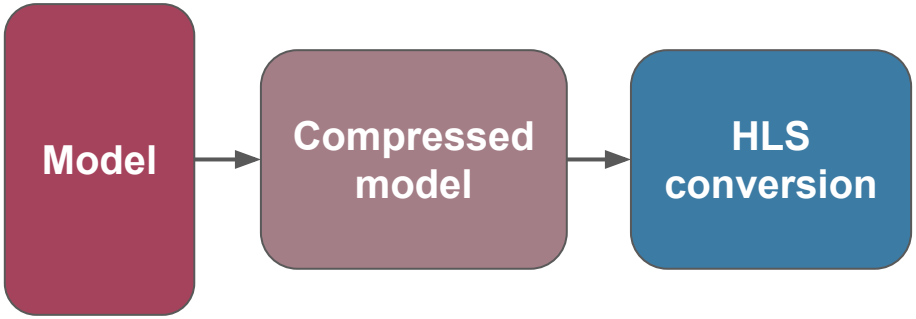
PYTORCH

ONNX

K + TensorFlow

https://github.com/fastmachinelearning/

Integration with a hardware synthesis tool for ML



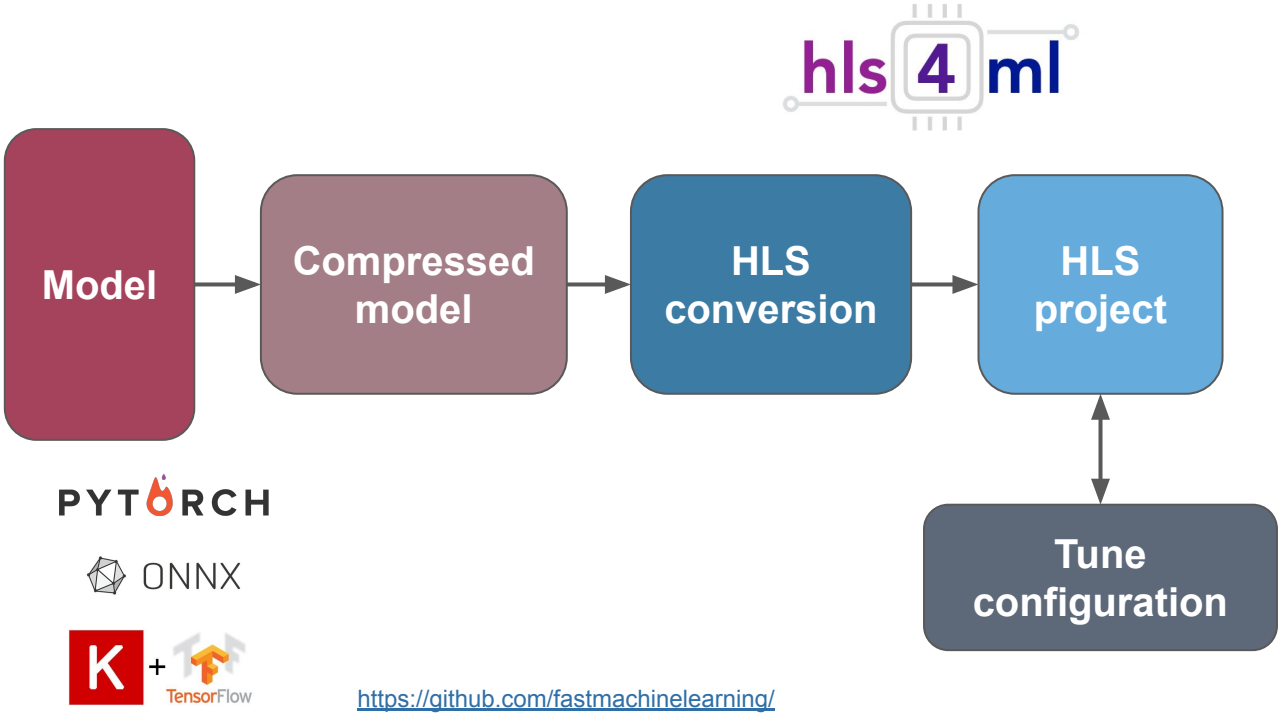
PYTORCH

ONNX

K + TensorFlow

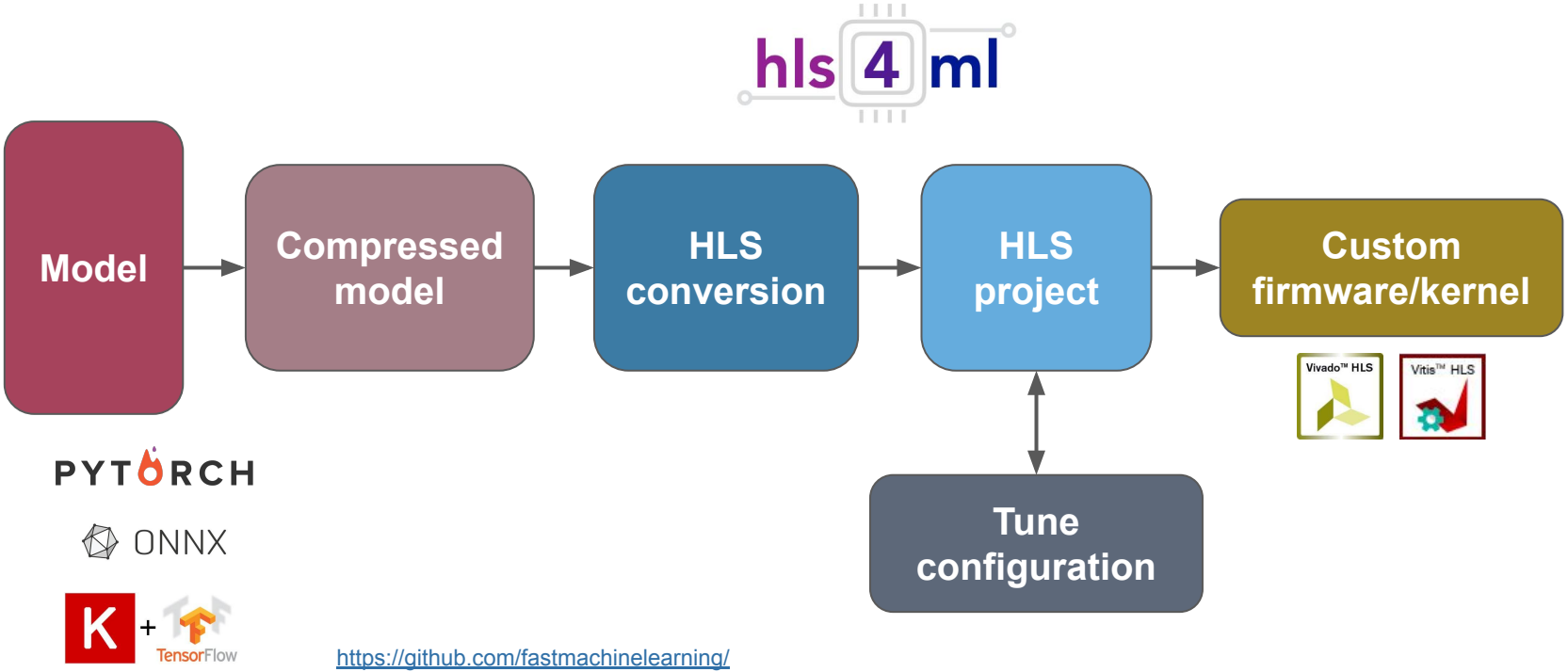
https://github.com/fastmachinelearning/

Integration with a hardware synthesis tool for ML



https://github.com/fastmachinelearning/

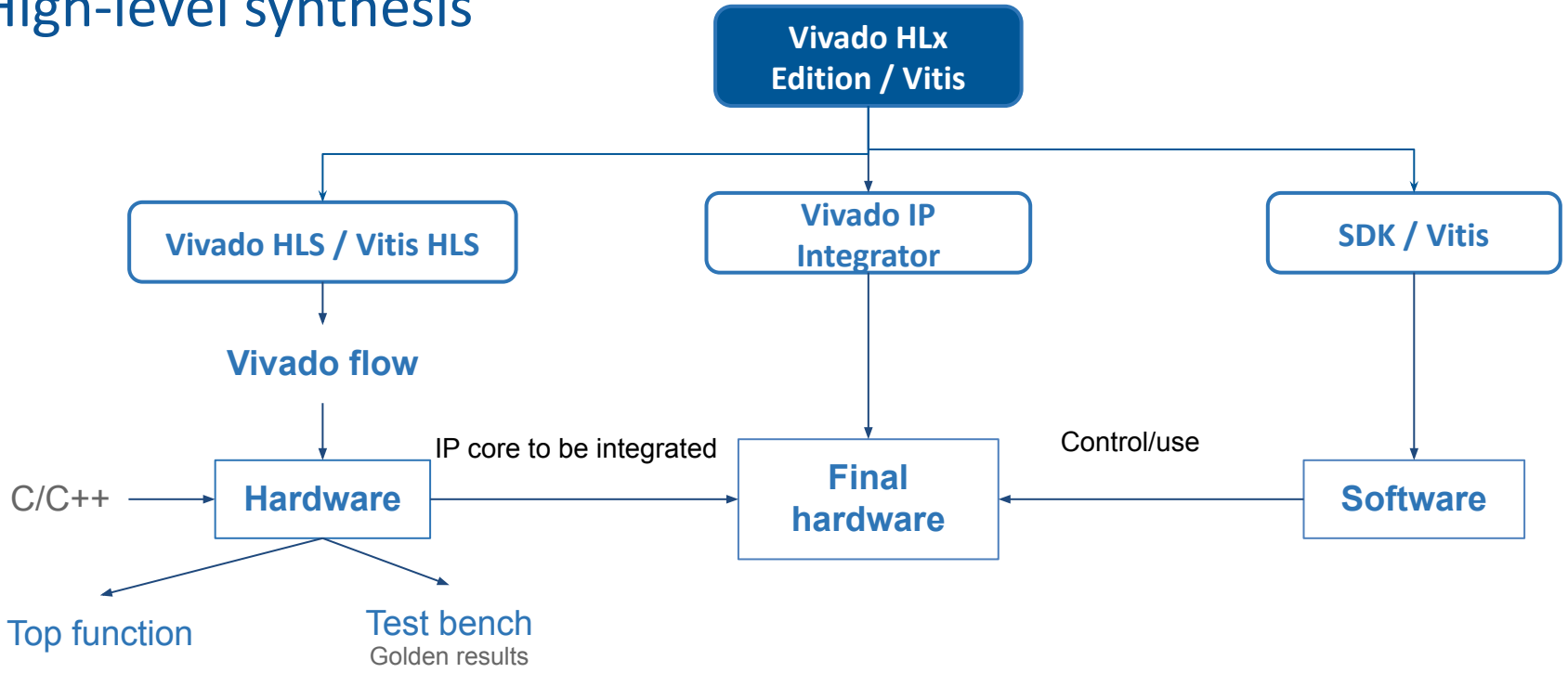
Integration with a hardware synthesis tool for ML



https://github.com/fastmachinelearning/

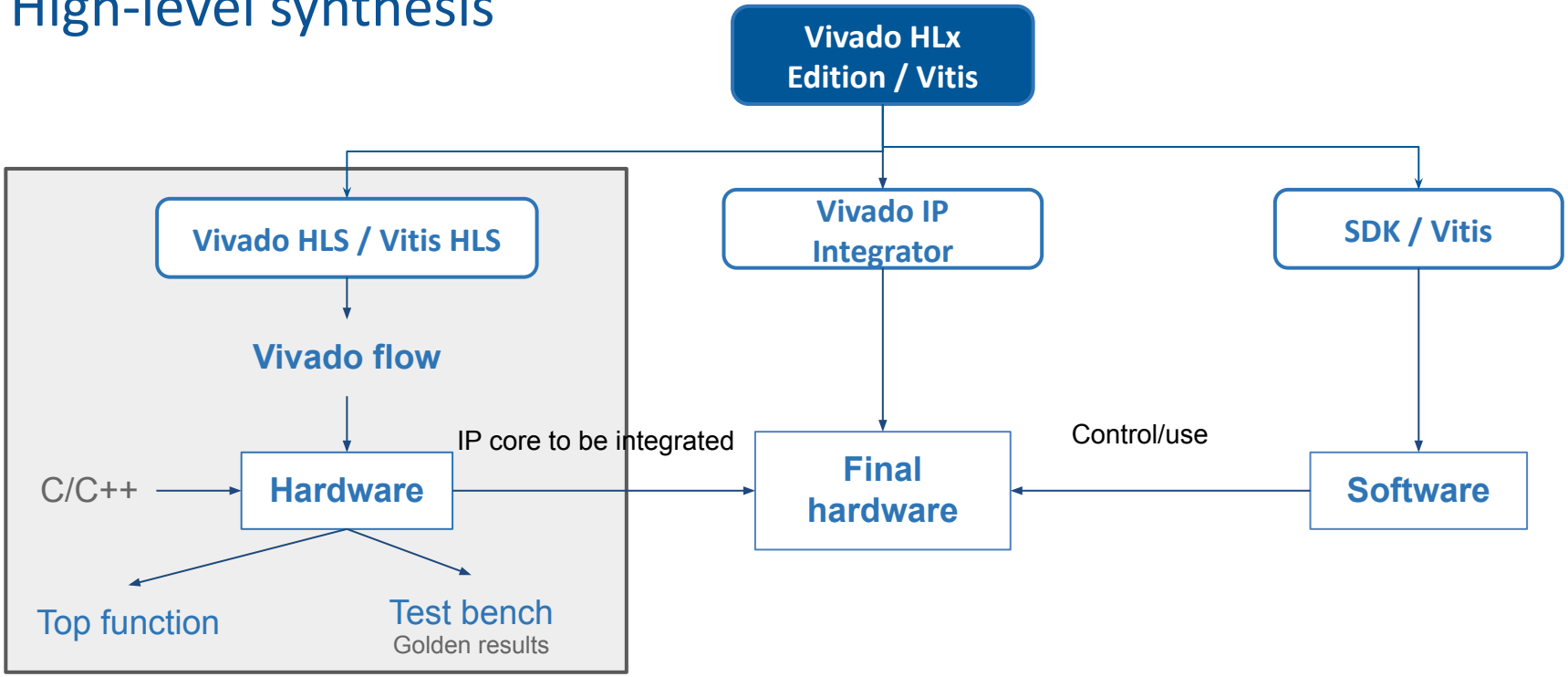
Integration with a hardware synthesis tool for ML

