Accelerating Transformer Neural Networks on FPGAs for High Energy Physics Experiments

Filip Wojcicki*, Zhiqiang Que*§, Alexander D Tapper†, Wayne Luk*
*Department of Computing, Imperial College London, UK, {faw18, z.que, w.luk}@imperial.ac.uk
† Department of Physics, Imperial College London, UK, a.tapper@imperial.ac.uk

***Abstract*—High Energy Physics studies the fundamental forces and elementary particles of the Universe. With the unprecedented scale of experiments comes the challenge of accurate, ultra-low latency decision-making. Transformer Neural Networks (TNNs) have been proven to accomplish cutting-edge accuracy in classification for hadronic jet tagging. Nevertheless, software-centered solutions targeting CPUs and GPUs lack the inference speed required for real-time particle triggers, most notably those at the CERN Large Hadron Collider. This paper proposes a novel TNN-based architecture, efficiently mapped to Field-Programmable Gate Arrays, that outperforms GPU inference capabilities involving state-of-the-art neural network models by approximately 1000 times while preserving comparable classification accuracy. The design offers high customizability and aims to bridge the gap between hardware and software development by using High-Level Synthesis. Moreover, we propose a novel model-independent post-training quantization search algorithm that works in general hardware environments according to user-defined constraints. Experimental evaluation yields a 64% reduction in overall bit-widths with a 2% accuracy loss.**

I. Introduction

The CERN Large Hadron Collider (LHC) is the world's highest-energy particle collider, the main aim of which is the study of proton-proton collisions. The detection and analysis of the data from such collisions is a challenge in various areas. One such area is *jet tagging*, associating observed collimated sprays of composite particles (hadrons) with the particles (typically quarks or gluons) which initiated the jet.

The detectors make use of different sub-detector systems to measure properties of the collisions, such as particle trajectories and energies. These low-level features may be used directly or combined into higher-level features. The latter have been successfully used in the past using more physically motivated machine learning (ML) algorithms, e.g. using computer vision [1]. However, more recently, various deep learning approaches have proven to outperform their predecessors [2, 3].

The information throughput of petabytes per second collected by the LHC detectors outclasses the real-time inference capabilities of typical state-of-the-art solutions. The real-time decision-making is often of utmost importance, and this paper is motivated by this challenge which includes exploring various types of neural network architecture as well as the necessary infrastructure and deployment processes. Recently, the `hls4ml` codesign workflow has been successfully adopted in particle physics experiments [4, 5], which allows ML researchers and physicists to easily deploy their solutions trained using common ML frameworks on reconfigurable devices, typically Field Programmable Gate Arrays (FPGAs), or application specific hardware, vastly improving the detection algorithms' inference capabilities. However, when it comes to neural network architectures that have been proven to outperform the previous state-of-the-art, at the time of writing `hls4ml` offers limited support for graph neural networks [6, 7] and does not work with transformer neural networks.

[§]Corresponding author.

This work develops novel, hardware-aware transformer neural network architectures as well as establishes efficient ways of mapping them onto FPGAs. The novel aspects of this work can be summarized as follows:

- **Hardware-Aware Transformer Network Design:** A novel transformer architecture is designed with hardware in mind, exploiting the capabilities of the target platform to achieve cutting-edge latency and accuracy.
- **Efficient Hardware Implementation:** A codesign workflow with a number of complex neural network layers supporting a range of optimization techniques, that can be generalized to other applications.
- **Post-Training Quantization Search:** A custom algorithm is proposed that finds the optimal set of bit-widths for a High-Level Synthesis (HLS) model according to user-defined constraints.

The specific transformer-based structure and coefficients are specific for LHC, but the requirement for low latency would benefit many other applications, especially those requiring real-time response. For example, low latency designs would benefit adaptive radiotherapy [8], gravitational wave detection [9] and high-frequency trading [10]. The developed approach can be adapted to address these other applications.

II. Background

A. Real-time data processing at the LHC

Future upgrades to the ATLAS and CMS experiments at the LHC are the main motivation for this work. The real-time processing systems envisaged for these detectors are composed of triggers split into two levels [11, 12, 13]:

- **L1T** - hardware (FPGAs) and firmware, pipelined and fixed latency.

- **HLT** - software, using farms of CPUs, with behavior close to offline algorithms.

The Level-1 Trigger (L1T) key specifications used to evaluate the design in this paper are the input data frequency of 40 MHz, which with a pipeline depth of 500 slots results in a fixed 12.5 μs latency budget, and the output frequency to HLT of 750 kHz (both of which correspond to the CMS experiment case, though the ATLAS design is very similar). The L1T receives data directly from custom detector electronic systems over high-speed serial optical links. Data that cannot be processed within the latency budget are completely lost. The High Level Trigger (HLT) relies on thousands of multithreaded CPUs and recently GPUs.

B. Transformer Neural Networks and Attention

A promising architecture which is the topic of this paper is the transformer neural network (TNN). Similarly to recurrent neural networks (RNNs), TNNs were designed for sequential input data, most commonly found in natural language processing applications. However, compared to RNNs, they process all input data at once. In RNNs, fully-connected, convolutional [14] or attention mechanisms [15] are used in a recurrent manner to allow models to learn the representation and connections between different parts of the input sequence, which most commonly are words in a sentence. This limits the parallelizability as the network is handled serially - each hidden state needs to wait for the result generated by the previous one. In TNNs, a modified mechanism called self-attention [16] is used which can find global relations in data, without relying on the temporal, sequencing information. The self-attention combines several simpler operations to achieve its strength, including linear layers, matrix multiplication and the softmax function.

