Imperial College London

MEng Individual Project

Imperial College London

Department of Computing

Reconfigurable Acceleration of Transformer Neural Networks with Meta-Programming Strategies for Particle Physics Experiments

Author:
Filip Wojcicki

Supervisor:
Prof. Wayne Luk

Second Marker:
Prof. Alexander Tapper

June 10, 2022

Abstract

Particle Physics studies the fundamental forces and elementary particles building the Universe. In order to verify the correctness of the theories, countless experiments have to be designed and carefully executed, with the main driving force of the myriads of engineers, physicists and researchers at the Large Hadron Collider (LHC) operated by the European Organization for Nuclear Research (CERN). With the unprecedented experiments' scale comes the challenge of accurate, ultra-low latency decision-making. Transformer Neural Networks (TNN) have been proven to accomplish cutting-edge accuracy in various domains, including classification for jet tagging, which is the target of this project. However, software-centered solution implemented for CPUs and GPUs lack the inference speed needed for real-time particle triggers.

This report proposes two novel TNN-based architectures efficiently mapped to Field-Programmable Gate Arrays (FPGAs). The first one outperforms the current state-of-the-art models' GPU inference capabilities by roughly 1000 times while maintaining comparable classification accuracy. The second one trades off some of its speed for accuracy and undergoes a broad design-space exploration, which involves both pre-training and post-training quantization. The latter one leverages a custom-developed tool chain that augments existing solutions in terms of granularity and ease-of-use while following an innovative algorithm for relatively quick convergence.

Confirm this number

In this project, several recently researched neural network components are designed to target FPGAs using High-Level Synthesis (HLS). The resulting open-sourced building blocks are both highly customizable and abstract, and aim to bridge the gap between hardware and software development, effectively reducing the time and complexity needed for creating efficient neural network hardware accelerators.

Acknowledgements

I would like to express my gratitude to Professor Wayne Luk for his guidance, insightful suggestions and constant encouragement throughout the project.

I would like to thank Professor Tapper for giving me a different view on the project's meaning and providing with the behind-the-scenes information about the LHC.

I want to thank Zhiqiang Que for his continuous technical support, our weekly meetings and always being available to answer any of my questions.

Lastly, I am very grateful for my family whose support was invaluable during this project and the degree as a whole.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

LHC is the world's highest-energy particle collider that is capable of producing and detecting the heaviest types of particles that emerge from events such as proton-proton collisions. The detection is a challenging process as some particles, like quarks and gluons, cannot exist on their own, and they nearly instantly combine which results in collimated sprays of composite particles (hadrons) that are referred to as jets [1]. The initial particles created upon collision and their behaviors are of main interest of the physicists, which leads to jet tagging - the challenge of associating particle jets with their origin.

There are many detector types used for the analysis of the particle collisions, each based on a different physical phenomenon, which results in availability of both higher and lower level features from each event. The former have been successfully used in the past using more physically motivated machine learning (ML) algorithms, e.g. using computer vision [2]. However, more recently, various deep learning approaches have proven to outperform their predecessors [3]. It has also been found that all the detected features carry the same underlying information, with convolutional neural networks trained on higher-level data achieving nearly identical accuracy as dense neural networks trained on the data from the other end of the spectrum [4].

The information throughput of Petabytes per second collected by the LHC detectors outclasses the real-time inference capabilities of the typical state-of-the-art solutions. The real-time decision-making is often of utmost interest, hence 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, ***hls4ml*** codesign workflow have been successfully adopted in particle physics experiments [5], which allows ML researchers and physicists to easily deploy their solutions trained using common ML frameworks on reconfigurable or application specific hardware, vastly improving the detection algorithms' inference capabilities. However, *hls4ml* lacks support for a number of neural network architectures that have been proven to outperform the previous state-of-the-art, including graph neural networks [6, 7] and transformer neural networks [8].

1.2 Objectives and Challenges

The purpose of this project is to develop novel, hardware-aware neural network architectures as well as to establish efficient ways mapping them onto FPGAs. Another objective is to use metaprogramming strategies to integrate them into the *hls4ml* library or standalone tools, with various optimizations approaches that offer trade-offs between latency, throughput and hardware resources usage. Hence, there is an emphasis on creating parametrizable and reusable designs that can support creation of ultra-low latency systems, effectively transforming proof-of-concept implementations into optimized hardware accelerators.

The two main challenges of the project involve:

- Developing deep, complex and fast neural network models which require much lower abstraction levels than a typical ML framework. It is also crucial to stay aware of the underlying hardware architecture to exploit its strengths while keeping compile-time and run-time configuration easily accessible.
- Bridging the abstraction gap for the translation between high-level representation of neural networks and their optimized mapping to hardware. The design space exploration is a long and difficult process, which needs careful examination and analytical performance models to find the optimal solutions.

1.3 Contributions and Publication

The project aims to benefit the open-source community of ML researches that are in need of faster and more parametrizable neural network inference. The main audience for that operation are particle physicists, nonetheless, the hope is for the work to positively contribute many ML fields by both offering a reliable tool for acceleration of existing designs and providing a useful resource for learning about the nature of reconfigurable hardware and its optimization potential.

The bulk of the work and analysis conducted in this project was summarized in the paper "Accelerating Transformer Neural Networks on FPGAs for High Energy Physics Experiments" and submitted in the long paper category to the *18^{th} International Symposium on Applied Reconfigurable Computing*. A journal article derived from this project is being prepared for publication.

1.4 Report Outline

This report begins by discussing the necessary particle physics background to understand the scope of the work, followed by the related work in the field of machine learning, with an emphasis on the state-of-the-art architectures, and a deeper dive into the reconfigurable hardware technology in chapter 2. Then in chapter 3, the two proposed novel neural network architectures are described in details, including the necessary training and processing steps. After that, chapter 4 covers both the existing and custom ways of conducting design space exploration and how they were applies in this project. Chapter 5 discusses the evaluation metrics and collected experimental results, which is concluded by chapter 6, which also proposes future work derived from this analysis.

Chapter 2

Background and related work

This chapter provides a closer look at the concepts required to understand this work. The following sections firstly discuss background and related work for topics in particle physics, then machine learning and finally reconfigurable hardware research. At the end of this chapter, the ethical issues that could arise from the project are listed and discussed.

2.1 Particle Physics

To be able to understand the scope of the project and the applicability of the work in modern research, this chapter gives an overview of the key concepts from particle physics that appear through the paper. The explanation is written for readers with no prior background in physics.

2.1.1 Standard Model

the Standard Model is a theory that describes the connections between weak, strong and electromagnetic interactions, which are three of the fundamental forces. The possible unification with the forth one - gravity - is an ongoing research [9], and while certainly outside the scope of this project, it should be noted that some of the physical experiments that this work explores aim to help with it [10, 11].

The Standard Model also provides a classification of all the elementary particles. A non-exhaustive list of them is described below, with particles that this report is concerned about (as they appear in the proton-proton collisions) being highlighted.

- Fermions
  - Leptons - participate in electroweak interactions; include electron (e^-)
  - Quarks - participate in strong interactions; include **light quarks (q)**[1] and **top (t) quark**
- Bosons
  - Gauge bosons - force carriers; include photon (γ), **W boson (W^+, W^-)**, **Z boson**, **gluons**
  - Scalar bosons - give rise to mass; include **Higgs boson (H^0)**

The information about the following decay processes form the dataset of this report, with visualization in figure 2.1 (obtained from [6]). It is important to note that where applicable, the particles on the left-hand side of the arrows undergo a series of decays before reaching the right-hand side, when the only particles left are those composed of quarks and antiquarks (denoted by the vertical bar), referred to as hadrons.

Possibly clear page

$$q/g$$

$$H^0/W/Z \rightarrow q\bar{q}$$

$$t \rightarrow Wq \rightarrow q\bar{q}q$$

[1] Light flavor quarks: up (u), down (d), charm (c), and strange (s) quarks

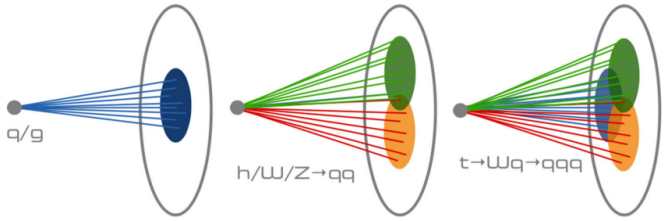

Figure 2.1: Representation of different decay processes, based on the number of resulting jet clusters.

