Quantum machine learning classifiers on FPGA for ultra-low latency applications

L.Borella*, *A.Coppi*, *J.Pazzini*, *A.Stanco*, *A.Triossi*, *M.Zanetti*

University of Padua, Italy

19 July 2024, ICHEP, Prague

Università degli Studi di Padova

ICSC Centro Nazionale di Ricerca in HPC, Big Data and Quantum Computing

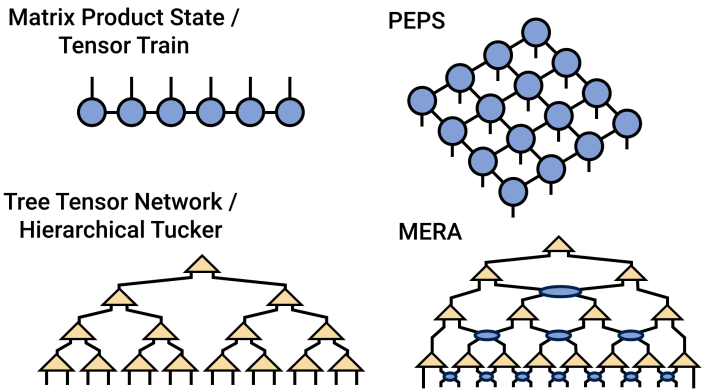
QUANTUM FRONTIERS

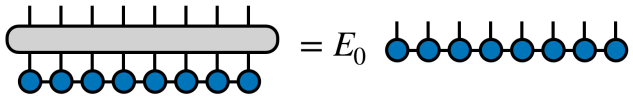
This work is partially supported by ICSC, Centro Nazionale di Ricerca in High Performance Computing, Big Data and Quantum Computing, funded by European Union, NextGenerationEU.

* *lorenzo.borella.1@phd.unipd.it*

Tree Tensor Networks: machine learning

Tensor network methods were born to represent wavefunctions $|\psi\rangle$ and hamiltonians H of many-body quantum systems on classical computers. They consist of the factorization of very high-order tensors into networks of smaller tensors, specifically built to avoid the **curse of dimensionality** [1].



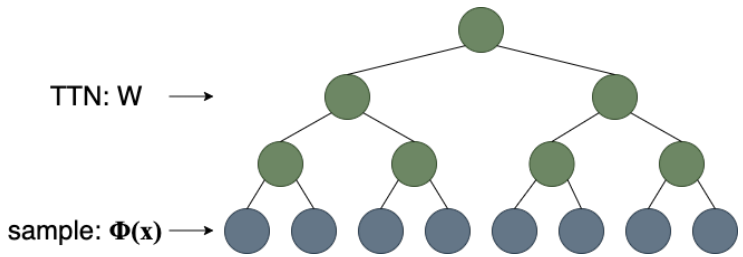


They are typically exploited in energy minimization algorithms or for time evolution simulations, but they can also be exploited in several **Machine Learning** contexts.

For example **Tree Tensor Networks** (TTNs), originally devised to represent quantum entangled states, can be trained as ML classifiers.

By mapping the features of a classic dataset into a higher dimensional space, we can represent each sample as a separable quantum state $\Phi(x)$.

With a supervised learning approach, we can teach the TTN how to properly classify each sample $\Phi(x)$ following the **decision function** $f(x) = W \cdot \Phi(x)$.



Why quantum objects for ML?

•Bond dimension

Tensor networks can be **optimized during training** to reduce the number of parameters. The size of the inner links of the network can be reduced with SVD.

→ **Less parameters!**

•Quantum correlation

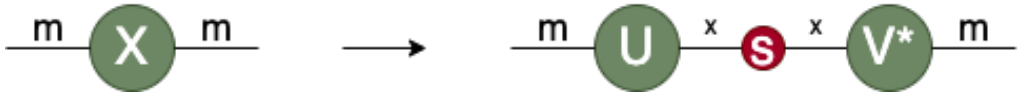
These measurements allow to remove redundancies: if two features are **strongly (anti-)correlated** they convey the same type of information and one of them can be discarded.

→ **Less features!**

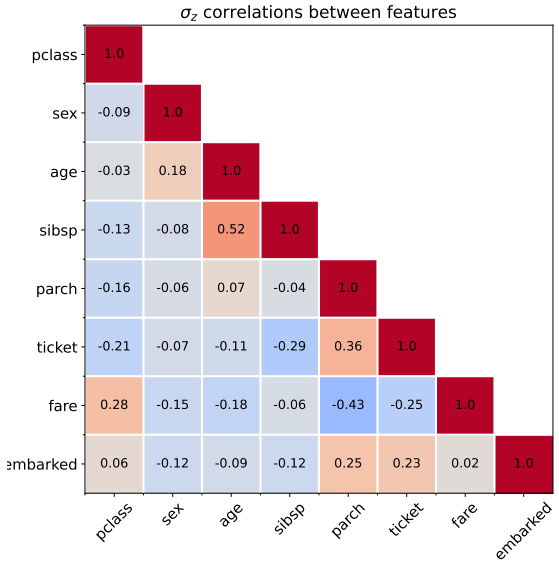
•Von Neumann Entropy

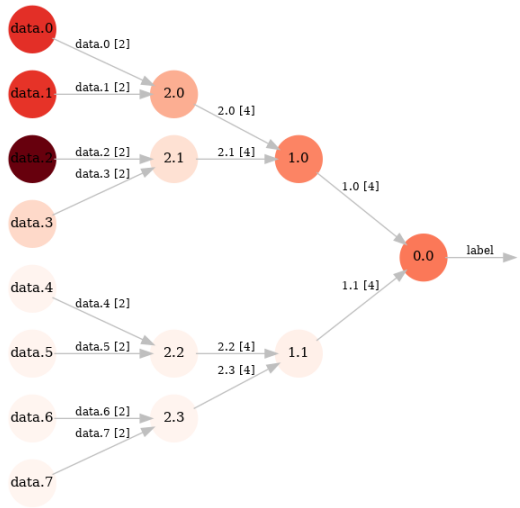
TTN bipartitions enable Schmidt decomposition: entropy measurements represent the **relevance of the information** encoded in each bipartition with respect to the classification task.