High-level synthesis



Integration with a hardware synthesis tool for ML

High-level synthesis



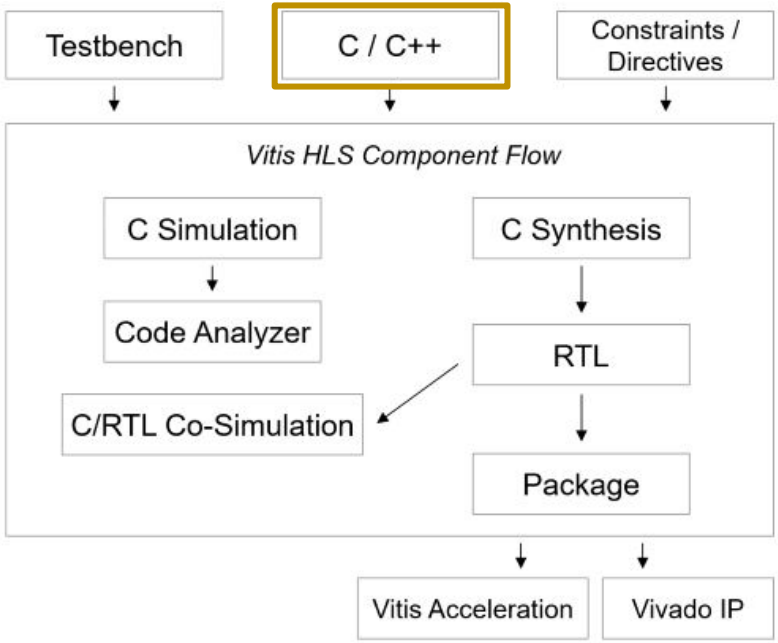
Integration with a hardware synthesis tool for ML

High-level synthesis

- **High-Level Synthesis**
  - It provides the facility to create RTL from a high level of abstraction.
  - Several implementations are possible from the same source description.
  - Implements the design based on defaults and user applied directives.
  - It allows the optimization of the input code using directives to:
    - Reduce latency.
    - Improve performance and throughput.
    - Reduce resource utilization.