2.1.2 Particle Accelerators and Triggers at LHC

The two LHC experiments that are of most concern in this report are CMS and ATLAS. They are both large general-purpose particle detectors, that were notably involved in the discovery of the Higgs boson [12]. Several processing steps happen between particles colliding and theories being proven, however real-time particle detectors comprises the very first elements of this pipeline. They are composed of triggers split into the following levels [13, p.16]:

- **Level 1 trigger (L1T)** - it is implemented in hardware (FPGAs) and firmware, it is pipelined (a term explained in details in subsection 2.3.5) and it cannot allow for any dead-time, which means that it has to continuously process data with a fixed latency.
- **Level 2 trigger (L2T)** - it is implemented in hardware and software and can include regional processing.
- **Level 3 trigger (L3T)** - it is implemented in software, using farms of CPUs. It is close in behavior to non-real-time algorithms.

LHC operates in intertwined periods of operation and shutdown. The latter come from the demanding nature of the experiments that necessitates maintenance and upgrades to the apparatus and machinery, as well as the science and engineering advancements which allow for more efficient algorithms and technologies to be adopted. Very recently, after four years of break, LHC restarted experiments, which marks the beginning of "Run 3" [14]. Since their origin, the L2T and L3T have been merged into High Level Trigger (HLT) [15, p.47], which is planned to rely on thousands of multithreaded CPUs and GPUs. As for the L1T key specifications that will be used to evaluate the design in this paper, its input data frequency is 40 MHz, which with a pipeline depth of 500 results in a 12.5 μs latency, and its output frequency to HLT is equal to 750 kHz.

2.1.3 Dataset and Notations

The datasets used in this work has been simulated to mimic the 13 TeV proton-proton collisions performed at LHC, and it includes information about the most energetic jets [16] (30 [17], 50 [18], 100 [19] and 150 [20]) that were constructed using the anti-K_t clustering algorithm [21]. A number of jet representations are available in the dataset:

- High level features (HLF), which are physically inspired,
- Images, which are related to an energy heat-map,
- Constituent list, which contains jets' constituent hadrons from the following list: light quarks, top quarks, W bosons, Z bosons, and gluons.

Compared to the other two, the constituent list is a lower-level representation, however, as mentioned in chapter 1, this should not affect the classification accuracy [4]. It is also worth mentioning that a simpler dataset that contains only the HLF jet representation [22] is also used in this project as it vastly reduces the complexity of a design while offering comparable accuracy. A more thorough discussion between the differences in their use cases is carried in chapter 5, but it is worth mentioning that the HLF representation has been successfully used in conjunction with deep neural networks [23], while the images and constituent lists were adopted for graph neural networks [6].

To facilitate further analysis, this subsection also explains the notation used for the dataset as it is important throughout the report. Each constituent element $\boldsymbol{x^l}$ is a 16-dimensional vector, where l denotes the index in the list:

$$\boldsymbol{x^l} = \begin{bmatrix} x_0^l & x_1^l & \dots & x_{15}^l \end{bmatrix}^T \in \mathbb{R}^{16} \tag{2.1}$$

The physical meaning of each element's dimension is not taken into consideration, and all of them are treated as equally important. A constituent list $\boldsymbol{x^i}$ acts a single sample, with index i within a dataset, and it varies in terms of the number of constituents L, but it has no more of them than the dataset name suggests:

$$\hat{\boldsymbol{x^i}} = \begin{bmatrix} \boldsymbol{x^{i,0}} & \boldsymbol{x^{i,1}} & \dots & \boldsymbol{x^{i,L-1}} \end{bmatrix} \in \mathbb{R}^{L \times 16} \tag{2.2}$$

Using the 30 jet dataset as an example, each sample has between 1 and 30 constituents, although in the majority of samples it is the upper boundary. Hence, the whole dataset with N samples can be represented as D:

$$D = \begin{bmatrix} \hat{\boldsymbol{x^0}} \\ \hat{\boldsymbol{x^1}} \\ \vdots \\ \boldsymbol{x^{\hat{N}-1}} \end{bmatrix} = \begin{bmatrix} \boldsymbol{x^{0,0}} & \boldsymbol{x^{0,1}} & \dots & \boldsymbol{x^{0,L-1}} \\ \boldsymbol{x^{1,0}} & \boldsymbol{x^{1,1}} & \dots & \boldsymbol{x^{1,L-1}} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{x^{N-1,0}} & \boldsymbol{x^{N-1,1}} & \dots & \boldsymbol{x^{N-1,L-1}} \end{bmatrix} \in \mathbb{R}^{N \times L \times 16} \tag{2.3}$$

The main jet datasets contain $880,000$ samples regardless of the number of jets per sample, while the HLF dataset contains $830,000$ examples, which are split into training and test samples in $70:30$ and $80:20$ proportions accordingly.

2.2 Machine Learning

Neural networks belong to a wider field of machine learning (ML) - the study of using experience to improve algorithms. This section assumes a basic understanding of ML and gives a brief overview of the topics needed to understand the scope of the project. It then explains in more details the background and related work for the architectures involved in this research.

2.2.1 Metrics

There are several key metrics used for assessing the success of an ML algorithm, and the following will be used throughout the report:

- **Classification accuracy** - a simple measure of the percentage of correctly classified samples.
- **Area Under the Curve (AUC) for the Receiver Operator Characteristic (ROC)** - a more complex measure of the model's ability to correctly distinguish between classes. It can be used similarly to the classification accuracy, but it favors discriminative over representative models.
- **Confusion matrix** - a tabular metric that compares the actual samples' classes with the predicted ones, effectively categorizing results into four groups: true positive, false positive, false negative, and true negative. This allows for an easy calculation of precision and recall values.

Give equations for TPR, FPR, AUC etc.

Explain when accuracy is not enough and AUC and confusion matrix gives a better picture

|

2.2.2 Deep Neural Networks

While there exist a number of ML techniques that have proven successful for various use cases at LHC, like Support Vector Machines [24] or Boosted Decision Trees [25], in the last years deep neural networks (DNN) have been proposed with improved results for applications like infrastructure monitoring [26], offline data analysis [27], and the main interest of this report - detectors' trigger mechanisms.

In many uses cases the neural networks architectures are optimized and accelerated to shorten the training time (measured in hours or even days) to reduce the time needed for evaluating different design configurations and easily performing the hyperparameter search. However, this work focuses on accelerating the inference to match the extremely low latency required in the LHC detectors' L1 triggers. Although often measured in milliseconds, sub-milliseconds inference time has been achieved for this application with the use of FPGAs using architectures for basic DNN [22], and recently sub-microseconds latency for graph neural networks (GNN) [28, 29]. These implementations serve as a baseline latency for this project which aims to achieve comparable performance with higher AUC value.

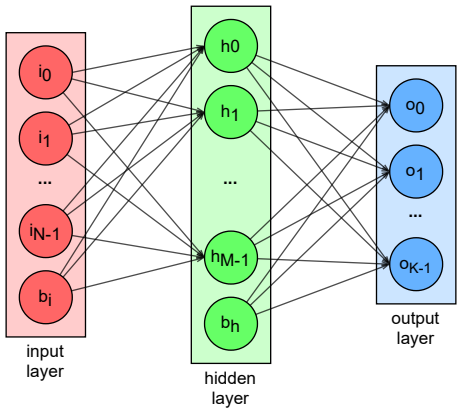

Figure 2.2: Diagram of a fully connected layer.

