Tree Tensor Network implemented on FPGA as ultra-low latency binary classifiers.

Lorenzo Borella,[a,b,c,*] **Alberto Coppi,**[b] **Jacopo Pazzini,**[a,b,c] **Andrea Stanco,**[a,d,e] **Andrea Triossi**[a,b,c] **and Marco Zanetti**[a,b,c]

[a] *Istituto Nazionale di Fisica Nucleare (INFN), Via Francesco Marzolo 8, 35131 Padova, IT*

[b] *Dipartimento di Fisica e Astronomia "Galileo Galilei", University of Padua, Via Francesco Marzolo 8, 35131 Padova, IT*

[c] *CERN, Esplanade des Particules 1/1211, 23 Genève, CH*

[d] *Dipartimento di Ingegneria dell'Informazione (DEI), University of Padua, Via Giovanni Gradenigo 6b, 35131 Padova, IT*

[e] *Padua Quantum Technology Research Center, University of Padua, Padova, IT*

E-mail: lorenzo.borella.1@phd.unipd.it

Tensor Networks (TNs) are a computational framework traditionally used to model quantum many-body systems. Recent research has demonstrated that TNs can also be effectively applied to Machine Learning (ML) tasks, producing results comparable to conventional supervised learning methods. In this work, we investigate the use of Tree Tensor Networks (TTNs) for high-frequency real-time applications by harnessing the low-latency capabilities of Field-Programmable Gate Arrays (FPGAs). We present the implementation of TTN classifiers on FPGA hardware, optimized for performing inference on classical ML benchmarking datasets. Various degrees of parallelization are explored to evaluate the trade-offs between resource utilization and algorithm latency. By deploying these TTNs on a hardware accelerator and utilizing an FPGA integrated into a server, we fully offload the TTN inference process, demonstrating the system's viability for real-time ML applications.

42nd International Conference on High Energy Physics (ICHEP2024)
18-24 July 2024
Prague, Czech Republic

[*]Speaker

1. Introduction

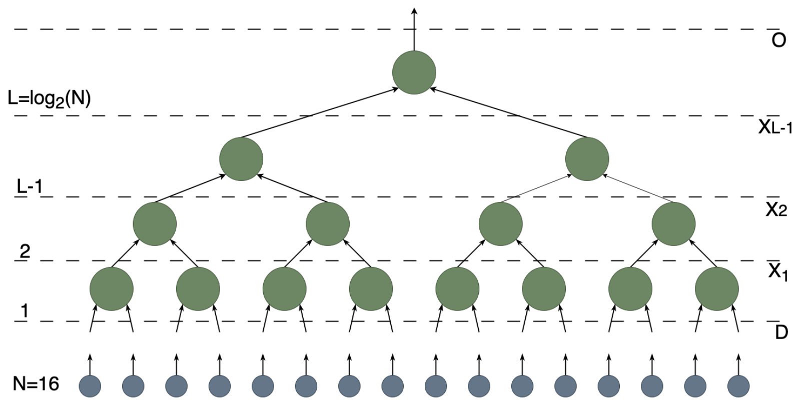

Figure 1: Generic representation of Tree Tensor Network architecture (green tensors), with 16 input features (blue tensors) and variable set of hyperparameters $[D, X_1, X_2, X_{L-1}, O]$

Tensor Network (TN) methods are widely used to represent and simulate many-body quantum systems on classical computers. These methods involve factorizing high-order tensors into networks of smaller tensors, thus overcoming the curse of dimensionality [1]. Tree Tensor Networks (TTNs), a specific application of this approach, were originally designed to represent weakly entangled states but have also been successfully applied to standard Machine Learning (ML) tasks [2, 8, 9]. In this context, the features of a classical ML dataset can be mapped into a higher-dimensional space, where each sample is represented as a separable quantum state. Through supervised learning, the entire dataset can then be encoded into a TTN architecture.

TTNs offer several advantages that make them highly effective. One key benefit is the ability to optimize bond dimensions during training, reducing the overall number of parameters through Singular Value Decomposition (SVD) without losing critical information. This results in flexible, compressible architectures that require fewer parameters while maintaining accuracy. Additionally, TTNs, being quantum-inspired, allow the measurement of quantum correlations between dataset features after training. This makes it possible to remove redundancies, potentially further compressing the network and helping to identify which features are essential and which can be discarded. Another powerful feature is the ability to measure the Von Neumann entropy across different bi-partitions of the network, offering insights into where the information is concentrated and which features are most relevant for the ML task [7]. Combining quantum correlations with entropy analysis enables ranking of the most important features while minimizing the network size, making TTNs highly efficient for machine learning tasks [3].