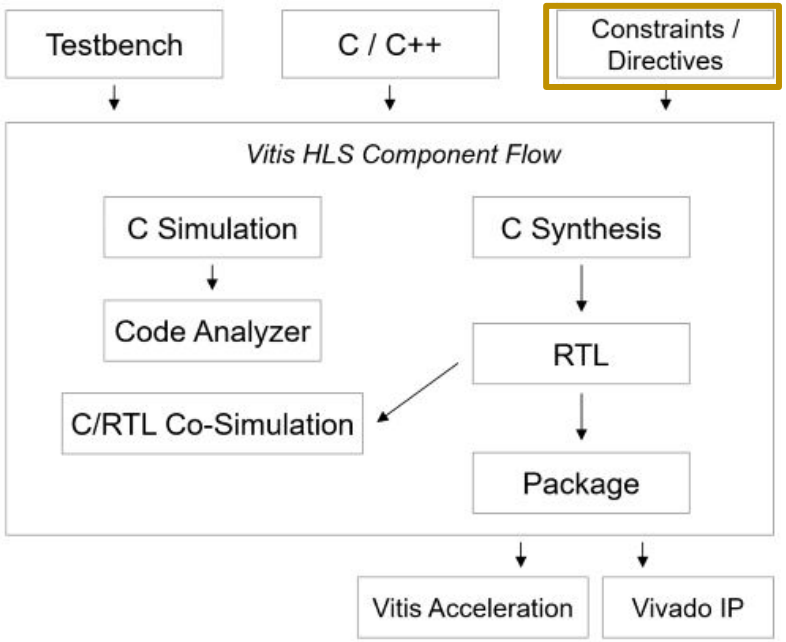
Without optimization, HLS tool will look to minimize latency and improve concurrency.

HLS Component Development Flow



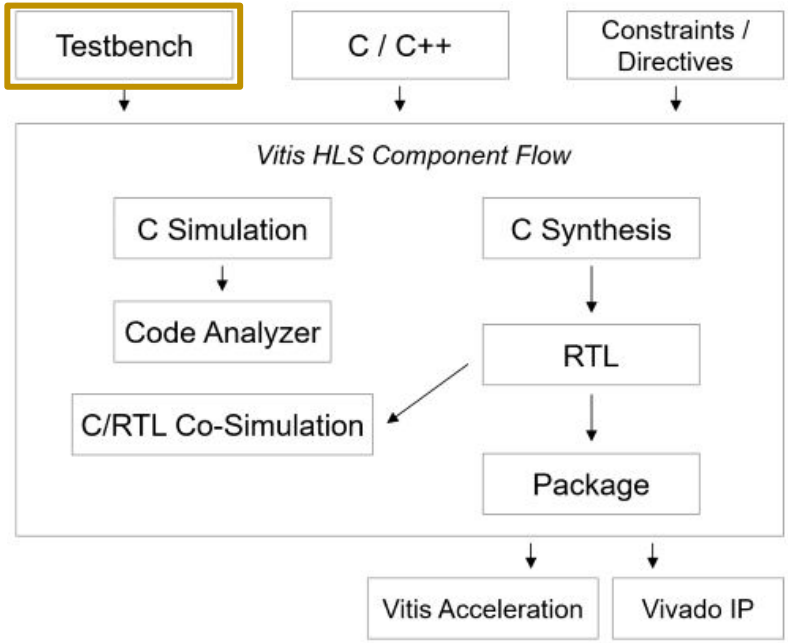
https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

HLS Component Development Flow



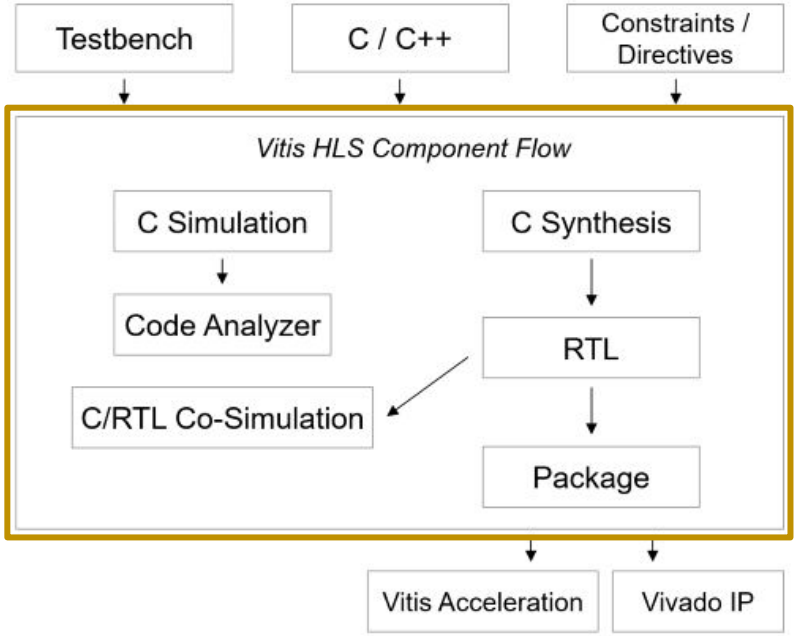
https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

HLS Component Development Flow



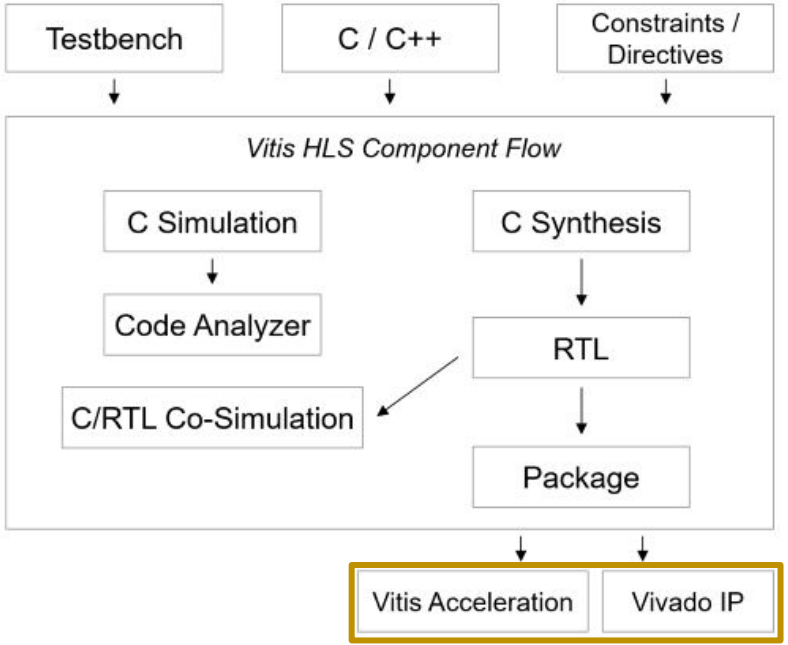
https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

HLS Component Development Flow



https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

HLS Component Development Flow



https://docs.amd.com/r/en-US/ug1399-vitis-hls/Introduction-to-Vitis-HLS-Components

Integration with a hardware synthesis tool for ML

ML framework support:

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

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

Integration with a hardware synthesis tool for ML

ML framework support:

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

Neural networks architectures:

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

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

Integration with a hardware synthesis tool for ML

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)

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

High-Level Synthesis for Machine Learning

hls4ml is tested on the following platforms:

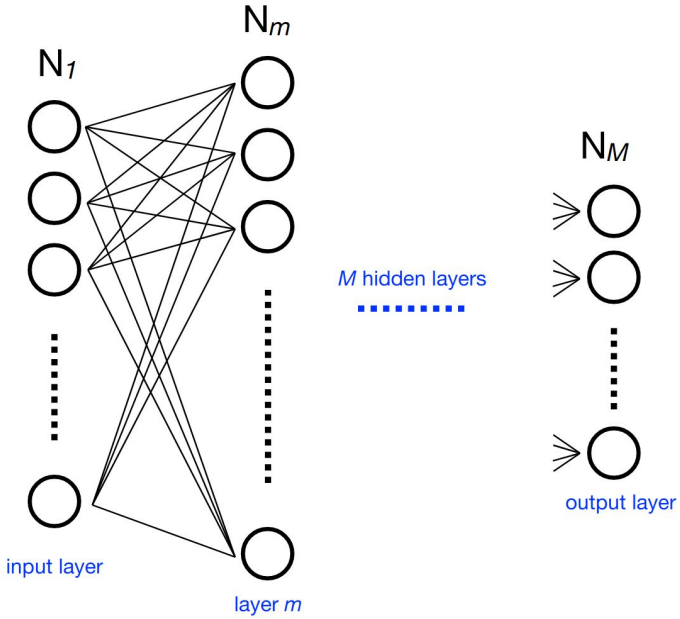
Vivado HLS versions 2018.2 to 2020.1
Intel HLS versions 20.1 to 21.4. Versions > 21.4 have not been tested.
Vitis HLS versions 2022.2 to 2024.1. Versions <= 2022.1 are known not to work.
Catapult HLS versions 2024.1_1 to 2024.2
oneAPI versions 2024.1 to 2025.0

[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.