Figure 2.2 shows an overview of a fully connected neural network with one hidden layer, which allows to derive the mathematical formulae. Each layer consists of neurons which hold a value, which means that input, output and the intermediate hidden layer can be modelled similarly. The arrows between neurons represent learnable weights, while the (optional) biases involved in the calculation, denoted as b, can be depicted as arrows from the "bias neurons", in all but the last layer. The value of a neuron depends on all the neurons in the previous layer as well as the weights and biases between them, which can be formulated into the equation 2.4 using the hidden layer as an example:

$$h_0 = f(w_{i,0} \cdot i_0 + w_{i,1} \cdot i_1 + \ldots + w_{N-1} \cdot i_{i,N-1} + b_{i,0}) = f(\sum_{j=0}^{N-1} w_{i,j} \cdot i_j + b_{i,0}) \tag{2.4}$$

What is also not displayed in the diagram, but can be seen in the equation is the activation function f that is required to introduce non-linearity in the computations. Without it, all consecutive layers involving solely multiplication and addition could be simplified to s a single layer thanks to the distributive property in linear algebra, defeating the point of having multiples of learnable parameters. The activation function can be as simple a piece wise linear function called Rectified Linear Unit (ReLU) defined in equation 2.5:

$$\text{ReLU}(x) = \max(x, 0) = \begin{cases} x & \text{x} > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.5}$$

or more complicated like the Sigmoid Linear Unit (SiLU), based on the sigmoid function, both presented in equation 2.6

$$\mathrm{SiLU(x)} = x \cdot \sigma(x) = x \cdot \frac{1}{1 + e^{-x}} \tag{2.6}$$

As layers are tightly connected to each other, this type of neural layer is often referred to as *fully connected* or *linear*. It requires a relatively large number of separate weights and biases, which makes it both computationally and memory intensive, but nonetheless, modern network architectures can have dozens of these layers, not to mention plethora of other types.

2.2.3 Convolutional Neural Networks

Quick introduction and visualization of CNNs as other notable jet-tagging algorithms use them and are mentioned in this report

2.2.4 Graph and Recurrent Neural Networks

Quick introduction and visualization of graph and recurrent (including GRU and LSTM) NNs as other notable jet-tagging algorithms use them and are mentioned in this report

2.2.5 Batch and Layer Normalization

Batch norm vs layer norm as both are used in the architecture

2.2.6 Transformer Neural Networks and Self-Attention

A promising architecture that has been chosen as the topic of this project is the transformer neural network (TNN). Similarly to 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, convolutional [30] or attention mechanisms [31] 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 [32] is used which can find global relations in a 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, formula for which is presented in equation 3.5.

$$\text{softmax}(x) = \frac{exp(x_i)}{\sum exp(x_i)} \tag{2.7}$$

Softmax can be described as mapping a vector to a ratio between each input's exponentiation result and the sum of all such values, which gives the property of the resulting vector entries sum equal to one. This characteristic means that the output can be treated as vector of probabilities, which is often exploited in the final activation layer of a neural network.

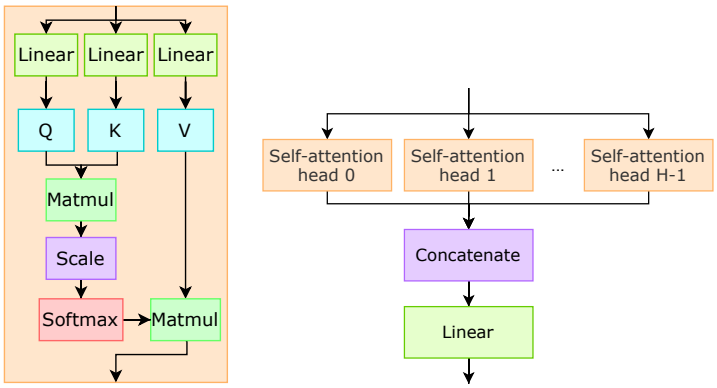

Figure 2.3: Left: diagram of a self-attention head. Right: illustration of H self-attention heads forming a multi-headed self-attention block.

A diagram representing the initial implementation of the self-attention can be seen on the left in figure 2.3. The Q, K, and V stand for *queries*, *keys*, and *values* respectively, which although arbitrary, are meant to give a better understanding behind the idea of this mechanism. 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 of an input. The results of all heads are simply concatenated, increasing output's dimensionality, and multiplied with a learned weight, as seen on the right in figure 2.3. To better comprehend the interactions between information learned by the heads, figure 2.4 shows a visualization for 1, 2 and 8 heads on an example sentence, obtained using Tensor2Tensor library [33, 34]. While it is quite clear in the example that "it" is mostly associated with "The" and "animal" with 1 attention head, the interpretability is worse in case of 2 and 8 attention heads.

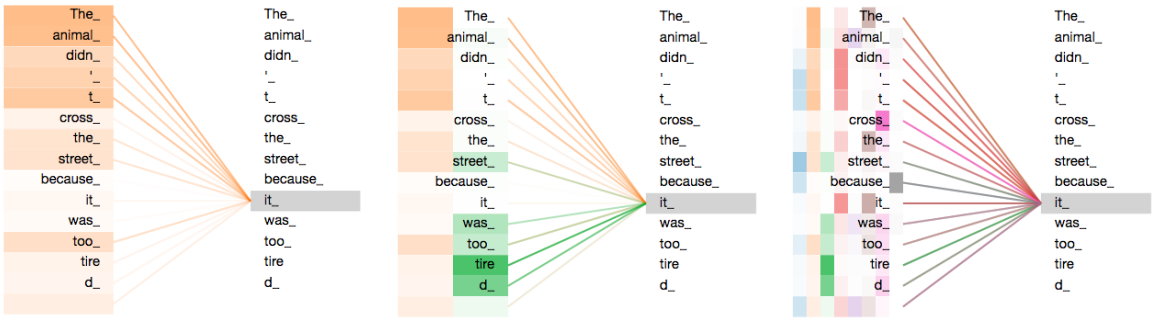

Figure 2.4: From left to right: visualizations, for an input word, of the words focused by 1, 2 and 8 attention heads.

It has to be mentioned that in terms of the AUC value, a recent implementation of a transformer called ConstituentNet [8] has been shown to outperform previous state-of-the-art GNN implementations like JEDI-net [6] and thus serves as an inspiration for the starting point architecture of chapter 3, which is entirely devoted to a further analysis of the network design and suitability for jet tagging. More specifically, given the strong results of its software implementation, an FPGA-mapped design is believed to have a possibility to be a viable alternative to existing designs for L1T at LHC.

2.3 Reconfigurable Hardware

A significant portion of the project's work involves exploiting reconfigurable hardware to vastly reduce the inference time of the proposed neural networks. This section explains in more detail the technology and characteristics of reconfigurable hardware, and in particular, FPGAs.

2.3.1 Landscape of Hardware for Computing

The modern landscape of digital integrated circuits (IC) is very rich can be divided into numerous categories depending on the technology used and expected functionality [35]. A list of platform types is described below, with the emphasis of their suitability for neural networks applications.

- **Central Processing Units (CPU)** - the most commonly found ICs that are at the core of personal computers, laptops and handheld devices. They are capable of executing a broad range of predefined instructions. As CPUs have become widely adopted in research long before the emergence of the other technologies from this list, they were the first platforms for the training and inference of neural networks with promising results back in the 1980s and 1990s for applications like high energy physics [36] or biology [37]. Although it is possible to achieve speed-ups of over 10x the baseline performance with careful optimizations [38], CPUs are now consistently outperformed by more suitable technologies, and only limited to certain inference tasks.
- **Graphic Processors (GPU)** - ICs originally specialized in graphics processing intended for displaying images. Since their inception, due to the type of calculations involving matrix and vector operations, other applications related to cryptography and neural networks have also adopted GPUs as their main platform. In the former domain, cryptocurrency mining has transitioned from CPU to GPU to increase profitability [39], while for the latter, the more powerful hardware drastically reduced training and inference times, thus allowing for deeper and more complex architectures yielding higher accuracy [40, 41].
- **Application Specific Integrated Circuits (ASIC)** - as suggested by the name, those are the custom designed ICs heavily specialized for a particular application. It is hard to generalize them, as the use cases can cover any modern computing problem, but the commonality is a vast improvement in performance and power usage compared to more general purpose solutions. However, the long and expensive development process pose extremely high barriers to entry for most users. Fortunately, off-the-shelf products like the Graphcore Intelligence Processing Units [42], that are designed specifically with machine learning applications in

mind, as well as other custom designs [43, 44] are starting to offer a compelling platform for working with neural networks.