→ **Less branches!**



Singular Value Decomposition in TN notation.





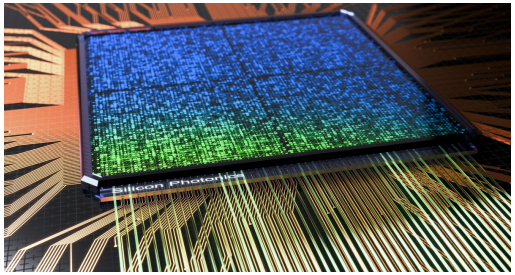
$$C_{i,j} = \frac{\langle\psi|\,\sigma_i^z \sigma_j^z\,|\psi\rangle}{\langle\psi|\psi\rangle}$$

$$|\psi\rangle = \sum_{\alpha}^{\chi} \lambda_{\alpha}^{L,R}\,|\psi\rangle^{L} \times |\psi\rangle^{R}$$

$$S = -\sum_{\alpha} \lambda_{\alpha}^{2} \ln \lambda_{\alpha}^{2}$$

Quantum ML for HEP

- TTNs are good candidates to be deployed in frameworks where **resource saving** is a crucial prerequisite.
- Only **linear operations** are involved when dealing with TTNs, without losing representation capacity.
- **FPGAs** are naturally good at performing fast parallel computations like matrix multiplication and tensor contraction.



The combination of **quantum-inspired networks** and **programmable logic** allows to produce a system able to make classification predictions in a **ultra-low latency** environment, which can be deployed in the **Trigger pipeline** of several HEP experiments.

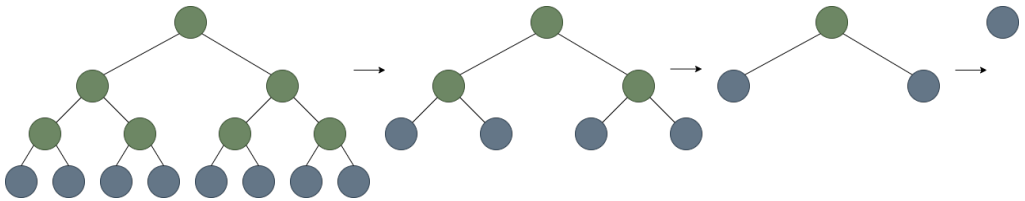
Project outline:

- **Task**: binary classification. The scalar output is the probability of belonging to a specific class.

$$P_i = \frac{|\langle \Phi(x)|\psi_i\rangle|^2}{\sum_i |\langle \Phi(x)|\psi_i\rangle|^2}$$

- **Dataset**: different data with increasing task complexity. Classic ML datasets Iris[2] and Titanic[3] for benchmarking and LHCb OpenData for b/anti-b flavor tagging [4].
- **Architectures**: three TTNs with 4, 8, 16 input features, different sets of hyperparameters, and increasing sizes.
- **Training**: performed in software with local tensor update algorithms to avoid barren plateau[5].
- **Inference**: hardware implementation of full TTN contraction with different degrees of parallelization.

Inference and tensor contraction



Inference: Full TTN contraction with data sample.

Tensor contraction is the basic operation that needs to be implemented.

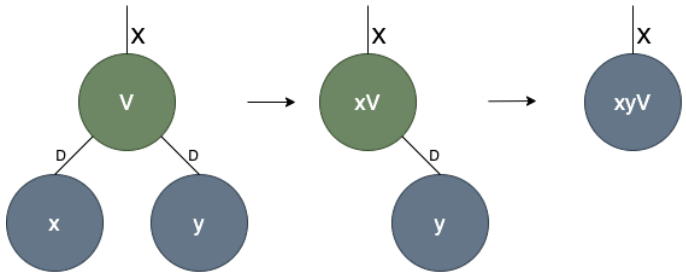
$$z_i = \sum_{j=1}^{D} \sum_{k=1}^{D} V_{ijk} x_j y_k$$

We construct it by directly instantiating **DSPs**, splitting the three factors multiplication into different stages:

1: cartesian product between x and y .
2: multiply results of **1** by the corresponding weights V
3: parallel sums to compute final vector components.

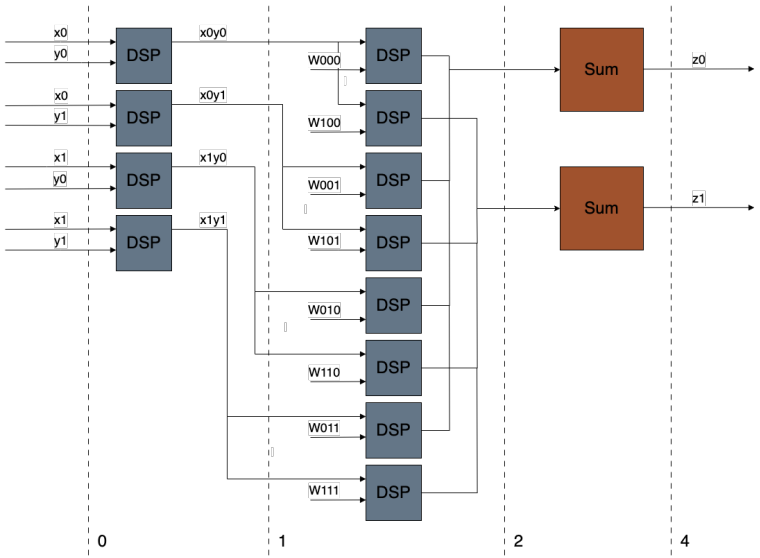
Inference in hardware

- FPGA programmed with **architecture-specific firmware**. Hyperparameters are fixed after implementation.
- Software-trained **weights** are written on static blocks of **RAM**.
- The **feature map** is implemented in hardware with Look Up Tables (**LUTs**).
- Raw data are streamed in input and corresponding the output is returned to host PC.