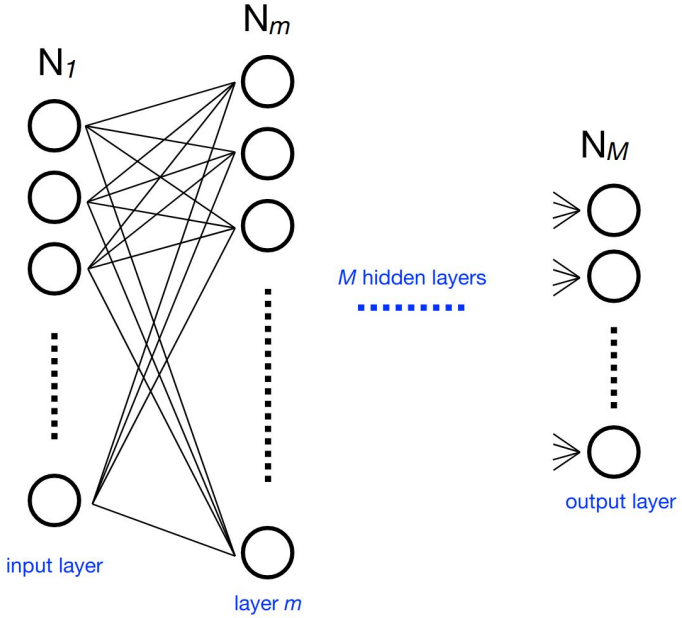
[hls4ml] https://fastmachinelearning.org/hls4ml/

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/

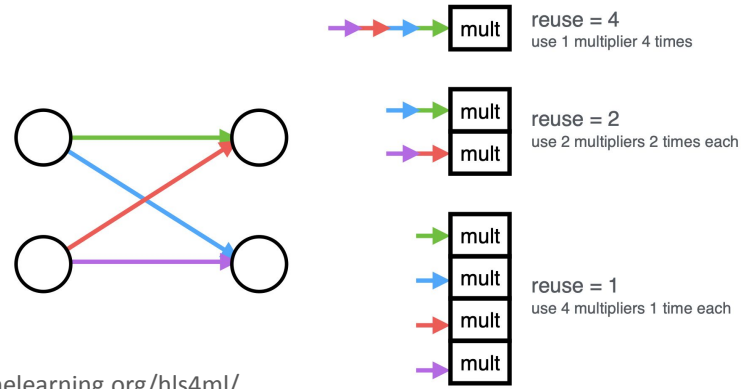
Trade-off between latency and FPGA resource usage determined by the parallelization of the calculations in each layer.

[hls4ml] 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.

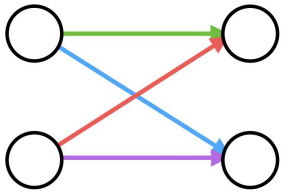


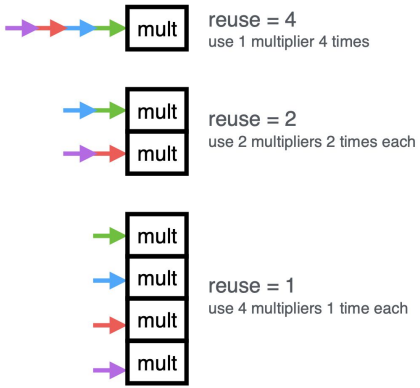
[hls4ml] https://fastmachinelearning.org/hls4ml/

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

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

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

[hls4ml] 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

[hls4ml] 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.

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

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.

[hls4ml] 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.

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

Activations - Implementation parameter

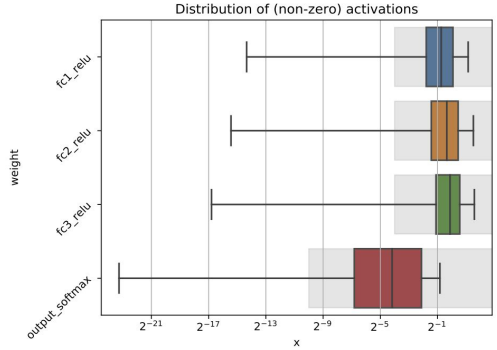
- **latency:** Good latency, high resource usage. **It does not work well if there are many output classes**.
- **stable:** Slower but with better accuracy, useful in scenarios where **higher accuracy is needed**.
- **legacy:** An older implementation with poor accuracy, but good performance. Usually 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.

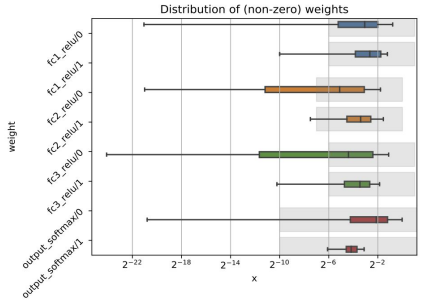
[hls4ml] https://fastmachinelearning.org/hls4ml/

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.**





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

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

Integration with a hardware synthesis tool for ML

Python integration

```
1 from tensorflow.keras.models import load_model
2 from sklearn.metrics import accuracy_score
3 model = load_model('model_keras_MLP.h5')
4 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


```
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 (QAT)