- **Field-Programmable Gate Arrays (FPGA)** - differently from the previous listed IC types, FPGAs are not manufactured for a specific use case, and in fact, they can be reprogrammed to be a platform for a different application at any time. The reprogrammability comes at a cost of performance and power consumption compared to ASICs [45], but at the same time outperforms GPUs in these regards [46, 47]. It is also suggested, that with some technological improvements focused on ML applications, FPGAs can narrow the gap between ASICs without needing to stick to one particular design [45, 48, 49].

FPGAs offer an interesting trade-off between implementation effort and acceleration potential when it comes to neural networks and for that reason they have been chosen as the target technology in this report. The following subsections give a closer look at some of their characteristics and associated tools.

FPGA lattice overview to visualize explain the idea behind this technology

|

|

|

2.3.2 High-Level Synthesis

For many years, FPGAs have been modelled using register-transfer level (RTL) design abstraction with the use of hardware description languages like Verilog or VHDL. However, to increase productivity and allow for a more convenient design state space exploration (DSE), a more abstract modelling process called High-Level Synthesis (HLS) can be adopted. The design can be expressed in a software programming language like C, C++ or Java, which can be both manually and automatically optimized, and transformed to an equivalent RTL. This is especially beneficial in research, where compared to industrial environment, it is more likely that a slightly lower quality of results can be afforded for increased productivity and easier DSE. In fact, a recent study shows that on average only one third of design time and half of the lines of code are needed for an equivalent project done in HLS in comparison to RTL while the quality of results varies and can even outperform the RTL implementations for some applications [50].

This report's work is based on Xilinx Vivado HLS design suite. When developing a solution, it is important to note, that the synthesis process can take a significant amount of time (from a couple of hours to days on a modern powerful machine), and so there exist two simulation methods - a C-simulation that can quickly and directly evaluate a software benchmark against an emulation of the design, and a more truthful, cosimulation that firstly synthesizes a design and accompanying test bench to RTL and then performs an RTL simulation. A final, definitive evaluation of the results requires programming a target FPGA with the generated bit stream of the design and exchanging input/output data with a program that usually runs on a CPU.

HLS to RTL flow diagram

|

|

|

2.3.3 `hls4ml` Codesign Workflow

A commonality between the recent best performing hardware-mapped neural network models is the use of the *hls4ml* codesign workflow that was mentioned in section 1.1.

More about hls4ml

Difficulty: rtl > hls > python hl4ml, draw comparison with assembly

|

|

|

|

|

|

|

|

2.3.4 Latency, Throughput, and Hardware Resources

To properly navigate during the DSE and assign scores the solutions, the following characteristics have to be considered:

- **Latency** - A time measure of a system between receiving an input signal and producing a *corresponding* output. It is crucial in real-time processing where it has to be lower than the period between subsequent input samples. Depending on the application, latency in the microseconds or nanoseconds range can be expected from an FPGA. To recall from section 2.1.2, the latency constraint for this work comes from the specification of L1T at LHC and is equal to 12.5 μs.
- **Throughput** - A rate of samples processed in a unit of time. For architectures that only start to process new elements after the previous one has finished, it is directly linked to latency. However, in modern ICs, especially in FPGAs, it is one of the defining metrics of performance and designs tend to exploit pipelining and parallelizability to marginally trade off their latency to increase it. Despite that, in this work, this measure is of little interest given the fixed latency and no dead-time constraint of L1T.
- **Resource utilization** - A more complicated, often multidimensional, metric that describes either the raw number or ratio of total usage of the hardware components of an FPGA. Typically, the higher it becomes, the more power is drawn by an FPGA, however, it is most often used to guide the design process to avoid running out of a certain resource. This can be done by potentially deploying an alternative method that can be implemented using a different, less contested resource.

While FPGAs vary in terms of hardware resource configurations, several key components can be distinguished:

- **Block Random Access Memory (BRAM)** -

  finish

- **Digital Signal Processing (DSP) logic element** -

  finish

- **Flip-Flop (FF)** -

  finish

- **Lookup Table (LUT)** -

finish

To fully understand the trade-offs between designs, one cannot forget about the metric directly related to the specific task that is accelerated in hardware. In the case of this report, classification accuracy and AUC described in section 2.2.1, will also play key roles in evaluating various configurations.

2.3.5 Serial, Parallel, and Pipelined Architectures

Hardware architectures use components that can be configured in different ways depending on the overall goal or a limiting factor. The high-level configurations of building blocks are displayed in figure 2.5 and can be described as follows:

- **Serial** - elements are arranged in a chain, processing one after another. This way uses less resources than an equivalent parallel configuration by reusing given components R times, thus approximately trading-off R times less required resources for R times higher latency.

- **Parallel** - elements share a common input and start processing data at the same time. This way ends in a lower latency than an equivalent serial configuration consuming using more resources.

- **Pipelined** - a more sophisticated arrangement, in which subsequent processing blocks (that can be either placed serially or in parallel) form a pipeline of processing stations separated by simple storage elements called pipeline registers, implemented using FFS. This maximizes the usage of the design blocks, hence increasing throughput with a minimal sacrifice of latency and resource usage.

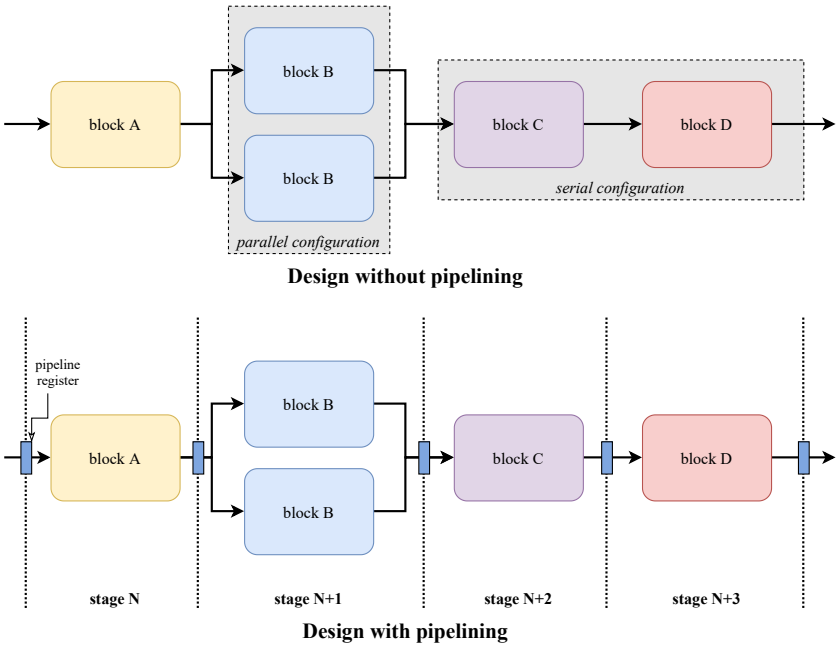

Figure 2.5: Diagram comparing serial and parallel configurations as well as showcasing designs with and without pipelining.

2.3.6 Pareto Front and Roofline Model

To make an informed design decision, various architectures can be compared by arranging them on a dependency graph (e.g. latency vs resource usage) and observing the Pareto front - the set of solutions for which there are no better ones in regard to one quality given that the other measure is not worse. The slightly complex definition can be easily understood from figure 2.6, which also highlights another use of this method - finding design configurations that are yet to be explored.

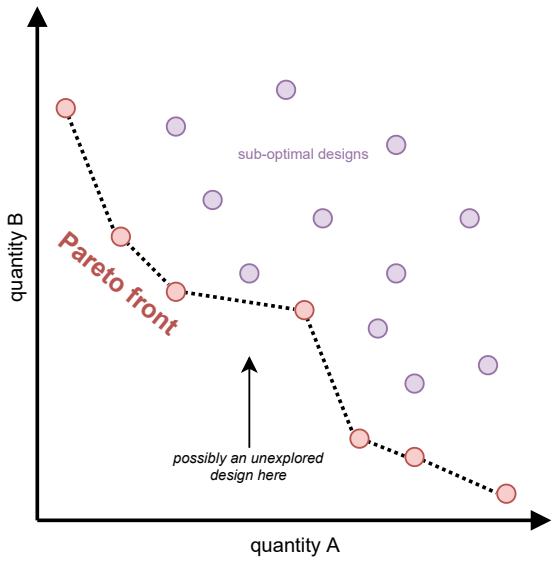

Figure 2.6: Example graph with designs plotted against quantities A/B, Pareto front highlighted.