To explore several combinations of **resource usage** and **latency**, we implemented the operation with two different **degrees of parallelization**. The **Full Parallel** approach maximize the resources and minimize latency, while the **Partial Parallel** approach reduces the number of DSPs, resulting in an increase in latency.

Tensor contraction: full parallel



Block diagram example for D=2, X=2.

Resources: $O(X^3)$

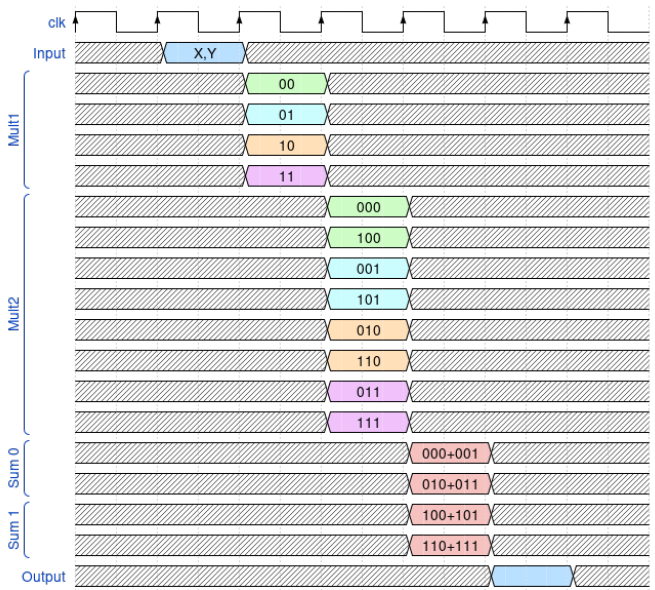
D^2 at multiplication **1** to compute xy cartesian product.
XD^2 at multiplication **2** to multiply each D^2 result of **1** by the X weights factors.
Parallel sums at **3** obtained with **adder trees**.

$$DSP = \sum^{L} \frac{N}{2^i} X_{i-1}^2 (X_i + 1)$$

Latency: $O(\log_2 X^2)$

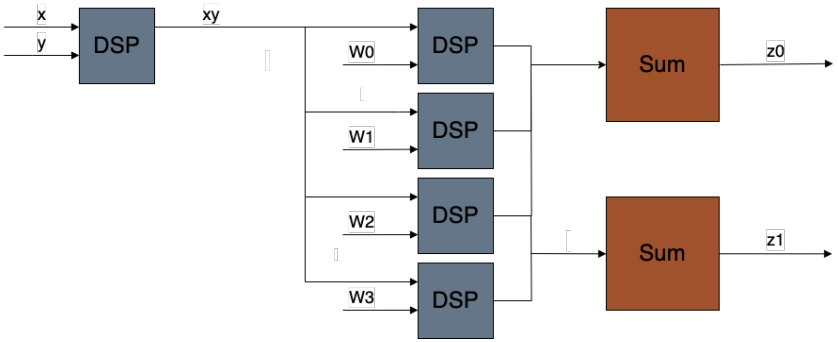
Parallel computation, fully pipelined.
Variable **DSP latency**, tunable with number of internal register Δt_{DSP}.
Adder tree latency scaling as $\log_2 D^2$.

$$\Delta T = \Delta t_{DSP} \sum_{i=1}^{L} 2 + \log_2(X_{i-1}^2)$$



Wave diagram example for D=2, X=2.

Tensor contraction: partial parallel



Contraction block diagram for D=2 and X=2.

Latency: $O(X^2)$

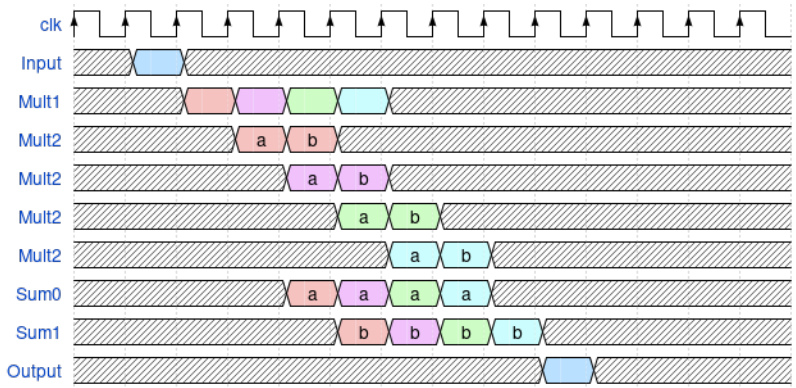
Serial computation, fully pipelined.
Variable **DSP latency**, tunable with number of internal register Δt_{DSP}.
Latency becomes polynomial in X.

$$\Delta T = \Delta t_{DSP} \sum_{i=1}^{L} X_{i-1}^2 + X_i + 1$$

Resources: $O(X^2)$

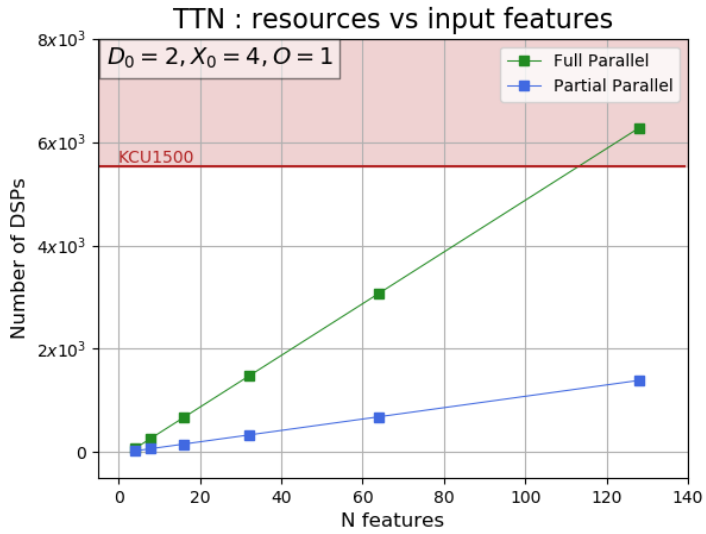
1 **DSP** at multiplication **1** to compute x and y cartesian product.
D^2 at multiplication **2**, where each result of **1** if multiplied by the X weights factors.
X serial sums at **3**.