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)
```

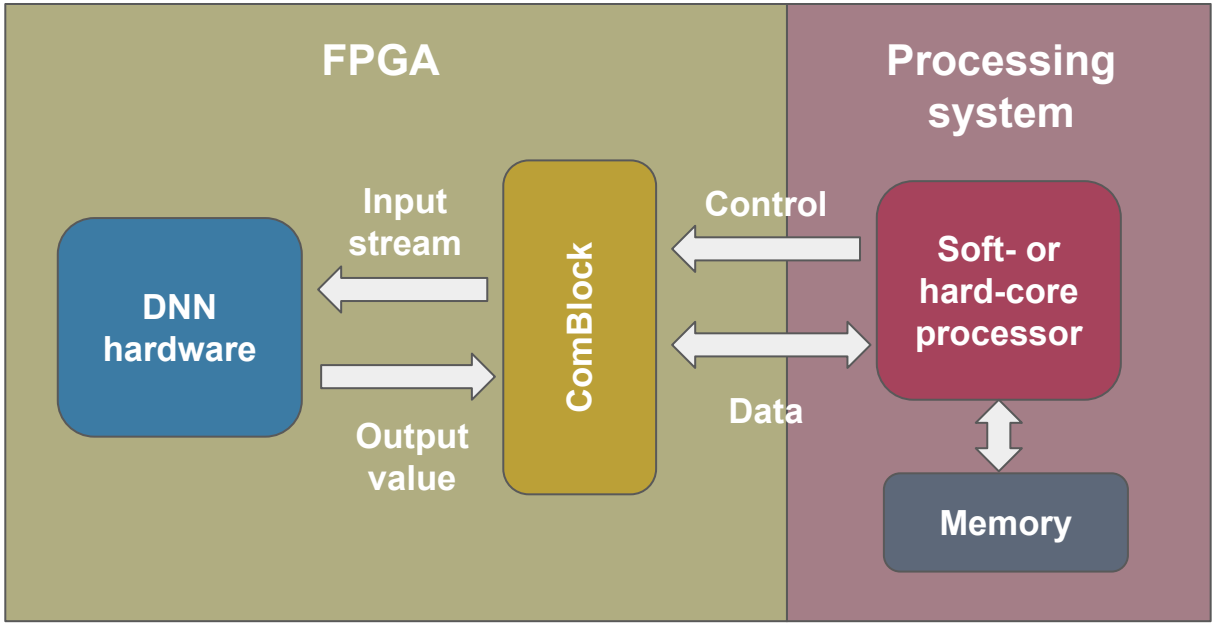
Integration with a hardware synthesis tool for ML

High-level synthesis project generated with hls4ml

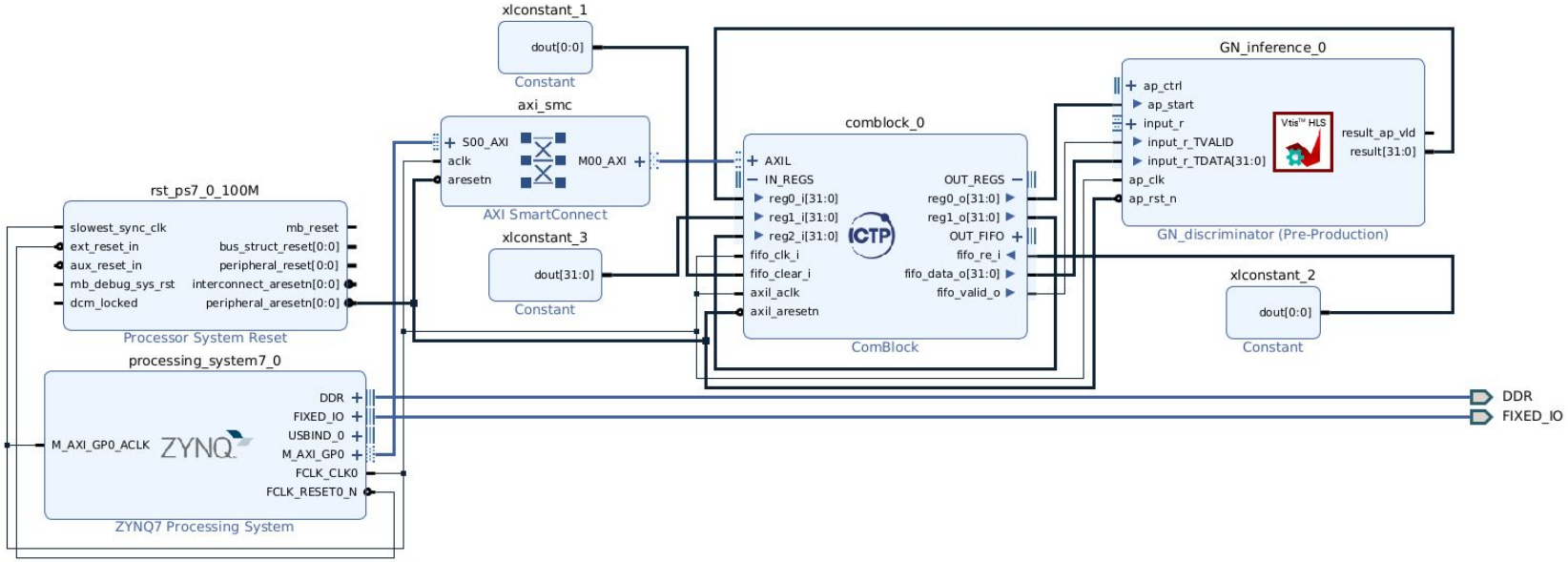

```
layer2_t layer2_out[N_LAYER_2];
#pragma HLS ARRAY_PARTITION variable=layer2_out complete dim=0
nnet::dense<input_t, layer2_t, config2>(input_4, layer2_out, w2, b2); // fc1

layer3_t layer3_out[N_LAYER_2];
#pragma HLS ARRAY_PARTITION variable=layer3_out complete dim=0
nnet::linear<layer2_t, layer3_t, linear_config3>(layer2_out, layer3_out); // fc1_linear

layer5_t layer5_out[N_LAYER_5];
#pragma HLS ARRAY_PARTITION variable=layer5_out complete dim=0
nnet::dense<layer3_t, layer5_t, config5>(layer3_out, layer5_out, w5, b5); // fc2

layer6_t layer6_out[N_LAYER_5];
#pragma HLS ARRAY_PARTITION variable=layer6_out complete dim=0
nnet::linear<layer5_t, layer6_t, linear_config6>(layer5_out, layer6_out); // fc2_linear