Another intuitive performance visualization comes in the form of the Roofline model, which compares the obtained results with theoretical limits coming from inherent hardware limitations like clock frequency or memory bandwidth. An example can be seen in fig 2.7.

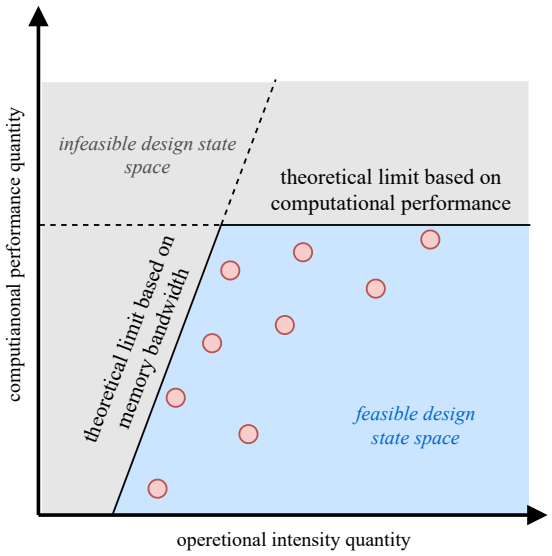

Figure 2.7: Example graph with computational and memory bandwidth limitations showcasing the Roofline model

2.4 Ethical Considerations

The purpose of this project is to advance the next-generation particle physics experiments. There are two main aspects that need to be considered - the development of a hardware-mapped transformer neural network architecture and the easy-to-access translation and optimization toolchain for efficiently expressing and networks in common machine learning frameworks.

The first feature is aimed at a purely civilian, scientific audience, and it is tailored towards particle collision datasets. With that in mind, it is important to mention that, as with most machine learning research, there is potential for a misuse of the acceleration techniques towards a military or malevolent application that could negatively impact the society. However, this also means that there is a low risk for new emerging threats from this particular work; rather the already present ones could become slightly more serious. Fortunately, this should result in existing harm prevention measures staying intact or solely requiring adjustments to their accuracy or speed thresholds.

With the second element's goal of making the creation and deployment of neural networks more accessible, it could be argued that this may in turn increase the number of high energy physics experiments requiring immense energy consumption, like those at LHC [51], thus negatively affecting the environment. However, this is considered a very low likely cause of action, as the research work of this project is aimed at helping already running experiments and more importantly, the negative environmental implications (for which there are various mitigation strategies [52, 53]) are heavily outweighed by potential beneficial technological advancements coming from the scientific discoveries.

Aside from the aforementioned ethical issues, the project is aimed at benefitting the open-source scientific community world-wide. Its outcome could lead to a much more accessible and efficient inference methods that are applicable in many domains outside particle physics.

Chapter 3

Architecture Exploration

This chapter presents the proposed neural network architectures. It starts with a baseline TNN network implemented in `PyTorch`, which then undergoes a series of hardware-aware adaptions specific to jet tagging. During this process two separate architectures are developed, which differ by the input type and design goal. The first one, referred to as the *ultra-low latency* one, targets the HLF jet representation and aims to achieve the lowest possible latency at the cost of accuracy and AUC values. The second one, called *accuracy-focused*, is based on the constituent list jet representation and trade-offs latency for quality of classification while still remaining within L1T timing constraints.

3.1 Base Architecture

The starting point of this analysis is derived from transformer architecture used in the original paper [32] and recent proof-of-concept used for jet tagging [8]. The overview of the network components can be seen in figure 3.1.

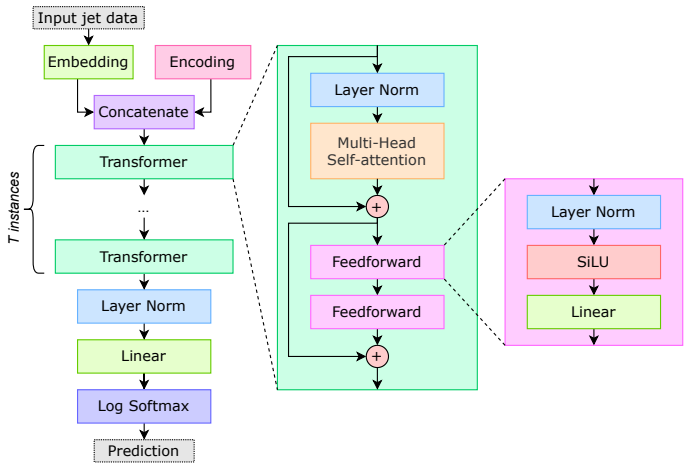

Figure 3.1: Diagram with an overview of the baseline architecture.

The straight-forward path between model's input and output highlights the sequential nature of transformer which stands in opposition to recurrency present in GRU and LSTM models. While this allows for the aforementioned parallelizability and pipelining on FPGAs, it also poses a challenge of increased hardware footprint and synthesis complexity when compared to recurrent models, where the key components can get reused to meet the resource constraints. To better understand the transformer's complexity, the next subsections derive the equations linking the internal components and explain the involved terminology.

3.1.1 Input embedding and Residual Connections

Although the model lacks any recurrency, the transformer includes two residual connections which have been widely adopted since their successful application in ResNet neural networks [54]. They offer improvements to training time and resulting accuracy [55], however, they require standardized data dimensionality to ensure the summation can be logically executed. In this project, this is obtained thanks to input embedding, which transforms the input $\hat{\boldsymbol{x}}^{\boldsymbol{i}} \in \mathbb{R}^{L\times 16}$ into a shape $\mathbb{R}^{L\times d}$ that is used through the design, as seen in equation 3.1.

$$\hat{\boldsymbol{x}}^{\boldsymbol{i}}{}_{\text{emb}} = \text{embedding}(\hat{\boldsymbol{x}}^{\boldsymbol{i}}) = w_{\text{embed}}\, \hat{\boldsymbol{x}}^{\boldsymbol{i}} + b_{\text{embed}} \in \mathbb{R}^{L\times d} \tag{3.1}$$

This dimensionality change can be conveniently performed using a linear layer, and it has to be remembered that each such layer increased the model learning capacity thanks to the learnable weights and bias. The network's inner dimension d is treated as a hyperparameter as it influences the model's accuracy and performance, but it has to be noted that the other dimension prevalent in the network comes from the input's number of jet constituents L (which is set to 1 in case of the HLS representation), meaning that the model is also susceptible to a parameter which cannot be easily tuned.

3.1.2 Input Encoding

Along the embedding, an input encoding is concatenated and fed to the transformer layer. In natural language processing, the encoding is meant to allow the model to benefit from the sequential information of the words in a sentence. It can be obtained from a sinusoidal function using the position index or simply treated as another learnable parameter. The sequential relations are not present in the jet data, because all the jets originate from the same proton-proton collision, hence, the latter approach is used in this project. It is worth mentioning, that from empirical analysis, the learnable encodings have a significant impact on the final results as they represent a trained, hidden state concatenated to all inputs during evaluation, as shown in equation 3.2. Its impact is especially prevalent for the HLF data (where $L = 1$), where the hidden state matched input's dimension and effectively doubles it after concatenation.

$$\text{encoding}(\hat{\boldsymbol{x}}^{\boldsymbol{i}}{}_{\text{emb}}) = w_{\text{encoding}} \in \mathbb{R}^{1\times d} \implies \text{concat}(\hat{\boldsymbol{x}}^{\boldsymbol{i}}{}_{\text{emb}},\ w_{\text{encoding}}) \in \mathbb{R}^{(L+1)\times d} \tag{3.2}$$

Choosing a learned hidden state is also more efficient for inference in hardware, as the increased training cost associated with back-propagation of this parameter yields a constant set of values that are known during compile-time of the FPGA and can be implemented using a LUT.

3.1.3 Normalization and Parameter Extraction

As layer normalization does not track and gather running mean and variance statistics, this mechanism is implemented on top of the existing `PyTorch` implementation to facilitate extracting the aggregated statistics after training. These, along with all the learned weights and biases, are extracted and transformed into specific C++ formats supported in HLS using a custom tool developed for this purpose. This allows for directly initializing the FPGA's BRAMs and LUTs with the model parameters, which avoids the need for an interaction with a host machine.