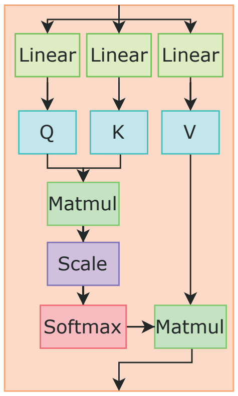

Fig. 1. Block diagram of a self-attention head.

A diagram of the self-attention can be seen in Fig. 1. The Q, K, and V stand for *queries*, *keys*, and *values* respectively which are meant to give a better understanding behind the mechanism's idea. It is also important to note that multiple blocks of self-attention, referred to as *heads*, can be used together, which allows for each head attending information about a different hidden characteristic. The results of all heads are concatenated, increasing output's dimensionality, and multiplied with a weight, as seen in figure 2.

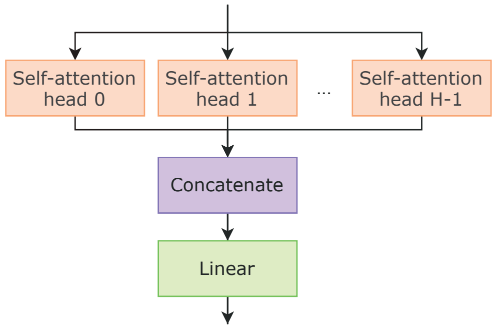

Fig. 2. Illustration of H self-attention heads forming a multi-headed self-attention block.

The complexity of a multi-head self-attention depends on the dimensions of the underlying *queries* $\in \mathbb{R}^{N_Q \times d_Q}$, *keys* $\in \mathbb{R}^{N_K \times d_Q}$, and *values* $\in \mathbb{R}^{N_K \times d_V}$ parameters. The resulting time complexity is represented in equation 1, while the space complexity can be seen in equation 2.

$$\mathcal{O}(h \cdot (N_Q N_K (d_Q + d_V) + d_Q^2 (N_Q + N_K) + d_V^2 N_K + N_Q d_V d_{out}))\quad (1)$$

$$\mathcal{O}(h \cdot (N_Q (N_K + d_V) + d_Q (N_Q + N_K) + d_V N_K + N_Q d_{out}))\quad (2)$$

The symbols have the following meaning: h - number of heads, d_{out} - output dimension, $N_i \times d_i$ with $i \in \{Q, K, V\}$ - dimensions of *queries*, *keys*, and *values* weight matrices.

III. Proposed Neural Network Architecture

A. Model Design

This section addresses the complexity problem of TNNs through a number of approaches. In order to track the accuracy and latency effects of the design changes, the complexity of the initial architecture (seen in Fig. 3) is high, but then it is gradually lowered with various simplification and optimization techniques. A number of optimization steps were tried, and the successful ones are listed below, where the criterion each time was the lack of notable decrease in accuracy:

1) First change was the reduction in the number of transformer layers from 3 to 1 and self-attention latent dimensions from 128 to 16.
2) Although proven to outperform simpler ReLU activation functions [17], the original SiLU blocks were simplified to allow for easier FPGA mapping.
3) Similarly for the normalization layers [18], they are replaced by batch normalization. Furthermore, these layers, along with dropout that also combats over-fitting [19], were removed from the final design, which can be attributed to the change in the used dataset, that already includes normalized data.

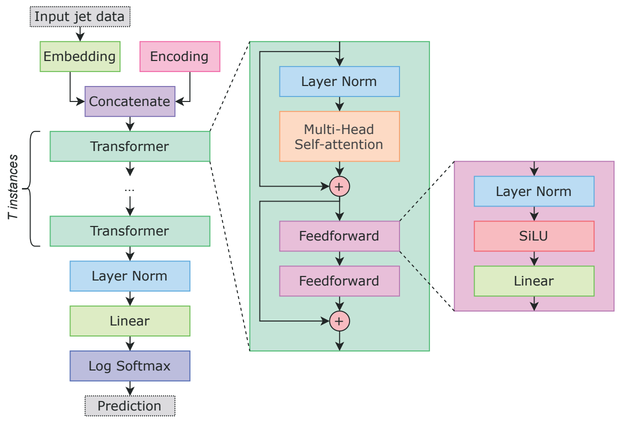

Fig. 3. Flow chart of the original architecture

The resulting simplified design can be seen in Fig. 4, while its FPGA implementation is the topic of the next section. Interestingly, the achieved accuracy is only slightly lower than the starting point, with the value of 76.6%. This suggests that the starting model could be described as 'over-saturated' with learning capacity given the relatively easy to classify HLF dataset.