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
```

C. Hardware assessment framework

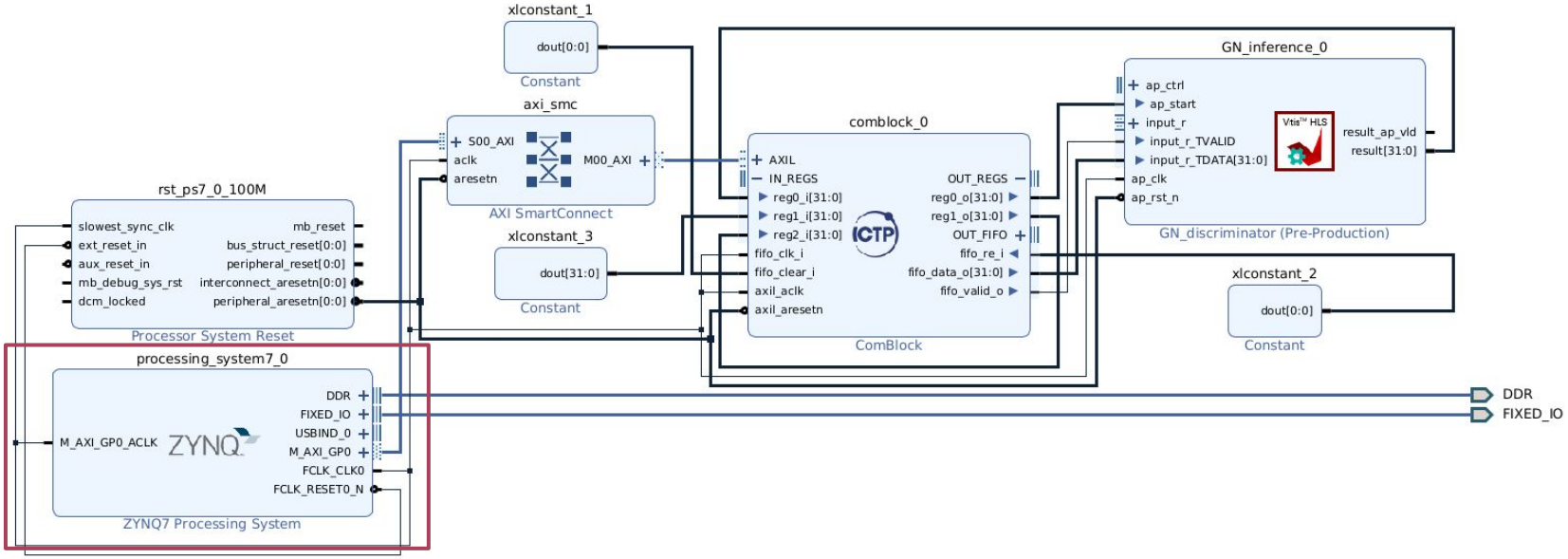
Hardware assessment framework



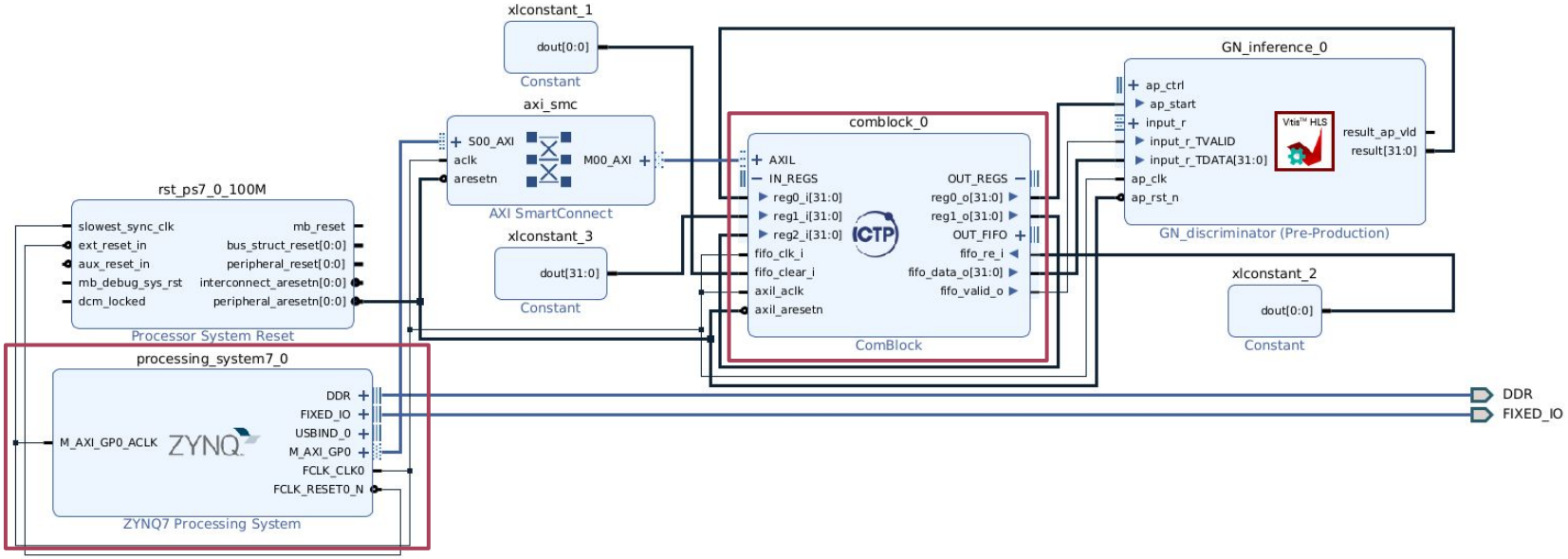
Hardware assessment framework



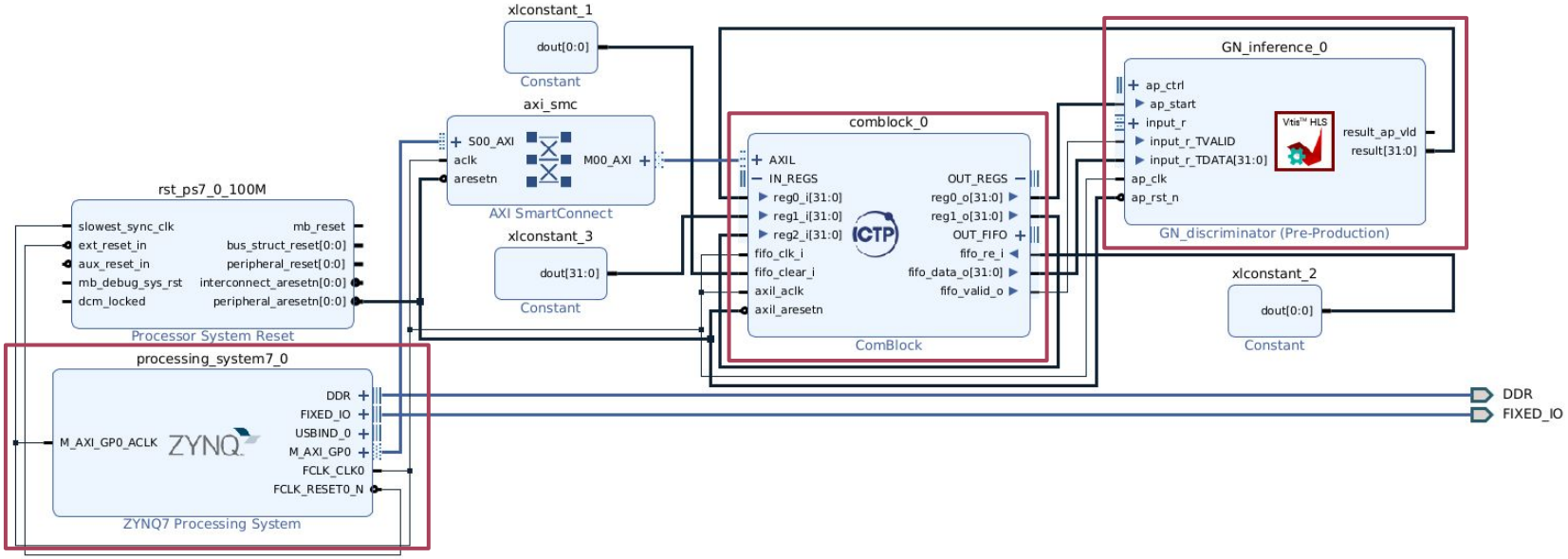
Hardware assessment framework



Hardware assessment framework



Hardware assessment framework



Machine Learning and SoC-based FPGA for real-case applications

Machine Learning and SoC-based FPGA for real-case applications

- Gamma/Neutron discrimination.
- Gamma/Neutron discrimination for diamond detector.
- Pest classification in fruit crops.
- Pulse shape discriminator for cosmic rays studies.
- Volcanic seismic event detection.
- Water quality monitoring applied to Dunav river.
- Object detection for adverse weather conditions, particularly haze and fog.

Gamma/Neutron discrimination

Machine Learning and SoC-based FPGA for real-case applications

Gamma/neutron discrimination

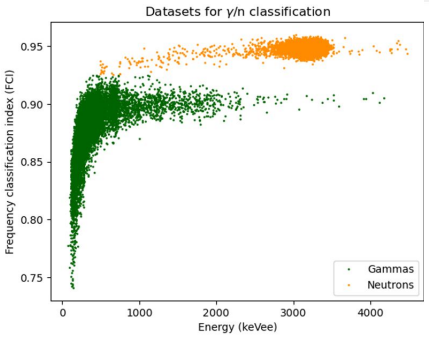
IAEA
International Atomic Energy Agency

- 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 total gamma and neutron events in this dataset are 10,913 and 27,696, respectively.**

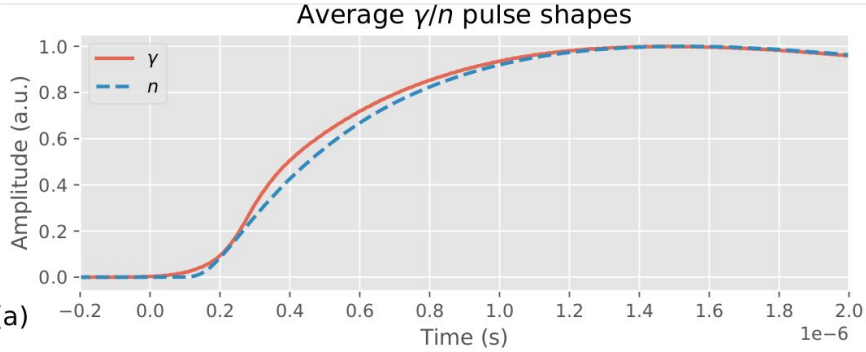
Machine Learning and SoC-based FPGA for real-case applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency



(a)



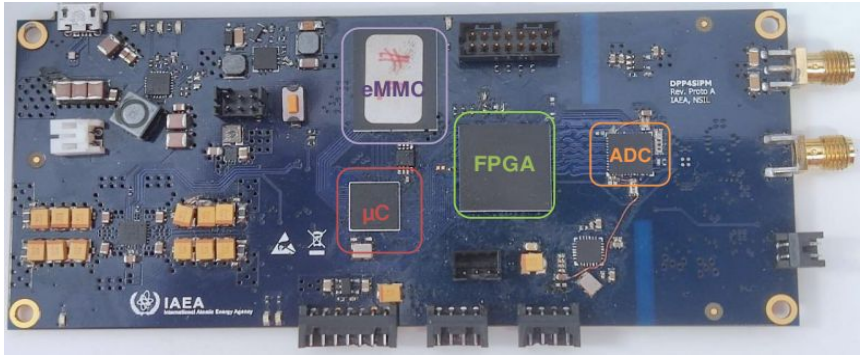
Morales, I. R., Crespo, M. L., Bogovac, M., Cicuttin, A., Kanaki, K., & Carrato, S. (2023). Gamma/neutron classification with SiPM CLYC detectors using frequency-domain analysis for embedded real-time applications. Nuclear Engineering and Technology.

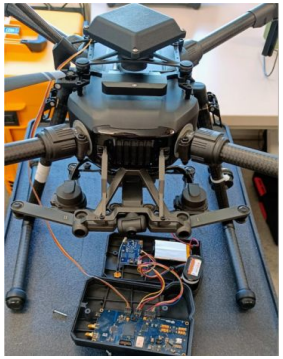
Dataset from https://doi.org/10.5281/zenodo.8037059

Machine Learning and SoC-based FPGA for real-case applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency



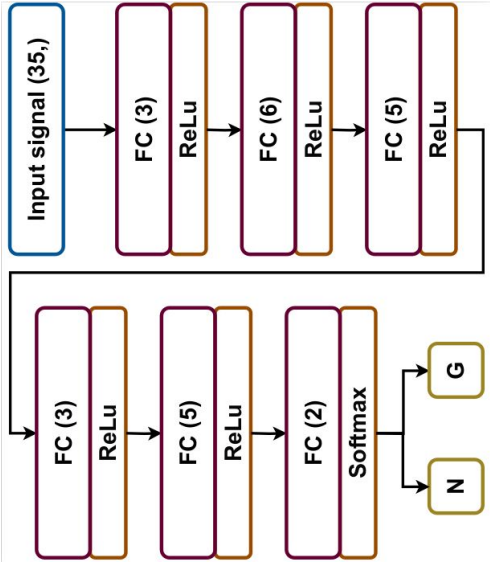


Publication:
Morales, I. R., Molina, R. S., Bogovac, M., Jovalekic, et al. (2024). *Gamma/neutron online discrimination based on machine learning with CLYC detectors.* IEEE Transactions on Nuclear Science.

Machine Learning and SoC-based FPGA for real-case applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

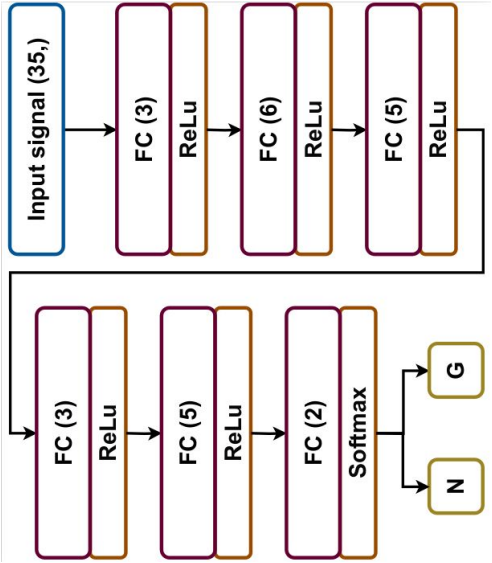