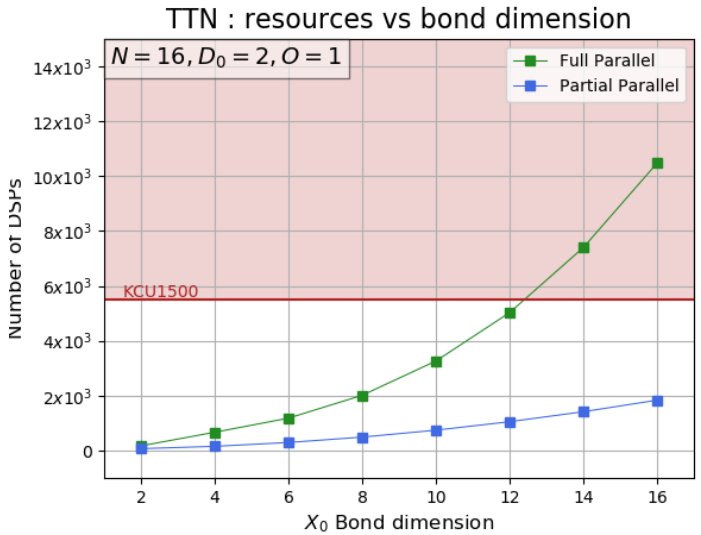
$$DSP = \sum_{i=1}^{L} \frac{N}{2^i}(X_{i-1}^2 + 1)$$

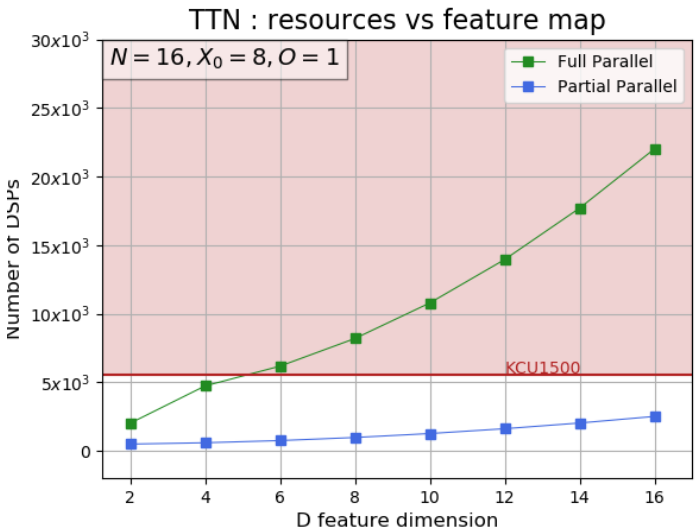


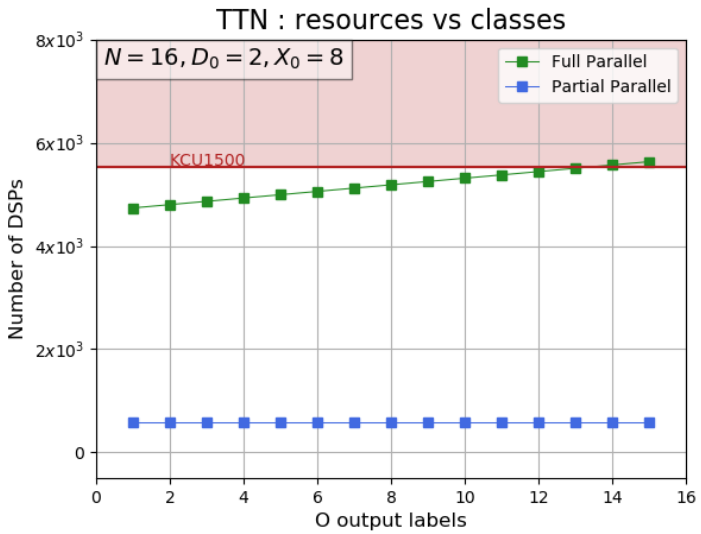
Wave diagram for D=2, X=2.

Resources

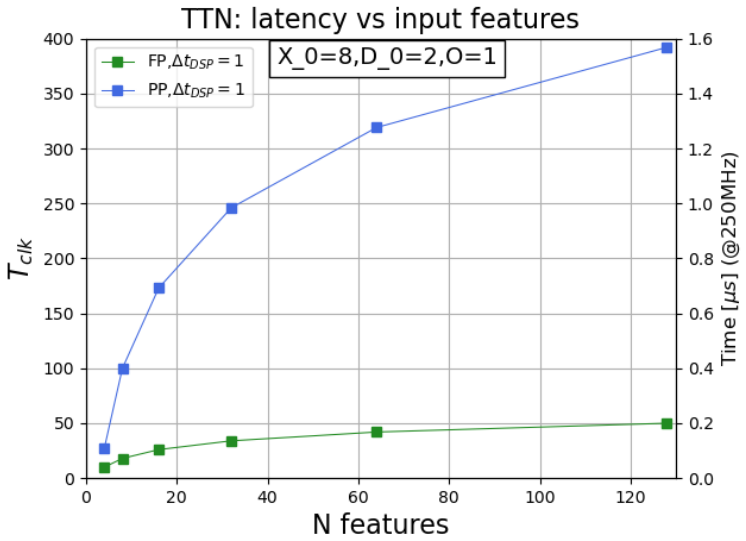

TTN : resources vs input features
$D_0 = 2, X_0 = 4, O = 1$
Full Parallel
Partial Parallel
KCU1500
Number of DSPs
N features