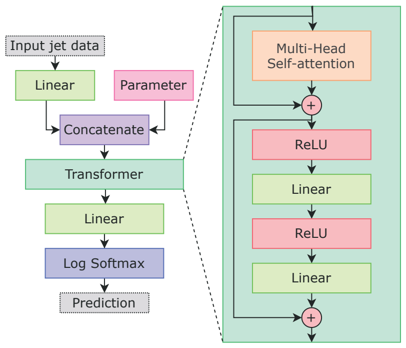

Fig. 4. Flow chart of the simplified architecture

IV. Hardware Mapping and Optimizations

The template for the hardware mapping was obtained using `hls4ml`, however, the bulk of the design (the transformer layer with its self-attention component) was developed independently, since they are not available in the `hls4ml` at the time of this work. During exploration, SiLU and layer normalization blocks were also implemented, but the final design does not depend on them.

The optimization that started the design process was the transition from floating-point to fixed-point arithmetic. It has been proven a successful strategy for a variety of applications [20], especially to meet real-time processing latency constraints [21]. Fixed-point representation offers advantages in terms of operations complexity, and hence latency, due to the lack of cumbersome mantissa and exponent conversions. It is also common to use fewer bits to represent numbers to further simplify computations by adhering to bit widths of FPGA building blocks, mainly Digital Signal Processing (DSP) slices, while also requiring less logic elements for registers and Look-Up Tables (LUT). This work benefits from 15/14 and 6/10 bit splits for the integer and fractional parts accordingly, at different stages in the design, depending on the precision required by an operation. The *matmul* blocks

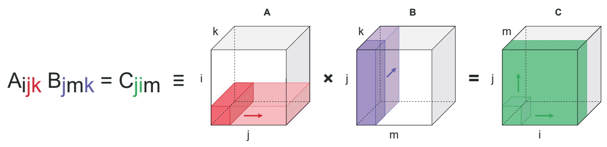

Fig. 5. Visualization of a tensor operation expressed in Einstein Summation notation

in Fig. 1 implement a custom tensor multiplication, which is specific to the self-attention formula [16]. It is normally expressed using Einstein Summation notation [22], which is supported by mathematical and machine learning libraries like `NumPy` or `PyTorch`. However, not present by default in HLS, it required careful design of the calculation loops in order to not cripple the performance by unnecessary computations and pseudo-random data accesses. As part of this research, an efficient and fully-customizable HLS block has been designed, that uses a very similar interface to the Python equivalent. Figure 5 shows a visualization for an example notation to give a better understanding of the necessary flexibility of a formula. Another simple optimization used in between the *matmul* blocks was the change in size scaling from using division ($1/\sqrt{size}$) to right-shifting, which required precomputing the logarithm of the square root of size, vastly simplifying the hardware required at run-time.

The adapted architecture requires finding the log softmax values for the final activation layer, which is used to map inputs to probability-like results. The numerical stability and computational efficiency of this operation is often explored in-depth [23] and varies depending on the programming language and target platform. The naive implementation comes straight from the definition of taking a logarithm of softmax, seen in equation 3.

$$\sigma(x_i) = e^{x_i} / \sum_{j=1}^{N} e^{x_j} \tag{3}$$

This paper proposes a different way of mapping this operation to hardware to improve stability while shortening the critical path and using less resources. It is based on the derivation shown in equation 4.

$$\begin{aligned}\log(\sigma(x_i)) &= log(e^{x_i} / \sum_{j=1}^{N} e^{x_j}) \\ &= \log(e^{x_i}) - \log(\sum_{j=1}^{N} e^{x_j}) \\ &= x_i - \log(\sum_{j=1}^{N} e^{x_j})\end{aligned} \tag{4}$$

The resulting hardware operations are depicted in Fig. 6 and 7. It is important to note that operations like exponentiation, division or taking a logarithm usually rely on precomputing a wide range of values and mapping them in memory or LUTs to allow for lookup at run-time. Hence, the optimized design requires one fewer lookup while also replacing multiplication by a subtraction, which can be simpler to express in hardware.

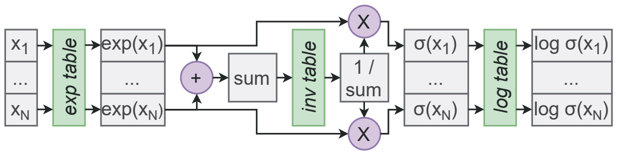

Fig. 6. Direct hardware implementations of log softmax.

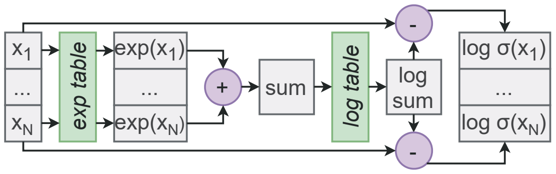

Fig. 7. Optimized hardware implementations of log softmax.

Although further simplifications, including approximating the summation by finding the maximum (see equation 5) or simply omitting the logarithm portion of the expression, were also evaluated, they noticeably lowered the final accuracy and were thus abandoned.

$$\begin{aligned}\log(\sigma(x_i)) &= e^{x_i} - \log(\sum_{j=1}^{N} e^{x_j}) \\ &= e^{x_i} - \sum_{j=1}^{N} \log(e^{x_j}) \\ &= e^{x_i} - \sum_{j=1}^{N} x_j \\ &\approx e^{x_i} - \max(x)\end{aligned} \tag{5}$$

The final and most crucial optimization involved pipelining the design. In order for an HLS design to be pipelined, several other optimizations have to be applied, either automatically (derived from project's constraints) or manually (using HLS `pragma` directives). The most notable ones are loop unrolling, array partitioning, and function inlining, all of which were set to trade-off the hardware resources for the lowest latency possible.