Publication:
Morales, I. R., Molina, R. S., Bogovac, M., Jovalekic, et al. (2024). *Gamma/neutron online discrimination based on machine learning with CLYC detectors.* IEEE Transactions on Nuclear Science.

Machine Learning and SoC-based FPGA for real-case 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.

Publication:
Morales, I. R., Molina, R. S., Bogovac, M., Jovalekic, et al. (2024). *Gamma/neutron online discrimination based on machine learning with CLYC detectors.* IEEE Transactions on Nuclear Science.

Machine Learning and SoC-based FPGA for real-case applications

Gamma/neutron discrimination

IAEA
International Atomic Energy Agency

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

Publication:
Morales, I. R., Molina, R. S., Bogovac, M., Jovalekic, et al. (2024). *Gamma/neutron online discrimination based on machine learning with CLYC detectors.* IEEE Transactions on Nuclear Science.

Machine Learning and SoC-based FPGA for real-case 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%**

Publication:
Morales, I. R., Molina, R. S., Bogovac, M., Jovalekic, et al. (2024). *Gamma/neutron online discrimination based on machine learning with CLYC detectors.* IEEE Transactions on Nuclear Science.

Machine Learning and SoC-based FPGA for real-case 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)**

Publication:

Morales, I. R., Molina, R. S., Bogovac, M., Jovalekic, et al. (2024). *Gamma/neutron online discrimination based on machine learning with CLYC detectors.* IEEE Transactions on Nuclear Science.

Volcanic seismic event detection

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Copahue Volcano - Geological setting and data

Image from
https://news.sky.com/story/chile-volcano-red-alert-issued-for-copahue-10444642

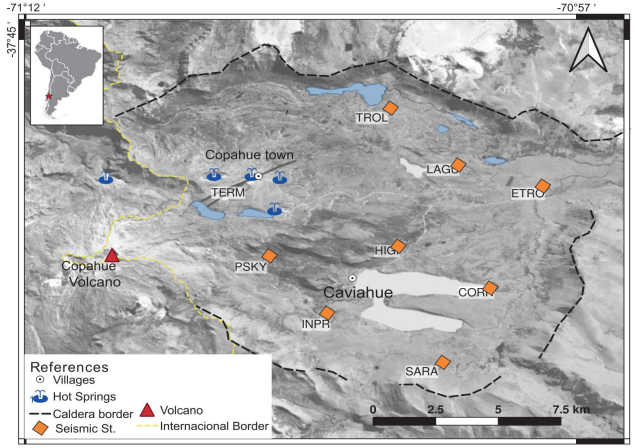
Publication:

Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Copahue Volcano - Geological setting and data



Between December 2017 and March 2018, the National University of Río Negro deployed a temporary network (called CP) of six broad-band and two-short period seismic stations.

CP covered an area of 12 km in the East-West (E-W) direction and 14 km in the North-South (N-S) direction in Caldera del Agrio.

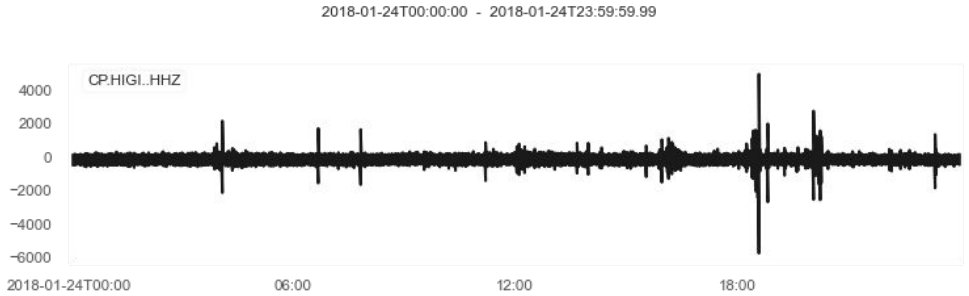