Obtaining the statistics taken for the data before normalization layers can also be viewed as a hardware-aware optimization. This can be explained with the mathematical derivation presented in equation 3.3

$$y = \frac{x - E[x]}{\sqrt{Var[x] + \epsilon}} \cdot \gamma + \beta = x \cdot \left(\frac{\gamma}{\sqrt{Var + \epsilon}}\right) + \left(\beta - \frac{\gamma * E}{\sqrt{Var + \epsilon}}\right) = w \cdot x + b \tag{3.3}$$

By treating the mean $E[x]$ and variance $Var[x]$ of input x as learned parameters, the square root and division operations can be fully omitted by fusing them into the existing γ and β parameters which simplifies the hardware required for the normalization layers. This is especially useful as FPGAs lack dedicated hardware for these computationally expensive operations, which could lead

to suboptimal designs being synthesized. Independently of the implementation in this work, a similar idea has been proposed and successfully used as an optimization in the past [56].

The algorithm behind the parameter extraction is rather simple, and the difficulty comes from the domain specific knowledge of handling `PyTorch` model parameters and generating the correct files for HLS. The break-down of the necessary steps can be seen in algorithm 3.1.

Algorithm 3.1 Mechanism behind model parameter extraction

```
state ←load_state(model)
sort(state)
curr_weight ←null
for param in state do
    mean ←find_mean(model, param)
    var ←find_var(model, param)
    if param is weight then
        curr_weight ←param
        new_param ←update_weight(param, var)
    else
        new_param ←update_bias(param, curr_weight, mean, var)
    end if
    save(new_param)
end for
```

3.2 Hardware Mapping

3.2.1 Tensor Multiplication and Scaling

Each self-attention head performs two tensor multiplications (referred to as *matmul* blocks in figure 2.3), which are normally expressed using Einstein Summation notation [57], 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.

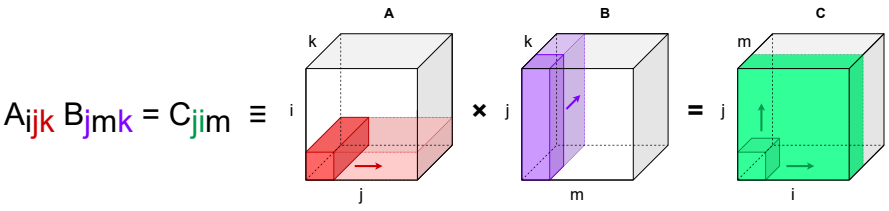

Figure 3.2: Visualization of a tensor operation expressed in Einstein Summation notation.

Figure 3.2 shows a visualization for an example notation to give a better understanding of the necessary flexibility of a formula. The translation between notations using the custom tool is showcased in listing TODO. While the `PyTorch` implementation uses 4-dimensional tensors, the first dimension refers to the batch, which is not present in the hardware implementation that processes input samples one-by-one.

Code listing showing starting `PyTorch` code and resulting C++ HLS implementation

Another simple optimization used alongside the tensor multiplication blocks was the change in size scaling from using division to performing an arithmetic right shift, which requires precomputing the logarithm of the size, seen in equation 3.4, vastly simplifying the otherwise expensive hardware required at run-time.

$$\frac{x}{\sqrt{\text{size}}} \equiv \text{ASR}(x,\ \log_2 \sqrt{\text{size}}) \equiv \text{ASR}(x,\ \frac{1}{2}\log_2 \text{size}) \tag{3.4}$$

3.2.2 Softmax and Log Softmax Activation

Despite an already existing `hls4ml` implementation of the softmax activation function, computing the logarithm of its result is not as simple as it may seem. This is because the numerical stability and computational efficiency of this operation is often explored in-depth [58] and varies depending on the programming language and target platform.

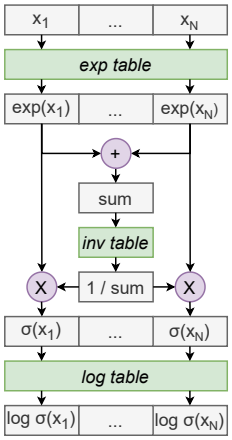

Figure 3.3: Direct hardware implementations of log softmax.

The naive implementation comes straight from the definition of taking a logarithm of softmax, seen in equation 3.5, and the required hardware operations are shown in figure 3.3.

$$\sigma(x_i) = e^{x_i} / \sum_{j=1}^{N} e^{x_j} \tag{3.5}$$

This report 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 3.6.

$$\log(\sigma(x_i)) = log(e^{x_i} / \sum_{j=1}^{N} e^{x_j}) = \log(e^{x_i}) - \log(\sum_{j=1}^{N} e^{x_j}) = e^{x_i} - \log(\sum_{j=1}^{N} e^{x_j}) \tag{3.6}$$

The resulting hardware operations are depicted in figure 3.4. 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 BRAMs or LUTs to allow for lookup on run-time. Hence, the optimized design requires one less of such lookups while also replacing multiplication by a subtraction, which can be simpler to express in hardware.

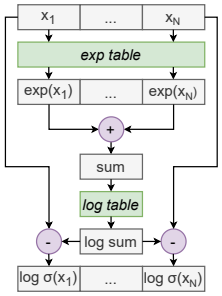

Figure 3.4: Optimized hardware implementations of log softmax.

Although further simplifications, including approximating the summation by finding the maximum (see equation 3.7) or simply omitting the logarithm portion of the expression, were also explored, they noticeably lowered the final accuracy and were thus abandoned.

$$\log(\sigma(x_i)) = e^{x_i} - \log(\sum_{j=1}^{N} e^{x_j}) = e^{x_i} - \sum_{j=1}^{N} \log(e^{x_j}) = e^{x_i} - \sum_{j=1}^{N} x_j \approx e^{x_i} - \max(x) \tag{3.7}$$

3.3 Ultra-Low Latency Architecture

3.3.1 Simplification and Tuning

3.3.2 Hardware Mapping

3.4 Accuracy-Focused Architecture

3.4.1 Hardware Mapping

3.5 Parameter Extraction for Custom Hardware

tool for extracting weight and biases

tool for embedding norm stats for layer norm as running stats not collected

stability issues solved by more normalization (coming from wide range of inputs of 30x16)

Chapter 4

Design Space Exploration

4.1 Pre-training Quantization

PyTorch Eager Mode

PyTorch FX Graph Mode

Brevitas

QPyTorch

4.2 Post-training Quantization

Custom tool

Algorithm 4.2 An algorithm with caption

Require: $n \geq 0$
Ensure: $y = x^n$
  $y \leftarrow 1$
  $X \leftarrow x$
  $N \leftarrow n$
  while $N \neq 0$ **do**
    if N is even **then**
      $X \leftarrow X \times X$
      $N \leftarrow \frac{N}{2}$ ▷ This is a comment
    else if N is odd **then**
      $y \leftarrow y \times X$
      $N \leftarrow N - 1$
    end if
  end while

4.3 High-Level-Synthesis Optimization

ScaleHLS

MLIR

Chapter 5

Evaluation

This chapter outlines the proposed evaluation plan for the project. The first objective of developing and optimizing a state-of-the-art neural network in hardware can be evaluated quantitatively, while integrating it into the *hls4ml* library and making it easy for new users to use requires a more qualitative approach.

Analytical models for latency/resources?

5.1 Quantitative results

The following describes the quantities to be measured for each neural network design:

- Classification accuracy, AUC and confusion matrix on a validation dataset
- Inference latency and throughput when running on the target platform
- Hardware resource utilization (exact values for comparison with other platforms and percentage of available resources for understanding limitations):
  - Block RAM (BRAM) and Ultra RAM (URAM)
  - Digital Signal Processing units (DSP)
  - Flip-Flops (FF)
  - Look-Up Tables (LUT)

In the early stages of the project, the above quantities will be measured from the results from the simulation and synthesis reports. At a later stage, the best designs will be run on actual hardware platforms to validate them under real-life use cases. The platform planned for this part is an Intel Stratix V FPGA hosted in a Maxeler MPC-X dataflow node with 8 Maia dataflow engines and 48 GB of DRAM. A consideration is also planned for the specific hardware used in the LHC L1T detectors and its available resources, which although cannot be directly tested on, can guide the state space exploration.