V. Evaluation and Analysis

A. Dataset and Preprocessing

The dataset used in this work consists of simulated 13 TeV energy LHC proton-proton collisions, and it includes information about the High-Level Features (HLF) [24] of particle jest that were constructed using the anti-k_t clustering algorithm [25]. The dataset distinguishes five classes: light quarks, top quarks, W bosons, Z bosons, and gluons.

It is also worth mentioning that normalization is the only preprocessing measure used in this work. The samples are simulated and do not contain any illegal or null values that would require dropping or substituting, and for the same reason, there is also no need for data augmentation.

B. Proposed Neural Network Inference Results

The simplification yields a significant reduction in FLOPS to 583,676 and parameter number to 2,605. Not surprisingly, this leads to a considerably faster inference, which is compared with other state-of-the-art models in Table I and listed for several other computing devices in Table II. The results have been obtained by firstly performing 5 warm-up runs, followed by CUDA-synchronization where applicable, and then timing 200 test runs, for which mean and standard deviation were calculated. On a closer inspection, the results are similar to the aforementioned DNN latency, however there are differences in the evaluation methodology and the test bench specification, which includes an Nvidia GTX 1080 GPU [26], which prevents further conclusions.

TABLE I
Summary of networks' inference time, accuracy, Floating-Point Operations Per Second and parameter number for optimal batch sizes, with best values in bold. Inference times measured using Nvidia GTX 1080 (*), Nvidia GTX 1080 Ti (‡).

Neural network	**Inference per batch (ms)**	**Accuracy / aver. AUC**	**FLOPS**	**Parameters**
DNN [26]	**1.0 ± 0.2***	0.760 / 0.941	**27 k**	14,725
CNN [26]	57.1 ± 0.5*	0.740 / 0.911	400 k	205,525
GRU [26]	23.2 ± 0.6*	0.750 / 0.912	46 k	15,575
JEDI-net [26]	121.2 ± 0.4*	- / **0.959**	116 M	33,625
JEDI-net with $\sum O$ [26]	402.0 ± 1.0*	- / 0.957	458 M	8,767
Proposed network	1.2 ± 0.1‡	**0.761** / 0.940	584 k	**2,605**

TABLE II
Comparison of simplified model's inference times with batch size of 128

	Device	**Inference time**	
		per batch (ms)	per sample (μs)
CPU	Intel Xeon Silver 4110 (Dual)	1.741 ± 0.027	13.604 ± 0.207
	Intel Xeon X5690 (Dual)	1.622 ± 0.026	12.670 ± 0.206
	Intel Xeon E5-2620 v3	1.325 ± 0.123	10.350 ± 0.963
	Intel Xeon Gold 6154 (Dual)	1.167 ± 0.066	9.112 ± 0.516
GPU	Nvidia GTX 1080 Ti	1.166 ± 0.112	9.111 ± 0.876
	Nvidia TITAN X	1.154 ± 0.119	9.017 ± 0.928
	Nvidia TITAN Xp	1.062 ± 0.036	8.296 ± 0.283

C. C Simulation and RTL Synthesis Results

With all of the optimizations, the HLS design was ready for evaluation. Thanks to its high-performance, XCU250 (variant figd2104-2L-e) was chosen as the target FPGA platform. C simulation yielded an accuracy of 73.4% on a test data set

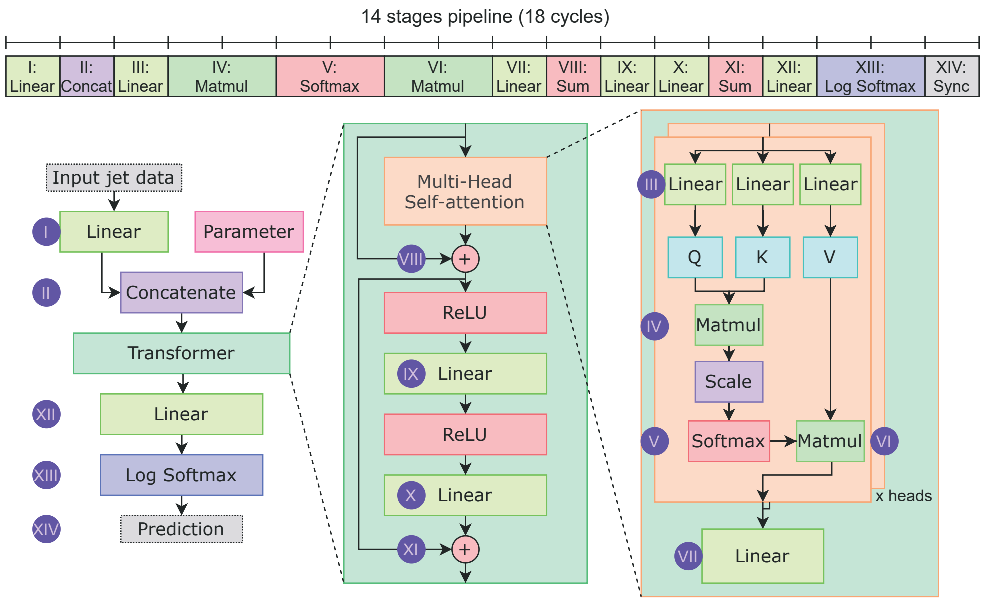

Fig. 8. Proposed architecture with highlighted pipeline stages.

composed of 165,760 (80/20 split with training data set) 16-dimensional HLF samples, which is a result worse by only 2.3 percent points compared to the Pytorch implementation. Due to the intrinsic reduced precision of the fixed-point arithmetic, that score matches the expectations.