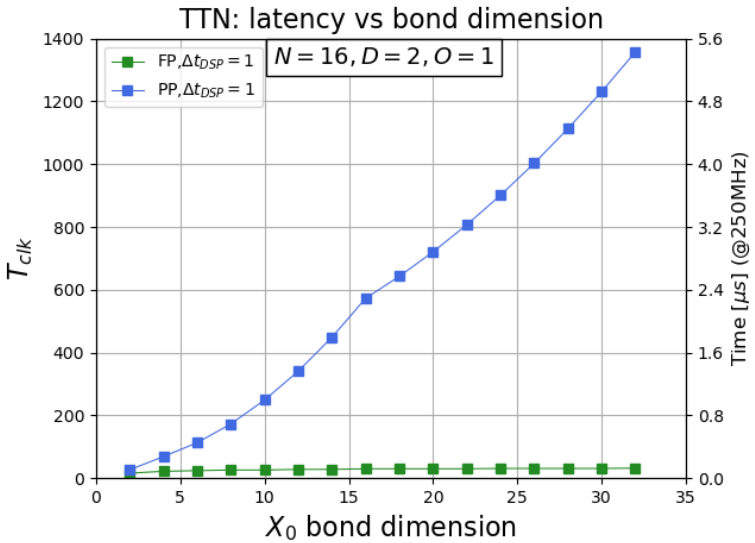

TTN : resources vs bond dimension
$N = 16, D_0 = 2, O = 1$
Full Parallel
Partial Parallel
KCU1500
Number of DSPs
X_0 Bond dimension

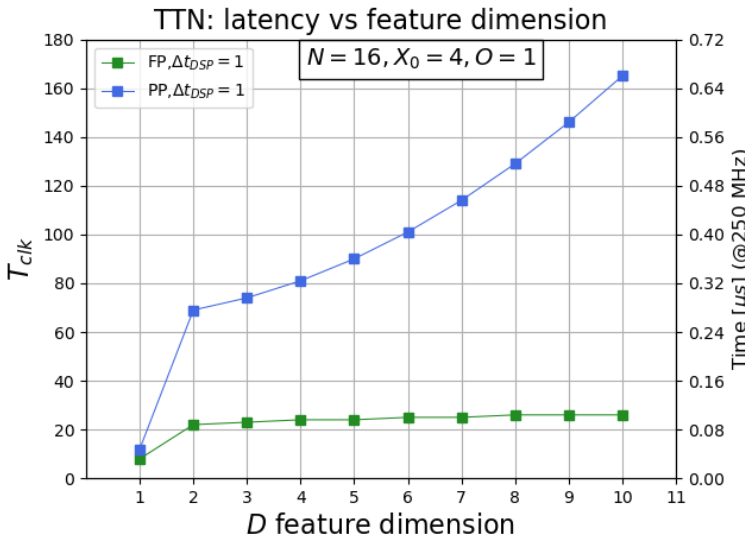

TTN : resources vs feature map
$N = 16, X_0 = 8, O = 1$
Full Parallel
Partial Parallel
KCU1500
Number of DSPs
D feature dimension

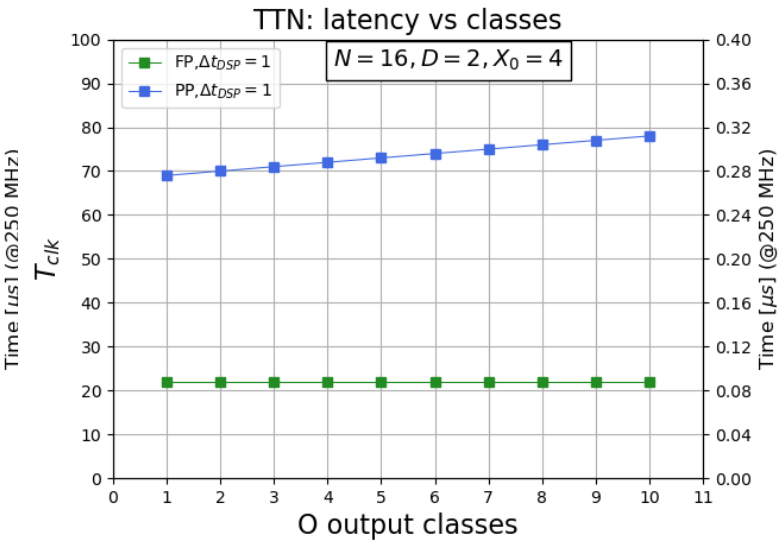

TTN : resources vs classes
$N = 16, D_0 = 2, X_0 = 8$
Full Parallel
Partial Parallel
KCU1500
Number of DSPs
O output labels