These characteristics make TTNs strong candidates for applications where resource efficiency is critical. Their inherently linear operations make them well-suited for hardware implementation on Field Programmable Gate Arrays (FPGAs), which are highly optimized for fast parallel computations. By combining quantum-inspired networks with programmable logic, it is possible to develop systems capable of making predictions on input data in low-latency environments. This is particularly valuable in the Trigger pipeline of high-energy physics (HEP) experiments, where rapid decisions must be made to retain or discard potentially important physics data.

This work focuses on binary classification using TTNs, exploring different architectures and hyperparameters. The networks are first trained in software, and once the desired accuracy is achieved, the inference is deployed on hardware. This demonstrates the feasibility of deploying TTN-based predictors on FPGAs in ultra-low-latency environments.

2. Implementation

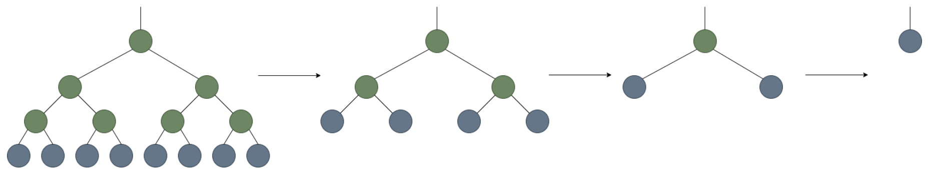

Figure 2: Full TTN contraction obtained by repeating the single node contraction for all the tree layers.

To perform inference with Tree Tensor Networks (TTNs) on hardware, the primary operation that needs to be implemented is the single-node contraction. This operation is mathematically described by the following equation:

$$z_i = \sum_j \sum_k x_j y_k V_{ijk} \tag{1}$$

In this operation, x and y are input vectors, and V is a rank-three tensor within the TTN. The result, z_i, is also a vector, and the full contraction of the TTN is accomplished by applying this process iteratively across all layers of the network (Fig.2). For binary classification tasks, the final output of the inference is typically a 2-dimensional vector representing the probabilities that the input belongs to each class.

To perform these arithmetic operations on an FPGA, we utilize Digital Signal Processors (DSPs), which are designed to handle basic arithmetic operations, including multiplications, as well as more complex computations [6]. In order to balance between resource usage and latency, two distinct firmware implementations are developed to handle the full contraction with different levels of parallelization. These approaches are referred to as the Full Parallel (FP) and Partial Parallel (PP) implementations.

The Full Parallel (FP) implementation is optimized to minimize the latency of the inference algorithm by fully utilizing the available resources. In this approach, DSPs are deployed to handle as many parallel operations as possible, reducing the time required to complete the contraction. As a result, this method maximizes resource consumption but minimizes the overall inference time. The DSP usage and latency for any TTN architecture using the FP approach can be predicted using the following equations:

$$DSP_{FP} = \sum_{l=1}^{L} \chi_{l-1}^2 (\chi_l + 1) \frac{N}{2^l}$$

$$LAT_{FP} = \Delta t_{DSP} \sum_{l=1}^{L} 2 + \log_2(\chi_{l-1}^2)$$

In contrast, the Partial Parallel (PP) implementation reduces the number of DSPs used for the multiplications within each node, resulting in a more resource-efficient design but with an inherent increase in latency. This approach carefully distributes the workload across fewer DSPs, meaning that the inference process takes more clock cycles to complete. The resource consumption and number of clock cycles required for the PP implementation can be computed with the following formulae:

$$DSP_{PP} = \sum_{l=1}^{L} (\chi_{l-1}^2 + 1) \frac{N}{2^l}$$

$$LAT_{PP} = \Delta t_{DSP} \sum_{l=1}^{L} \chi_{l-1}^2 + \chi_l + 1 \quad \text{if} \quad \chi_{l-1} \leq \chi_l$$

By defining both the FP and PP implementations (as shown in Fig.3(left)), these equations allow us to predict the number of DSPs and clock cycles required for any combination of TTN hyperparameters [D,X_1,X_2,X_{L-1},O]. In the projection plots, red lines indicate the hardware limits for the number of DSPs available on the XCKU1500 FPGA, as well as the 1 microsecond threshold latency when running the firmware at a 500 MHz clock speed. By adjusting the level of parallelization, these implementations offer a flexible solution to optimize both latency and resource consumption, making it possible to tailor the TTN inference process to the constraints of a specific hardware platform. This allows for efficient real-time processing, particularly in environments such as high-energy physics experiments, where rapid decision-making is critical.