The RTL synthesis report shows a remarkable improvement in the inference time, from the Pytorch implementation values in the magnitude of 10 μs down to 18 cycles at 200 MHz which equates to 90 ns per sample. What is more, the design is free from any pipeline stalls and achieves an initiation interval of 1, meaning that a new sample can be input on each cycle, with its result available after 18 cycles. Table III lists the hardware resource utilization. It can be seen, that the selected FPGA has a significant headroom, with DSP slices usage being most significant due to the calculation-intensive nature of transformer layers, followed by LUTs which were prioritized as the storage over BRAMs to reduce latency. Exploiting the design parallelism by using more hardware resources to achieve lower latency is possible, and we leave that for our future work.

TABLE III
FPGA RESOURCES UTILIZATION

	BRAM 18K	DSP48E	FF	LUT
Total used	12	4,351	58,942	298,881
Available	5,376	12,288	3,456,000	1,728,000
Utilization	0.22%	35.41%	1.71%	17.30%

D. Latency Analytical Model

A visualization of the architecture annotated with the pipeline stages can be seen in Fig. 8. As a consequence of pipelining the model, all the operations can be completed in either one or two cycles. Certain operations like ReLU or scaling share their stage with subsequent components due to their simplicity.

The design configuration affects several of the pipeline stages, and hence a significant portion of the latency. The number of transformer layers have a direct impact on stages III to XI, with each additional layer effectively duplicating them, increasing the latency by 12 cycles. Within the transformer block, each feed forward (linear and ReLU) layer contributes one cycle. It is not obvious at first, but the number of self-attention heads do not influence the latency as they are executed in parallel.

$$Latency\ (cycles) = C_{top} + T \cdot (C_{SA} + C_T + F \cdot C_{FF}) \quad (6)$$

Overall, the latency can be expressed using a few variables and constants, leading to the dependency seen in equation 6. As for the notation, C_{top}, C_{SA}, C_T, and C_{FF} stand for the constant cycles needed in the top-level, self-attention, transformer, and feed-forward components, while T and F mean the number of transformers and feed-forward layers. Given the design configuration of proposed model, the equation suggests 18 cycles which is a correct approximation, however without additional experiments, the analytical models allows for reasoning about alternative designs based on the proposed one.

E. Post-Training Quantization

Compared to the previous quantization methods, the proposed post-training quantization approach is noticeably easier to experiment with and deploy for existing models. By simply specifying the target environment, the algorithm adapts the

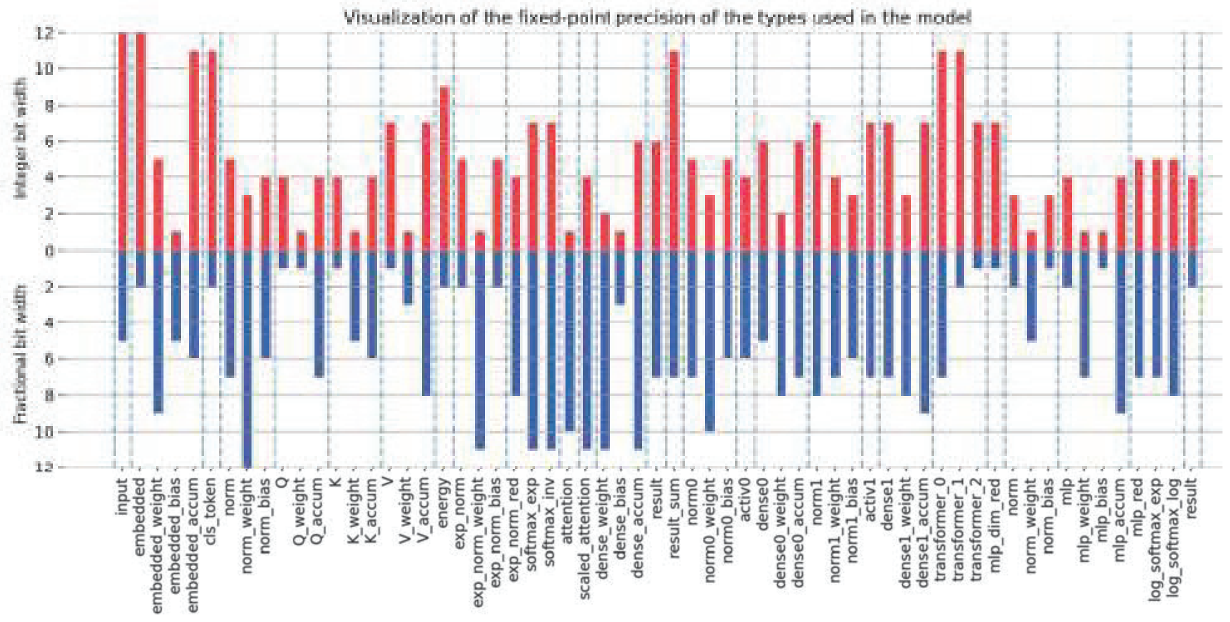

Fig. 9. Visualization of the optimal fixed-point precision of the types used in the proposed model.

hardware implementation and finds the optimal set of bit-widths. In this project, 60 individual variables are analyzed for a maximum quantization granularity, which significantly differs from the quantization-aware training experiments [27]. Fig. 9 showcases the final integer and fractional bit-widths, with types related to the same layer logically grouped.