Apart from clear design improvements, it is predicted that most evaluated designs will offer trade-offs between classification accuracy, AUC, inference throughput and hardware utilization. It is not possible to find a design that is superior in every way, hence a Pareto front and the Roofline model will play the key roles in understanding the overall performance and selecting configuration with specific needs in mind.

5.2 Qualitative Results

To assess the success of enhancing the *hls4ml* library, qualitative comparisons will be drawn between it and the already existing neural network components and architectures. Depending on the project's timeline, it is possible that the improvements can get official approval and get merged

into the main repository, however if this is not feasible before the final deadline, current users of the library will be surveyed and their opinion will be taken into consideration instead.

5.3 Quantization Results

pre-training quantization compared to varying floating-point widths

float16 doesnt learn anything (acc 20%) as its range is too small and we cannot consider normalizing inputs coz its real time system

brevitas only gets 34% accuracy

pytorch quantization is too experimental and doesnt support the model

post-training quantization

somewhere: fuse batch norm to linear???

Chapter 6

Conclusion

Conclude after writing all other sections

6.1 Future Work

Bullet points

References

[1] CERN. Jets at CMS and the determination of their energy scale | CMS experiment, .

[2] Josh Cogan, Michael Kagan, Emanuel Strauss, and Ariel Schwartzman. Jet-images: computer vision inspired techniques for jet tagging. *The journal of high energy physics*, 2015(2):1–16, Feb 18, 2015. doi: 10.1007/JHEP02(2015)118. URL https://link.springer.com/article/10.1007/JHEP02(2015)118.

[3] Luke de Oliveira, Michael Kagan, Lester Mackey, Benjamin Nachman, et al. Jet-images - deep learning edition. *The journal of high energy physics*, 2016(7):1–32, Jul 13, 2016. doi: 10.1007/JHEP07(2016)069. URL https://link.springer.com/article/10.1007/JHEP07(2016)069.

[4] Liam Moore, Karl Nordstrom, Sreedevi Varma, and Malcolm Fairbairn. Reports of my demise are greatly exaggerated: n-subjettiness taggers take on jet images. *SciPost physics*, 7(3):036, Sep 24, 2019. doi: 10.21468/SciPostPhys.7.3.036. URL https://hal.archives-ouvertes.fr/hal-01851157.

[5] Farah Fahim, Benjamin Hawks, Christian Herwig, James Hirschauer, et al. hls4ml: An open-source codesign workflow to empower scientific low-power machine learning devices. Mar 9, 2021. URL https://arxiv.org/abs/2103.05579.

[6] Harvey B. Newman, Avikar Periwal, Maria Spiropulu, Javier M. Duarte, et al. JEDI-net: a jet identification algorithm based on interaction networks. *The European physical journal. C, Particles and fields*, 80(1):1–15, Aug 14, 2019. doi: 10.1140/epjc/s10052-020-7608-4. URL http://cds.cern.ch/record/2688535.

[7] Abdelrahman Elabd, Vesal Razavimaleki, Shi-Yu Huang, Javier Duarte, et al. Graph neural networks for charged particle tracking on FPGAs. Dec 3, 2021. URL https://arxiv.org/abs/2112.02048.

[8] Xinyang Yuan. Constituentnet: Learn to solve jet tagging through attention. Technical report, -09-22 2021.

[9] K. Krasnov and R. Percacci. Gravity and unification: a review. *Classical and quantum gravity*, 35(14):143001, Jun 14, 2018. doi: 10.1088/1361-6382/aac58d. URL https://iopscience.iop.org/article/10.1088/1361-6382/aac58d.

[10] J. Walz, P. Grandemange, B. Vallage, et al. The GBAR antimatter gravity experiment. Technical Report 233, Springer International Publishing, 2015. URL http://cds.cern.ch/record/2055685.

[11] D. Pagano, M. Caccia, J. Fesel, S. Gerber, et al. Gravity and antimatter: the AEgIS experiment at CERN. Technical Report 1342, IOP Publishing, 2020. URL http://cds.cern.ch/record/2714100.

[12] Brian Greeene. How the higgs boson was found, July 2013. URL https://www.smithsonianmag.com/science-nature/how-the-higgs-boson-was-found-4723520/.

[13] Trigger, DAQ and FPGAs. URL http://www.hep.ph.imperial.ac.uk/~tapper/lecture/trigger.pdf.

[14] Sean Keane. CERN's large hadron collider restarts after three-year upgrade. URL https://www.cnet.com/science/cerns-large-hadron-collider-restarts-after-three-year-upgrade/.

[15] Alex Tapper. Triggering at collider experiments. URL http://www.hep.ph.imperial.ac.uk/~tapper/lecture/CMSIndia-2020.pdf.

[16] E. Coleman, M. Freytsis, A. Hinzmann, M. Narain, et al. The importance of calorimetry for highly-boosted jet substructure. *Journal of instrumentation*, 13(1):T01003, Jan 9, 2018. doi: 10.1088/1748-0221/13/01/T01003. URL https://search.proquest.com/docview/2365693051.

[17] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytsis. HLS4ML LHC jet dataset (30 particles), . URL https://doi.org/10.5281/zenodo.3601436.

[18] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytsis. HLS4ML LHC jet dataset (50 particles), . URL https://doi.org/10.5281/zenodo.3601443.

[19] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytsis. HLS4ML LHC jet dataset (100 particles), . URL https://doi.org/10.5281/zenodo.3602254.

[20] Maurizio Pierini, Javier Mauricio Duarte, Nhan Tran, and Marat Freytsis. HLS4ML LHC jet dataset (150 particles), . URL https://doi.org/10.5281/zenodo.3602260.

[21] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. The anti-kt jet clustering algorithm. *The journal of high energy physics*, 2008:063, Apr 1, 2008. doi: 10.1088/1126-6708/2008/04/063. URL http://iopscience.iop.org/1126-6708/2008/04/063.

[22] Edward Kreinar, Jennifer Ngadiuba, Zhenbin Wu, Philip Harris, et al. Fast inference of deep neural networks in FPGAs for particle physics. Technical Report 13, Institute of Physics (IOP), Apr 16, 2018. URL http://cds.cern.ch/record/2316331.

[23] Edward Kreinar, Jennifer Ngadiuba, Zhenbin Wu, Philip Harris, et al. Fast inference of deep neural networks in FPGAs for particle physics. Technical Report 13, IOP Publishing, Apr 16, 2018. URL http://cds.cern.ch/record/2316331.

[24] G. Valentino, R. W. Assmann, R. Bruce, and N. Sammut. Classification of LHC beam loss spikes using support vector machines. pages 355–358. IEEE, Jan 2012. doi: 10.1109/SAMI.2012.6208988. URL https://ieeexplore.ieee.org/document/6208988.

[25] Tianqi Chen and Tong He. Higgs Boson Discovery with Boosted Trees. In Glen Cowan, Cecile Germain, Isabelle Guyon, Balázs Kegl, and David Rousseau, editors, *Proceedings of the NIPS 2014 Workshop on High-energy Physics and Machine Learning*, volume 42 of *Proceedings of Machine Learning Research*, pages 69–80, Montreal, Canada, 13 Dec 2015. PMLR. URL https://proceedings.mlr.press/v42/chen14.html.

[26] Andrzej Skoczen, Maciej Wielgosz, and Matej Mertik. Using LSTM recurrent neural networks for monitoring the LHC superconducting magnets. Technical Report 867, Elsevier B.V, Nov 18, 2016. URL http://cds.cern.ch/record/2234465.

[27] Jie Ren, Lei Wu, and Jin Min Yang. Unveiling CP property of top-higgs coupling with graph neural networks at the LHC. *Physics letters. B*, 802:135198, Mar 10, 2020. doi: 10.1016/j.physletb.2020.135198. URL https://dx.doi.org/10.1016/j.physletb.2020.135198.

[28] Edward Kreinar, Zhenbin Wu, Gianluca Cerminara, Kinga Wozniak, et al. Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics. Technical Report 3, Frontiers Media S.A, 2020. URL http://cds.cern.ch/record/2728798.

[29] Abdelrahman Elabd, Vesal Razavimaleki, Shi-Yu Huang, Javier Duarte, et al. Graph neural networks for charged particle tracking on FPGAs. Dec 3, 2021. URL https://arxiv.org/abs/2112.02048.