3. Results

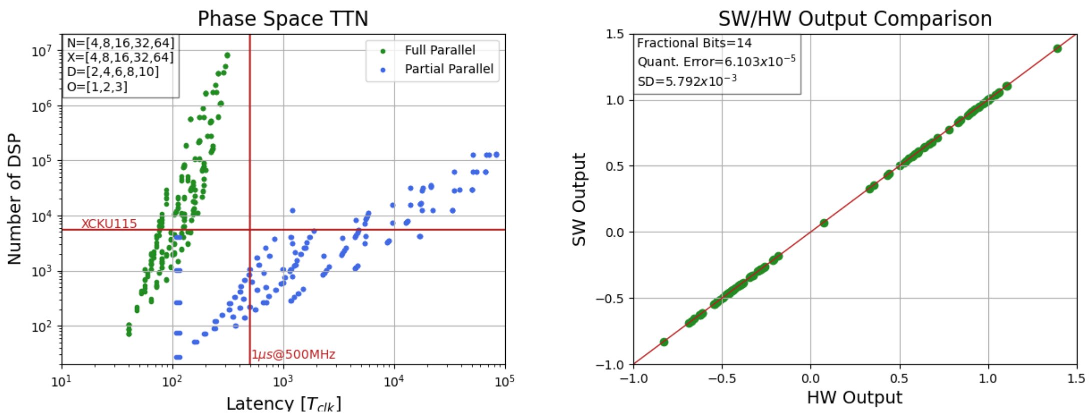

Figure 3: (left) Resources vs latency space filled with 375 hyperparameter combinations, represented both with FP and PP implementations. (right) Software and hardware output values comparison.

The functionality of the hardware inference is tested using a Kintex Ultrascale board connected to a server, which handles communication with the FPGA and manages the input/output data exchange. For the development of the TTN inference algorithm on FPGA, the network topology is predefined, and the board is programmed with firmware that includes the fixed hyperparameters of the network architecture. The trained tensor values are stored in block RAM (BRAM), equipping the logic with a series of registers that the host PC can access via the AXI Lite protocol [4]. These registers are mapped to the individual weights of the TTN. The inference process is executed by sending a stream of input data to the FPGA and retrieving the corresponding output values generated by the TTN, using the AXI Stream protocol to facilitate communication between the host and FPGA [5].

The entire inference process on the FPGA is validated by comparing the hardware-generated output values with those obtained from the software implementation, using the same subset of classification data. As shown in Fig.3(right), quantization of the software values introduces an error of $6.103 \cdot 10^{-5}$ in the hardware representation of the input samples and tree tensor weights. Despite this, the hardware classification values closely match the software results, with a standard deviation of $5.792 \cdot 10^{-3}$, ensuring accurate classification with identical labels to the software.

In conclusion, this work demonstrates the viability of leveraging quantum-inspired networks to solve binary classification tasks, with successful implementation and validation on FPGA hardware. The study confirms the possible deployment of these networks in the Trigger pipelines of high-energy physics (HEP) experiments, especially given their ultra-low latency performance, with sub-microsecond response times. This opens up promising possibilities for resource-efficient, high-speed classification in real-time systems.

References

[1] Evenbly, G., Vidal. *Tensor Network States and Geometry*. J Stat Phys 145, 891–918 (2011). https://doi.org/10.1007/s10955-011-0237-4

[2] E. Miles Stoudenmire and David J. Schwab, David J, *Supervised Learning with Tensor Networks*. arXiv:1605.05775v2 (2017) https://doi.org/10.48550/arXiv.1605.05775

[3] Felser, Trenti, Sestini, Gianelle, Zuliani, Lucchesi and Montangero.*Quantum-inspired machine learning on high-energy physics data*.Publisher: Sissa Medialab Place: Trieste, Italy (2021). https://doi.org/10.1038/s41534-021-00443-w

[4] *AXI4-Lite Interface*. Real Digital, computer engineering education, 2024.https://www.realdigital.org/doc/a9fee931f7a172423e1ba73f66ca4081

[5] *AXI4-Stream Interfaces*. AMD, 2024.https://docs.amd.com/r/en-US/ug1399-vitis-hls/AXI4-Stream-Interfaces

[6] Ibrahim, Dogan and Davies, Anthony. *The Evolution of Digital Signal Processors*. 6th IEEE History of Electrotechnology Conference, 2019.10.1109/HISTELCON47851.2019.9040130

[7] Okunishi, Kouichi and Ueda, Hiroshi and Nishino, Tomotoshi.*Entanglement bipartitioning and tree tensor networks*. Progress of Theoretical and Experimental Physics, 2023. https://doi.org/10.1093/ptep/ptad018

[8] Richik Sengupta and Soumik Adhikary and Ivan Oseledets and Jacob Biamonte. *Tensor networks in machine learning*. arXiv, 2022. https://arxiv.org/abs/2207.02851

[9] Hao Chen and Thomas Barthel. *Machine learning with tree tensor networks, CP rank constraints, and tensor dropout*. arXiv, 2023. https://arxiv.org/abs/2305.19440