This method achieved a 64 % reduction in total bit-width with a 2 % decrease in accuracy. Total bit-width relatively closely corresponds to the hardware resource utilization - while the FFs and LUTs are expected to vary linearly, DSPs and BRAMs change step-wise and are only affected after their width threshold is exceeded. In other words, the target FPGA's DSP slices allow up to 27 bits as their main input, hence 28 bits or more lead to an additional unit being instantiated. For that reason, the algorithm uses C simulation accuracy and bit-widths as its two driving signals, without the need of performing the synthesis. By exploiting this simplified bit-width to hardware resources relation, the total run-time of the tool is significantly shorter as each simulation runs within minutes instead of hours or even not finishing at all as it was discussed before. However, the total number of simulations still lead to a program run-time of around 7 days given the model's complexity. As a result, only one configuration of the algorithm's positive and negative accuracy tolerances was tested, where the negative one was assigned twice the importance of the positive tolerance, driving the substantial bit-width reduction while sacrificing a fraction of the accuracy.

The success of this method proves the hypothesis about strong correlation between neighboring layers. More specifically, the average total bit-width difference between layers is measured at approximately 4 bits. Furthermore, corresponding types also seem to follow the same trends among different layers - a good example are the widths of *queries*, *keys*, and *values* linear layers, all of which have similar input, weight, and accumulator widths, with the *values*' being slightly less similar, as they are used further along the computations than the other two. It is also worth mentioning that there is no noticeable difference in results regardless of whether the integer or the fractional parts are analyzed first.

VI. Related Work

Transformer architectures are computationally intensive due their underlying attention mechanism, which requires a number of matrix operations. In order to benefit from transformer's state-of-the-art accuracy in various domains, several acceleration techniques using FPGAs have been tried. That includes methods which partially trade-off accuracy for better performance by weight pruning [28, 29, 30, 31], as well as other approaches that overcome the complexity without affecting the results' precision by focusing on higher hardware utilization [32, 33], reduced memory accesses [34, 35], exploiting sparse data patterns [36, 37] or a combination of these methods [38].

High energy physics experiments often require ultra-low latency computation, which poses a challenge for real-time data processing using machine learning [39]. Aside from transformer architectures, other studies in this field have traditionally focused on Deep Neural Networks [4, 27, 40, 41] or Graph Neural Networks (GNNs) [26, 42] and their optimization [43, 44]. The relative popularity of GNN-based accelerators comes from their richer research history, making them a suitable choice for comparisons with novel transformer models.

VII. Conclusions and Future Work

This paper proposes an FPGA accelerator for TNN-based jet-tagging. The state-of-the-art software architecture is adapted in a hardware-aware manner to the HLF jet dataset. Then a hardware mapping with several optimizations is performed that leads to a design that achieves superior latency that is well within the constraints of real-time processing in L1T at LHC. The introduced computationally-efficient post-training quantization scheme offers a significant bit-width reduction potential.

The resulting work serves as a proof-of-concept highlighting the potential of the proposed architecture. There is a number of available tuning opportunities, namely quantization-aware training as well as pruning, to further improve the hardware performance. Lastly, the work also aims to contribute to the open-source community by examining the possibility of introducing useful hardware blocks into libraries such as those for `hls4ml` to enable future experiments with them automatically.

Acknowledgement

The support of the United Kingdom EPSRC (grant number EP/V028251/1, EP/L016796/1, EP/N031768/1, EP/P010040/1, and EP/S030069/1) and AMD Xilinx is gratefully acknowledged.

References

[1] J. Cogan, M. Kagan, E. Strauss, and A. Schwarztman, "Jet-images: computer vision inspired techniques for jet tagging," *The journal of high energy physics*, vol. 2015, no. 2, pp. 1–16, Feb 18, 2015. [Online]. Available: https://link.springer.com/article/10.1007/JHEP02(2015)118

[2] L. de Oliveira, M. Kagan, L. Mackey, B. Nachman *et al.*, "Jet-images - deep learning edition," *The journal of high energy physics*, vol. 2016, no. 7, pp. 1–32, Jul 13, 2016. [Online]. Available: https://link.springer.com/article/10.1007/JHEP07(2016)069

[3] L. Moore, K. Nordstrom, S. Varma, and M. Fairbairn, "Reports of my demise are greatly exaggerated: n-subjettiness taggers take on jet images," *SciPost physics*, vol. 7, no. 3, p. 036, Sep 24, 2019. [Online]. Available: https://hal.archives-ouvertes.fr/hal-01851157

[4] J. Duarte, S. Han, P. Harris, S. Jindariani, E. Kreinar, B. Kreis, J. Ngadiuba, M. Pierini, R. Rivera, N. Tran *et al.*, "Fast inference of deep neural networks in fpgas for particle physics," *Journal of Instrumentation*, vol. 13, no. 07, p. P07027, 2018.