Latency

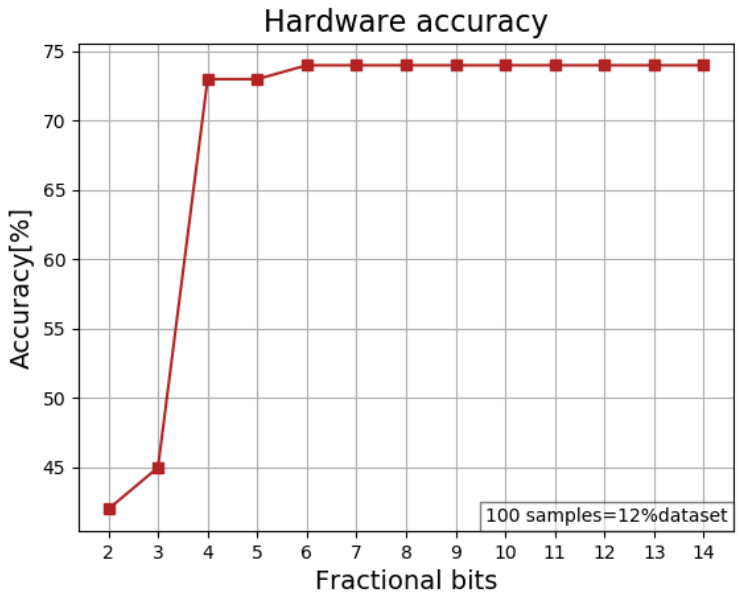

TTN: latency vs input features
X_0=8,D_0=2,O=1
FP,$\Delta t_{DSP}=1$
PP,$\Delta t_{DSP}=1$
T_{clk}
Time [μs] (@250MHz)
N features


TTN: latency vs bond dimension
$N=16, D=2, O=1$
FP,$\Delta t_{DSP}=1$
PP,$\Delta t_{DSP}=1$
T_{clk}
Time [μs] (@250MHz)
X_0 bond dimension

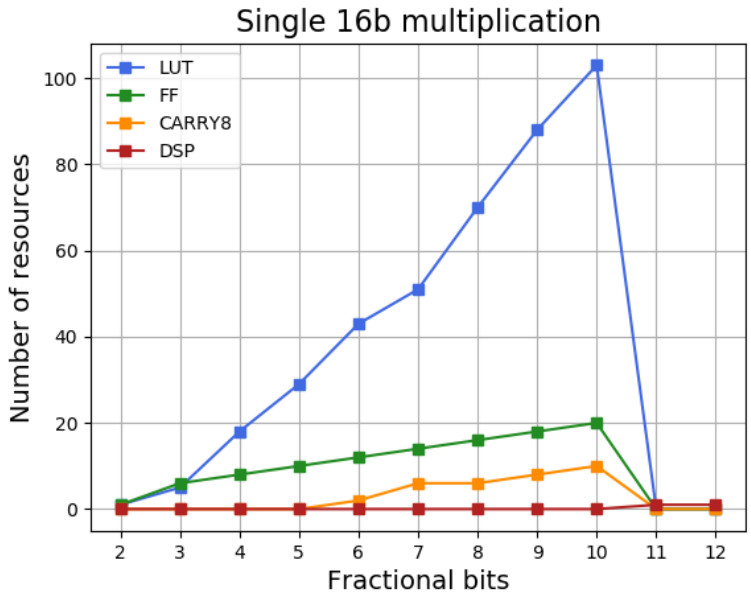

TTN: latency vs feature dimension
$N=16, X_0=4, O=1$
FP,$\Delta t_{DSP}=1$
PP,$\Delta t_{DSP}=1$
T_{clk}
Time [μs] (@250 MHz)
D feature dimension


TTN: latency vs classes
$N=16, D=2, X_0=4$
FP,$\Delta t_{DSP}=1$
PP,$\Delta t_{DSP}=1$
T_{clk}
Time [μs] (@250 MHz)
O output classes

Quantization



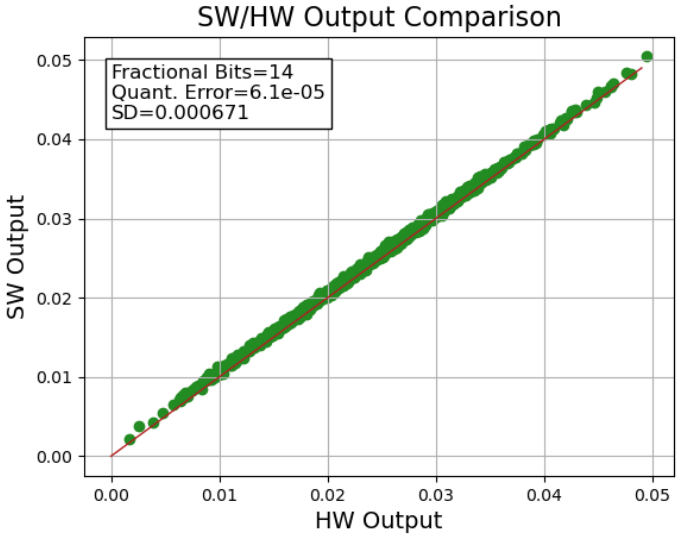
Accuracy vs quantization for TTN with N=8.



Resources inferred for single multiplication on **KCU1500**.

- The numeric accuracy is **architecture-specific**; we might not need 16 bits fixed-point numbers.
- For some architectures, we can devote fewer bits for the fractional part **without losing** classification **accuracy**.
- DSP usage for multiplication is **hardware-specific**: multiplications between fewer bits can be done with **LUTs**.
- Different resources and latency optimizations are possible if high numeric precision is not required.

Validation

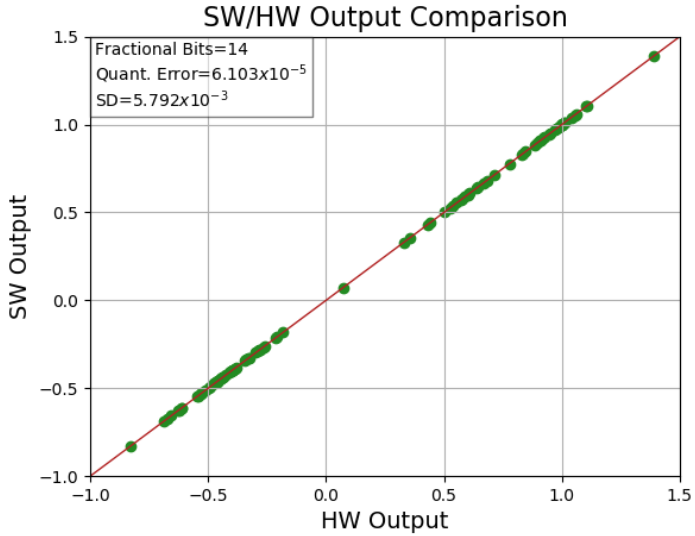


TTN with **N=16 features**, validated on **500 samples**.