[30] Gil Keren and Bjorn Schuller. Convolutional RNN: An enhanced model for extracting features from sequential data. pages 3412–3419. IEEE, Jul 2016. doi: 10.1109/IJCNN.2016.7727636. URL https://ieeexplore.ieee.org/document/7727636.

[31] Jan Chorowski, Dzmitry Bahdanau, Dmitriy Serdyuk, Kyunghyun Cho, et al. Attention-based models for speech recognition. Jun 24, 2015. URL https://arxiv.org/abs/1506.07503.

[32] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, et al. Attention is all you need. Jun 12, 2017. URL https://arxiv.org/abs/1706.03762.

[33] Ashish Vaswani, Samy Bengio, Eugene Brevdo, Francois Chollet, Aidan N. Gomez, Stephan Gouws, Llion Jones, Łukasz Kaiser, Nal Kalchbrenner, Niki Parmar, Ryan Sepassi, Noam Shazeer, and Jakob Uszkoreit. Tensor2tensor for neural machine translation. *CoRR*, abs/1803.07416, 2018. URL http://arxiv.org/abs/1803.07416.

[34] Jay Alammar. The illustrated transformer. URL https://jalammar.github.io/illustrated-transformer/.

[35] Mohammadreza Najafi, Kaiwen Zhang, Mohammad Sadoghi, and Hans-Arno Jacobsen. Hardware acceleration landscape for distributed real-time analytics: Virtues and limitations. pages 1938–1948. IEEE, Jun 2017. ISBN 1063-6927. doi: 10.1109/ICDCS.2017.194. URL https://ieeexplore.ieee.org/document/7980135.

[36] Dagli and Lammers. Possible applications of neural networks in manufacturing. page 605 vol.2. IEEE TAB Neural Network Committee, 1989. doi: 10.1109/IJCNN.1989.118423. URL https://ieeexplore.ieee.org/document/118423.

[37] Cathy Wu, Michael Berry, Sailaja Shivakumar, and Jerry McLarty. Neural networks for full-scale protein sequence classification: Sequence encoding with singular value decomposition. *Machine learning*, 21(1):177, Oct 1, 1995. doi: 10.1023/A:1022677900508.

[38] Vincent Vanhoucke, Andrew Senior, and Mark Z. Mao. Improving the speed of neural networks on CPUs. In *Deep Learning and Unsupervised Feature Learning Workshop, NIPS 2011*, 2011.

[39] Sainathan Ganesh Iyer and Anurag Dipakumar Pawar. GPU and CPU accelerated mining of cryptocurrencies and their financial analysis. pages 599–604. IEEE, Aug 2018. doi: 10.1109/I-SMAC.2018.8653733. URL https://ieeexplore.ieee.org/document/8653733.

[40] Gang Chen, Haitao Meng, Yucheng Liang, and Kai Huang. GPU-accelerated real-time stereo estimation with binary neural network. *IEEE transactions on parallel and distributed systems*, 31(12):2896–2907, Dec 1, 2020. doi: 10.1109/TPDS.2020.3006238. URL https://ieeexplore.ieee.org/document/9130887.

[41] Qianru Zhang, Meng Zhang, Tinghuan Chen, Zhifei Sun, et al. Recent advances in convolutional neural network acceleration. *Neurocomputing (Amsterdam)*, 323:37–51, Jan 5, 2019. doi: 10.1016/j.neucom.2018.09.038. URL https://dx.doi.org/10.1016/j.neucom.2018.09.038.

[42] Graphcore. Graphcore intelligence processing unit. URL https://www.graphcore.ai/products/ipu.

[43] Phil Knag, Jung Kuk Kim, Thomas Chen, and Zhengya Zhang. A sparse coding neural network ASIC with on-chip learning for feature extraction and encoding. *IEEE journal of solid-state circuits*, 50(4):1070–1079, Apr 2015. doi: 10.1109/JSSC.2014.2386892. URL https://ieeexplore.ieee.org/document/7015626.

[44] K. Venkata Ramanaiah and Cyril Prasanna Raj. ASIC implementation of neural network based image compression. *International Journal of Computer Theory and Engineering*, pages 494–498, 2011. doi: 10.7763/IJCTE.2011.V3.356.

[45] Andrew Boutros, Sadegh Yazdanshenas, and Vaughn Betz. You cannot improve what you do not measure. *ACM transactions on reconfigurable technology and systems*, 11(3):1–23, Dec 22, 2018. doi: 10.1145/3242898. URL http://dl.acm.org/citation.cfm?id=3242898.

[46] Eriko Nurvitadhi, Ganesh Venkatesh, Jaewoong Sim, Debbie Marr, et al. Can FPGAs beat GPUs in accelerating next-generation deep neural networks? FPGA '17, pages 5–14. ACM, Feb 22, 2017. doi: 10.1145/3020078.3021740. URL http://dl.acm.org/citation.cfm?id=3021740.

[47] Yixing Li, Zichuan Liu, Kai Xu, Hao Yu, et al. A GPU-outperforming FPGA accelerator architecture for binary convolutional neural networks. *ACM journal on emerging technologies in computing systems*, 14(2):1–16, Jul 27, 2018. doi: 10.1145/3154839. URL http://dl.acm.org/citation.cfm?id=3154839.

[48] Eriko Nurvitadhi, David Sheffield, Jaewoong Sim, Asit Mishra, et al. Accelerating binarized neural networks: Comparison of FPGA, CPU, GPU, and ASIC. pages 77–84. IEEE, Dec 2016. doi: 10.1109/FPT.2016.7929192. URL https://ieeexplore.ieee.org/document/7929192.

[49] Eriko Nurvitadhi, Jaewoong Sim, David Sheffield, Asit Mishra, et al. Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC. pages 1–4. EPFL, Aug 2016. doi: 10.1109/FPL.2016.7577314. URL https://ieeexplore.ieee.org/document/7577314.

[50] Sakari Lahti, Panu Sjovall, Jarno Vanne, and Timo D. Hamalainen. Are we there yet? a study on the state of high-level synthesis. *IEEE transactions on computer-aided design of integrated circuits and systems*, 38(5):898–911, May 2019. doi: 10.1109/TCAD.2018.2834439. URL https://ieeexplore.ieee.org/document/8356004.

[51] CERN. Facts and figures about the LHC | CERN, . URL https://home.cern/resources/faqs/facts-and-figures-about-lhc.

[52] R. Guida, M. Capeans, and B. Mandelli. Characterization of RPC operation with new environmental friendly mixtures for LHC application and beyond. *Journal of Instrumentation*, 11(07):C07016–C07016, jul 2016. doi: 10.1088/1748-0221/11/07/c07016. URL https://doi.org/10.1088/1748-0221/11/07/c07016.

[53] M. Capeans, R. Guida, and B. Mandelli. Strategies for reducing the environmental impact of gaseous detector operation at the CERN LHC experiments. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 845:253–256, 2017. doi: https://doi.org/10.1016/j.nima.2016.04.067. URL https://www.sciencedirect.com/science/article/pii/S0168900216302807. ID: 271580.

[54] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. pages 770–778. IEEE, Jun 2016. doi: 10.1109/CVPR.2016.90. URL https://ieeexplore.ieee.org/document/7780459.

[55] Christian Szegedy, Sergey Ioffe, Vincent Vanhoucke, and Alex Alemi. Inception-v4, inception-resnet and the impact of residual connections on learning. Feb 23, 2016. URL https://arxiv.org/abs/1602.07261.

[56] Hongxiang Fan, Shuanglong Liu, Martin Ferianc, Ho-Cheung Ng, et al. A real-time object detection accelerator with compressed SSDLite on FPGA. pages 14–21. IEEE, Dec 2018. doi: 10.1109/FPT.2018.00014. URL https://ieeexplore.ieee.org/document/8742299.

[57] Alan H. Barr. The einstein summation notation. *An Introduction to Physically Based Modeling (Course Notes 19), pages E*, 1:57, 1991.

[58] Pierre Blanchard, Desmond J. Higham, and Nicholas J. Higham. Accurate computation of the log-sum-exp and softmax functions. *arXiv preprint arXiv:1909.03469*, 2019.

Appendices

Appendix A

Something

something

Notes