[5] F. Fahim *et al.*, "hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices," Mar 9, 2021. [Online]. Available: https://arxiv.org/abs/2103.05579

[6] Y. Iiyama, G. Cerminara, A. Gupta, J. Kieseler, V. Loncar, M. Pierini, S. R. Qasim, M. Rieger, S. Summers, G. Van Onsem *et al.*, "Distance-weighted graph neural networks on fpgas for real-time particle reconstruction in high energy physics," *Frontiers in big Data*, vol. 3, p. 598927, 2021.

[7] A. Elabd, V. Razavimaleki, S.-Y. Huang, J. Duarte *et al.*, "Graph neural networks for charged particle tracking on FPGAs," Dec 3, 2021. [Online]. Available: https://arxiv.org/abs/2112.02048

[8] D. Thorwarth and D. A. Low, "Technical Challenges of Real-Time Adaptive MR-Guided Radiotherapy," *Frontiers in Oncology*, vol. 11, p. 332, 2021.

[9] Z. Que, E. Wang, U. Marikar, E. Moreno, J. Ngadiuba, H. Javed, B. Borzyszkowski, T. Aarrestad, V. Loncar, S. Summers *et al.*, "Accelerating recurrent neural networks for gravitational wave experiments," in *2021 IEEE 32nd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*. IEEE, 2021, pp. 117–124.

[10] S. Denholm *et al.*, "Low latency FPGA acceleration of market data feed arbitration," in *2014 IEEE 25th International Conference on Application-Specific Systems, Architectures and Processors*. IEEE, 2014.

[11] "Technical Design Report for the Phase-II Upgrade of the ATLAS TDAQ System," CERN, Geneva, Tech. Rep., Sep 2017. [Online]. Available: https://cds.cern.ch/record/2285584

[12] "The Phase-2 Upgrade of the CMS Level-1 Trigger," CERN, Geneva, Tech. Rep., Apr 2020. [Online]. Available: https://cds.cern.ch/record/2714892

[13] "The Phase-2 Upgrade of the CMS Data Acquisition and High Level Trigger," CERN, Geneva, Tech. Rep., Mar 2021. [Online]. Available: https://cds.cern.ch/record/2759072

[14] G. Keren and B. Schuller, "Convolutional RNN: An enhanced model for extracting features from sequential data." IEEE, Jul 2016, pp. 3412–3419. [Online]. Available: https://ieeexplore.ieee.org/document/7727636

[15] J. Chorowski, D. Bahdanau, D. Serdyuk, K. Cho *et al.*, "Attention-based models for speech recognition," Jun 24, 2015. [Online]. Available: https://arxiv.org/abs/1506.07503

[16] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit *et al.*, "Attention is all you need," Jun 12, 2017. [Online]. Available: https://arxiv.org/abs/1706.03762

[17] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural networks*, vol. 107, pp. 3–11, Nov 2018. [Online]. Available: https://dx.doi.org/10.1016/j.neunet.2017.12.012

[18] J. L. Ba, J. R. Kiros, and G. E. Hinton, "Layer normalization," Jul 21, 2016. [Online]. Available: https://arxiv.org/abs/1607.06450

[19] C. Garbin, X. Zhu, and O. Marques, "Dropout vs. batch normalization: an empirical study of their impact to deep learning," *Multimedia tools and applications*, vol. 79, no. 19-20, pp. 12 777–12 815, Jan 22, 2020. [Online]. Available: https://link.springer.com/article/10.1007/s11042-019-08453-9

[20] F. Cabello, J. Leon, Y. Iano, and R. Arthur, "Implementation of a fixed-point 2d gaussian filter for image processing based on FPGA." Division of Signal Processing and Electronic Systems, Poznan University of Technology (DSPES PUT), Sep 2015, pp. 28–33. [Online]. Available: https://ieeexplore.ieee.org/document/7365108

[21] R. Solovyev, A. Kustov, D. Telpukhov *et al.*, "Fixed-point convolutional neural network for real-time video processing in FPGA." IEEE, Jan 2019, pp. 1605–1611. [Online]. Available: https://ieeexplore.ieee.org/document/8656778

[22] A. H. Barr, "The einstein summation notation," *An Introduction to Physically Based Modeling (Course Notes 19), pages E*, vol. 1, p. 57, 1991. [Online]. Available: http://vucoe.drbriansullivan.com/wp-content/uploads/Einstein-Summation-Notation.pdf

[23] P. Blanchard, D. J. Higham, and N. J. Higham, "Accurate computation of the log-sum-exp and softmax functions," *arXiv preprint arXiv:1909.03469*, 2019. [Online]. Available: https://arxiv.org/abs/1909.03469

[24] E. Kreinar, J. Ngadiuba, Z. Wu, P. Harris *et al.*, "Fast inference of deep neural networks in FPGAs for particle physics," Institute of Physics (IOP), Tech. Rep. 13, Apr 16, 2018. [Online]. Available: http://cds.cern.ch/record/2316331

[25] M. Cacciari, G. P. Salam, and G. Soyez, "The anti-kt jet clustering algorithm," *The journal of high energy physics*, vol. 2008, p. 063, Apr 1, 2008. [Online]. Available: http://iopscience.iop.org/1126-6708/2008/04/063

[26] E. A. Moreno, O. Cerri, J. M. Duarte, H. B. Newman, T. Q. Nguyen, A. Periwal, M. Pierini, A. Serikova, M. Spiropulu, and J.-R. Vlimant, "Jedi-net: a jet identification algorithm based on interaction networks," *The European Physical Journal C*, vol. 80, no. 1, pp. 1–15, 2020.