Max **quantization precision**: 14 bits for the fractional part, LSB corresponding to $3.1x10^{-5}$.

Outputs in range [0,0.05]: distribution of **HW-SW** values with $\sigma = 6.71x10^{-4}$ corresponding to 4 LSBs.

SW accuracy: 63%. HW accuracy drops with less than 10 bits for the fractional part.



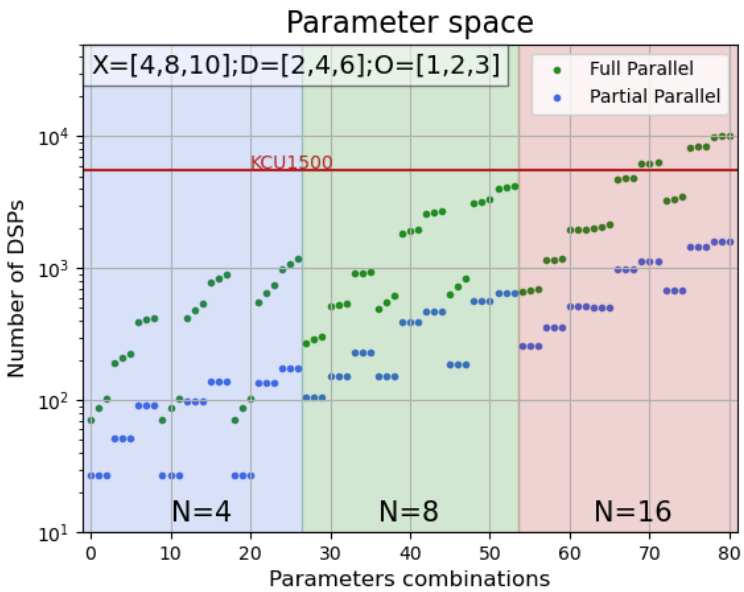
TTN with **N=8 features**, validated on **100 samples**.

Max **quantization precision**: 14 bits for the fractional part, LSB corresponding to $3.1x10^{-5}$.

Outputs in range [-1,1.5]: distribution of **HW-SW** values with $\sigma = 5.792x10^{-3}$ corresponding to 7 LSBs.

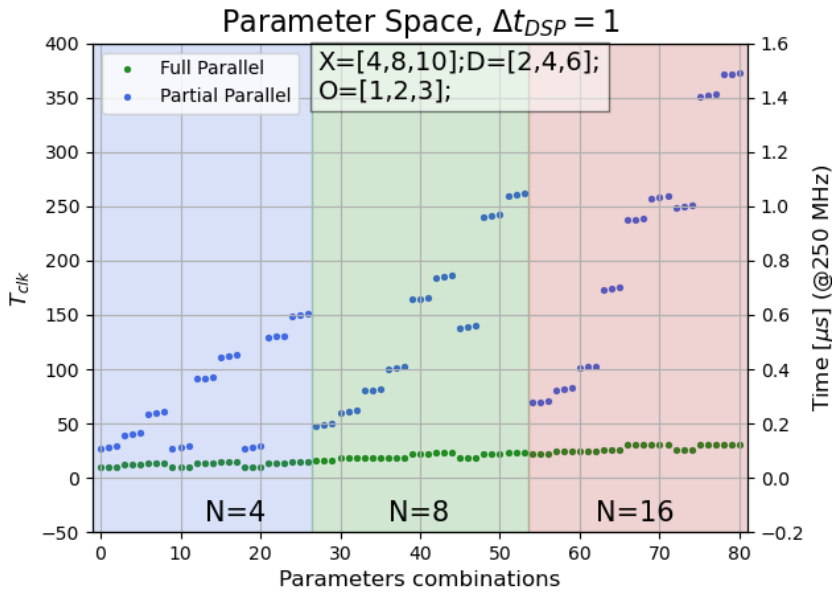
SW accuracy: 75%. HW accuracy drops with less than 4 bits for the fractional part.

Conclusions



- Firmware for **hardware inference** with different degrees of parallelization.
- Validated **ultra-low latency classifier**.

- **Tree Tensor Networks** as quantum machine learning classifiers for **physics**.
- **Projection of resources and latency** for different architectures.



Thank you for your attention!

References I

[1] E. Miles Stoudenmire and David J. Schwab.
Supervised learning with quantum-inspired tensor networks.
arXiv, 2017.
URL: https://arxiv.org/abs/1605.05775.