Publication:

Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Copahue Volcano - Geological setting and data



Raw data* acquired from sensors installed at different stations that formed the CP network. Each station generates signals in **three channels: vertical, eastern, and northern**. These files have MSEED format and each is **24h long**.

Publication:

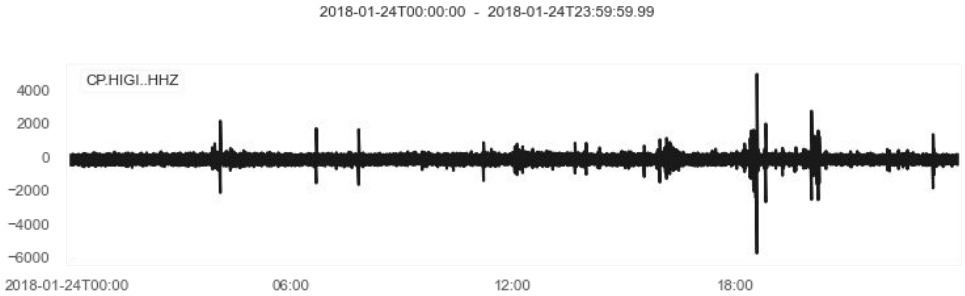
Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

*The signals used in this research were obtained from Montenegro's Ph.D. thesis, and were provided by "Laboratorio de Estudios y Seguimientos de Volcanes Activos' (LESVA), Universidad Nacional de Rio Negro.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Copahue Volcano - Geological setting and data



Seismic traces recorded at the HIGI station.

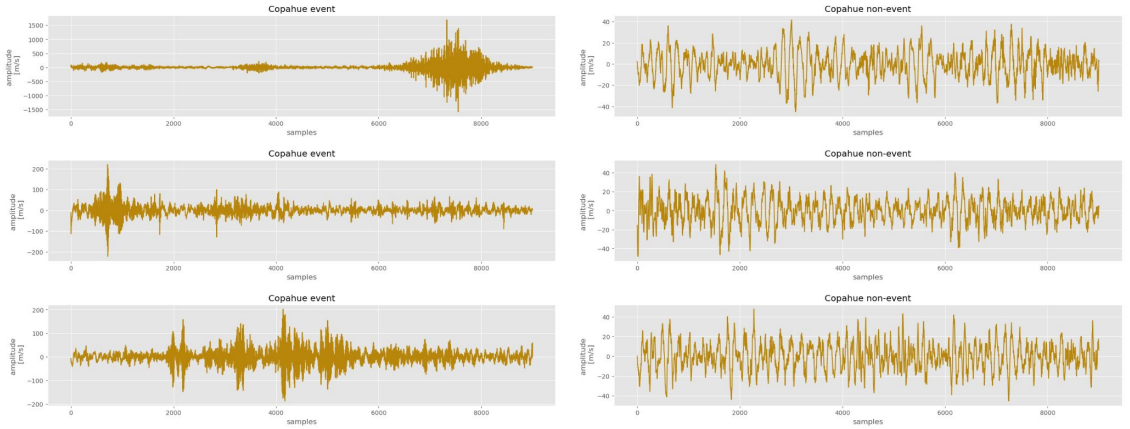
Publication:

Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Copahue Volcano - Geological setting and data



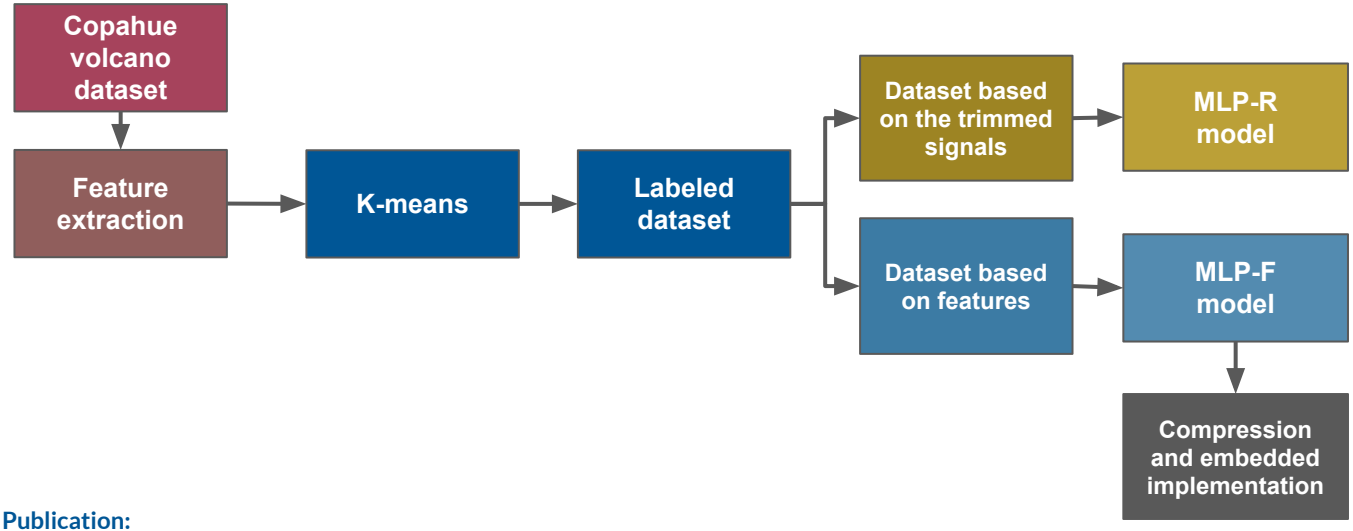
Publication:

Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Workflow for ML training



Publication:

Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

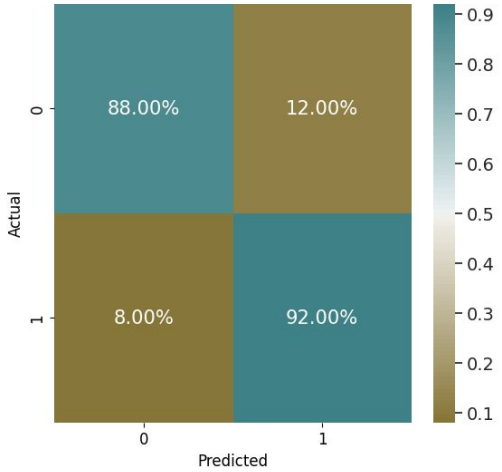
Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Model compression

ML-model with features as input

- Quantization - 8 bits
- Pruning (30 % sparsity)



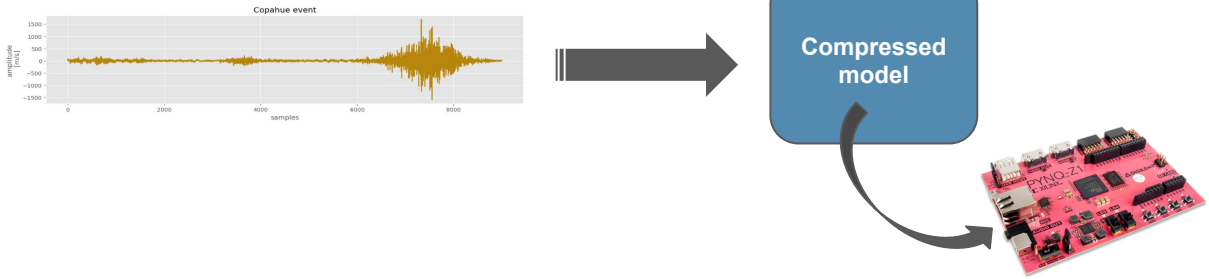
Publication:

Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Model deployment

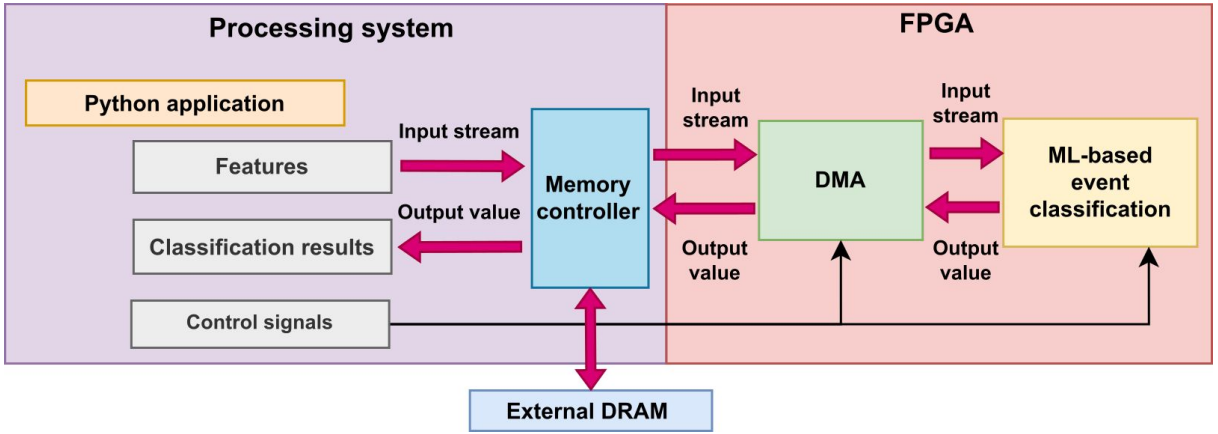


Publication:
Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

Machine Learning and SoC-based FPGA for real-case applications

Copahue volcano seismic event detection

Model deployment



Publication:
Sosa, Y. M., Molina, R. S., Spagnotto, S., Melchor, I., Nuñez Manquez, A., Crespo, M. L., ... & Petrino, R. (2024). *Seismic event detection in the copahue volcano based on machine learning: towards an on-the-edge implementation*. Electronics, 13(3), 622.

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.

1st Mesoamerican Workshop on Reconfigurable X-ray Scientific Instrumentation for Cultural Heritage

Machine learning and model compression for FPGA/SoC

Antigua Guatemala, June 2025

Romina Soledad Molina, Ph.D.