[27] C. N. Coelho, A. Kuusela, S. Li, H. Zhuang, J. Ngadiuba, T. K. Aarrestad, V. Loncar, M. Pierini, A. A. Pol, and S. Summers, "Automatic heterogeneous quantization of deep neural networks for low-latency inference on the edge for particle detectors," *Nature Machine Intelligence*, vol. 3, no. 8, pp. 675–686, 2021.

[28] H. Peng, S. Huang, and others., "Accelerating transformer-based deep learning models on fpgas using column balanced block pruning," in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*. IEEE, 2021, pp. 142–148.

[29] P. Qi, Y. Song, H. Peng, S. Huang, Q. Zhuge, and E. H.-M. Sha, "Accommodating transformer onto fpga: Coupling the balanced model compression and fpga-implementation optimization," in *Proceedings of the 2021 on Great Lakes Symposium on VLSI*, 2021, pp. 163–168.

[30] B. Li, S. Pandey, H. Fang, Y. Lyv, J. Li, J. Chen, M. Xie, L. Wan, H. Liu, and C. Ding, "Ftrans: energy-efficient acceleration of transformers using fpga," in *Proceedings of the ACM/IEEE International Symposium on Low Power Electronics and Design*, 2020, pp. 175–180.

[31] X. Zhang, Y. Wu, P. Zhou, X. Tang, and J. Hu, "Algorithm-hardware co-design of attention mechanism on fpga devices," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 20, no. 5s, pp. 1–24, 2021.

[32] W. Ye, X. Zhou, J. T. Zhou, C. Chen, and K. Li, "Accelerating attention mechanism on fpgas based on efficient reconfigurable systolic array," *ACM Transactions on Embedded Computing Systems (TECS)*, 2022.

[33] S. Hong, S. Moon, J. Kim, S. Lee, M. Kim, D. Lee, and J.-Y. Kim, "DFX: A Low-latency Multi-FPGA Appliance for Accelerating Transformer-based Text Generation," in *2022 55th IEEE/ACM International Symposium on Microarchitecture (MICRO)*. IEEE, 2022, pp. 616–630.

[34] Z. Zhao, R. Cao, K.-F. Un, W.-H. Yu, P.-I. Mak, and R. P. Martins, "An fpga-based transformer accelerator using output block stationary dataflow for object recognition applications," *IEEE Transactions on Circuits and Systems II: Express Briefs*, 2022.

[35] T. Wang, L. Gong, C. Wang, Y. Yang, Y. Gao, X. Zhou, and H. Chen, "Via: A novel vision-transformer accelerator based on fpga," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 2022.

[36] L. Liu, Z. Qu, Z. Chen, F. Tu, Y. Ding, and Y. Xie, "Dynamic sparse attention for scalable transformer acceleration," *IEEE Transactions on Computers*, 2022.

[37] C. Fang, A. Zhou, and Z. Wang, "An algorithm–hardware co-optimized framework for accelerating n: M sparse transformers," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 2022.

[38] H. Peng, S. Huang, S. Chen, B. Li, T. Geng, A. Li, W. Jiang, W. Wen, J. Bi, H. Liu *et al.*, "A length adaptive algorithm-hardware co-design of transformer on fpga through sparse attention and dynamic pipelining," in *Proceedings of the 59th ACM/IEEE Design Automation Conference*, 2022, pp. 1135–1140.

[39] J. Duarte, P. Harris, S. Hauck, B. Holzman, S.-C. Hsu, S. Jindariani, S. Khan, B. Kreis, B. Lee, M. Liu *et al.*, "Fpga-accelerated machine learning inference as a service for particle physics computing," *Computing and Software for Big Science*, vol. 3, no. 1, pp. 1–15, 2019.

[40] M. Rognlien, Z. Que, J. G. Coutinho, and W. Luk, "Hardware-aware optimizations for deep learning inference on edge devices," in *International Symposium on Applied Reconfigurable Computing*. Springer, 2022, pp. 118–133.

[41] J. Ngadiuba, V. Loncar, M. Pierini, S. Summers, G. Di Guglielmo, J. Duarte, P. Harris, D. Rankin, S. Jindariani, M. Liu *et al.*, "Compressing deep neural networks on fpgas to binary and ternary precision with hls4ml," *Machine Learning: Science and Technology*, vol. 2, no. 1, p. 015001, 2020.

[42] S. Thais, P. Calafiura, G. Chachamis, G. DeZoort, J. Duarte, S. Ganguly, M. Kagan, D. Murnane, M. S. Neubauer, and K. Terao, "Graph neural networks in particle physics: Implementations, innovations, and challenges," *arXiv preprint arXiv:2203.12852*, 2022.

[43] Z. Que *et al.*, "Optimizing Graph Neural Networks for Jet Tagging in Particle Physics on FPGAs," in *32th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2022.

[44] Z. Que, H. Fan, M. Loo, M. Blott, M. Pierini, A. D. Tapper, and W. Luk, "LL-GNN: Low Latency Graph Neural Networks on FPGAs for Particle Detectors," *arXiv preprint arXiv:2209.14065*, 2022.