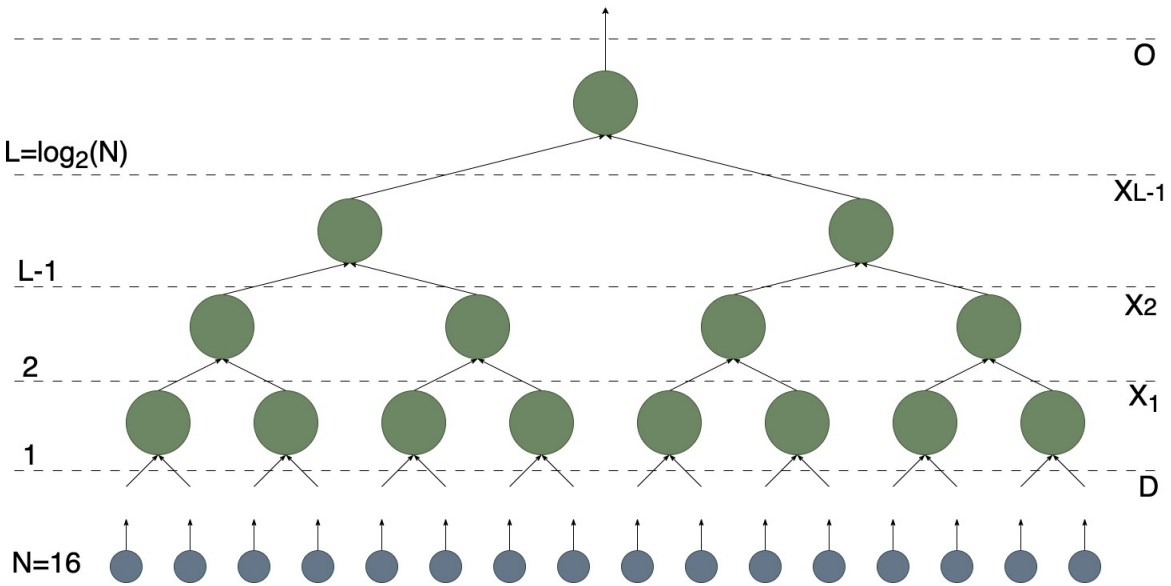
[2] Iris dataset.
https://www.kaggle.com/datasets/uciml/iris.

[3] Titanic dataset.
https://www.kaggle.com/c/titanic/data.

[4] A. Giannelle D. Zuliani T. Felser D.Lucchesi S.Montangero M. Trenti, L. Sestini.
Quantum-inspired machine learning on high-energy physics data.
Nature, 2021.
doi:10.1038/s41534-021-00443-w.

[5] Boixo S. Smelyanskiy V.N. et al. McClean, J.R.
Barren plateaus in quantum neural network training landscapes.
Nat Commun 9, 4812, 2018.
doi:10.1038/s41467-018-07090-4.

TTN architectures



Generic TTN architecture with $N = 16$ features.

Hyperparameters

N input features as **D**-dim vectors according to local feature map:

$$\Phi(x_i) = \begin{bmatrix} \sin\left(\frac{\pi x_i}{2}\right) \\ \cos\left(\frac{\pi x_i}{2}\right) \end{bmatrix}$$

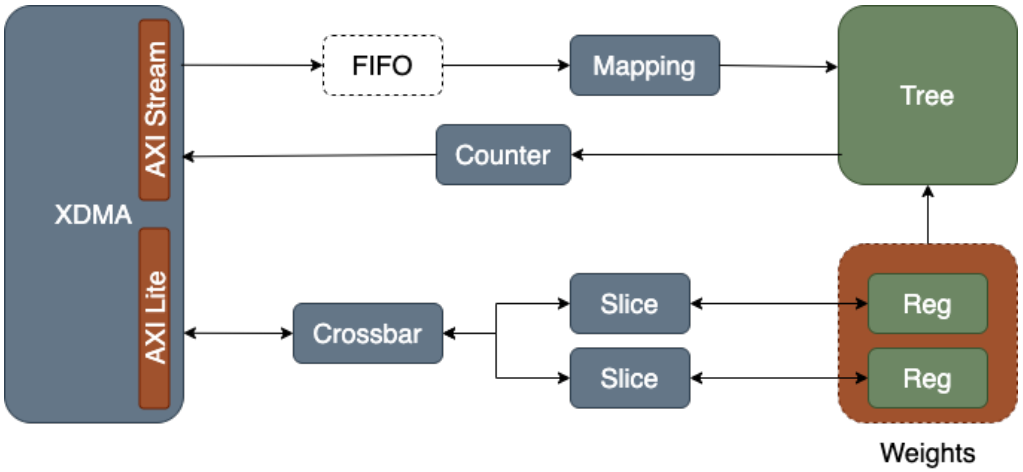
Binary trees: $L = \log_2 N$ layers, with bond dimension scaling as $X_i = min(X_0, D^{2^i})$.

Output dimension $O = 1$ for binary classification, contraction result is a scalar.

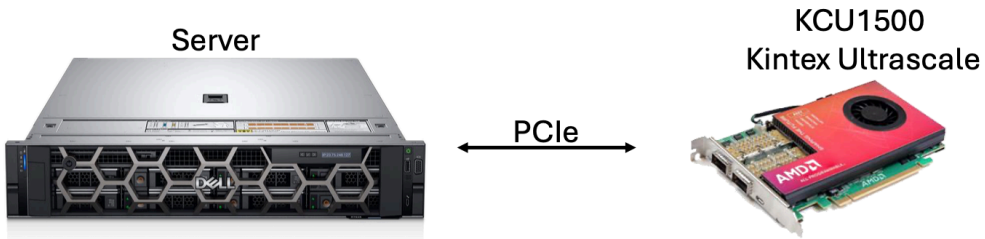
Limitations

- Restricted **amount of resources** on FPGA (DSP, LUT, FF, BRAM, I/O) and tight **physical constraints**. The number of architectures that can be implemented is limited.
- **Latency** constraints must be respected for systems deployed in the Trigger pipeline of HEP experiments. High rate of input data.
- **Real numbers** in hardware: fixed-point precision with **16b**, limited representation range.

Firmware



Full firmware block diagram.



The FPGA is programmed with **architecture-specific** firmware. The TTN **hyperparameters** are **fixed** after implementation.

The values of the **weights** are written on configurable **register blocks** via **AXI Lite** interface. Possibility of testing different networks and quantizations.

Input data are sent to the FPGA with **AXI Stream** protocol; different timings needed for FP and PP implementations.

Project developed on **KCU 1500 Kintex Ultrascale** board, plugged in a Host PC with **PCIe** communication.

The firmware runs with the **AXI Stream** clock fixed at **250 MHz**, but the Out Of Context (OOC) implementation of the TTN can reach up to **500 MHz**.