TrackCore-F: Deploying Transformer-Based Subatomic Particle Tracking on FPGAs

Arjan Blankestijn[1], Uraz Odyurt[2⋆] and Amirreza Yousefzadeh[1]

1 Computer Architecture for Embedded Systems, University of Twente, Enschede, The Netherlands
2 Faculty of Engineering Technology, University of Twente, Enschede, The Netherlands

⋆ u.odyurt@utwente.nl

The 2nd European AI for Fundamental Physics Conference (EuCAIFCon2025)
Cagliari, Sardinia, 16-20 June 2025

Abstract

The Transformer Machine Learning (ML) architecture has been gaining considerable momentum in recent years. In particular, computational High-Energy Physics tasks such as jet tagging and particle track reconstruction (tracking), have either achieved proper solutions, or reached considerable milestones using Transformers. On the other hand, the use of specialised hardware accelerators, especially FPGAs, is an effective method to achieve online, or pseudo-online latencies. The development and integration of Transformer-based ML to FPGAs is still ongoing and the support from current tools is very limited or non-existent. Additionally, FPGA resources present a significant constraint. Considering the model size alone, while smaller models can be deployed directly, larger models are to be partitioned in a meaningful and ideally, automated way. We aim to develop methodologies and tools for monolithic, or partitioned Transformer synthesis, specifically targeting inference. Our primary use-case involves two machine learning model designs for tracking, derived from the TrackFormers project. We elaborate our development approach, present preliminary results, and provide comparisons.

1 Introduction

The move to ML-assisted tracking solutions is deemed necessary and inevitable. While design efforts are moving forward, another important aspect to consider is deployment. The accurate tracking has been a post-mortem task, which is due to the computational requirements. While the introduction of ML algorithms will improve the computational performance, the increase in scale and frequency of experiments will somewhat counter and reduce effects of such improvements. This is especially the case for the scales expected from the upcoming High-Luminosity stage of the LHC (HL-LHC). On top of that, efficient execution of ML algorithms has predominantly been tied to Graphics Processing Units (GPUs) as the platform of choice. While not a universal dependency, GPUs dominate the ML deployment scene. Nevertheless, there are viable alternative forms of hardware acceleration, e.g., Field-Programmable Gate Array (FPGA), custom Application-Specific Integrated Circuit (ASIC), and Neuromorphic. We aim to deploy ML-assisted track reconstruction on FPGAs to achieve better or equivalent

latencies. Additionally, FPGAs enable on-site deployment of tracking and entail considerable energy efficiency potential.

Related work Deployment of models based on the Transformer architecture on hardware, especially FPGAs, has been a point of interest for the research community. This is especially noticeable during the past and the current year, 2024–2025. Focusing on the optimisations, 4 main trends are observable, namely: approaches focusing on quantization techniques [1–3], approaches utilising off-chip memory [4], approaches leveraging sparsity through pruning [5–7], and approaches based on optimisation of non-linear functions such as SoftMax [8].

One of the important metrics to consider when evaluating implementations is *throughput*. At the time of this writing, the most promising approaches demonstrating the highest throughput are [1] and [9], with the former achieving noticeably higher throughput.

2 Background

At the Large Hadron Collider (LHC), particles are accelerated in opposite directions in a circular accelerator and made to collide at four interaction points, where large-scale detectors are positioned. These major detectors are ALICE [10], ATLAS [11], CMS [12], and LHCb [13]. Detectors perform two key functions: *tracking* and *calorimetry*, allowing the calculation of particle momentum, p and measuring the energy, E, deposited by particles, respectively. Together, these measurements enable the calculation of a particle's mass, m, using the relativistic energy-momentum relation: $E^2 = (mc^2)^2 + (pc)^2$, where c is the speed of light. Accurately determining particle mass is essential for identifying known particles and discovering new ones.

2.1 Tracking algorithms

There has been continuous efforts channelled into the design and development of ML-based, or rather ML-assisted, tracking solutions. Two ML model architectures stand out: Graph Neural Networks (GNNs) and more recently, Transformers [14]. Within the scope of this paper, we focus on two Transformer designs from the project TrackFormers [15], *EncCla* and *EncReg*. Both models operate as so-called single-shot models, i.e., they take in a full event's data and perform hit to track association for the whole event.

The Encoder-Classifier (EncCla) is an encoder-only Transformer design. This approach takes in hit coordinates from an event and predicts their association to pre-defined class labels. Class labels, ensured to be unique, are generated through binning of the track parameter space. The largest variant has close to 1.5 million parameters with estimated memory consumptions of 5.69 MB and 0.07 MB for parameters and activations, respectively.

The Encoder-Regressor (EncReg) is also an encoder-only Transformer design. It does not rely on class labels, but instead regresses the parameters of potential tracks for a single event. As a post-processing step, a clustering algorithm has to be applied to model's output. As such, predicted track parameters per hit are clustered, forming track associations. EncReg uses HDBSCAN to achieve this clustering. The model has close to 76 484 parameters with estimated memory consumptions of 0.29 MB and 0.07 MB for parameters and activations, respectively.

2.2 Datasets

The two model designs have been trained with 5 different datasets, forming a progression of simple to complex representations of tracks and hits, i.e., track function complexity, track

count, and by extension, hit count: 10–50 (variable count) linear tracks per event (REDVID), 10–50 (variable count) helical tracks per event (REDVID), 50–100 (variable count) helical tracks per event (REDVID), 10–50 (variable count) tracks per event (TrackML), 200–500 (variable count) tracks per event (TrackML). The first three datasets are the result of simulations using REDVID simulation framework [16]. The last two datasets are scale-reduced versions of the data associated with the TrackML Kaggle challenge [17].

3 Implementation and results

The utilised test bench is an ARM Zynq UltraScale+ MPSoC ZCU102 evaluation kit. The on-board EG device (ZU9EG) consists of a Quad-Core ARM Cortex-A53 Processing Unit (PU) and a Programmable Logic (PL) with the following specification [18]: System Logic Cells 599 550, Configurable Logic Block (CLB) Flip-Flops 548 160, CLB LUTs 274 080, Distributed RAM (Mb) 8.8, Block RAM (Mb) 32.1, DSP Slices 2 520.

A variety of tools must be used in sequence to obtain a deployable synthesised kernel. In particular, for pre-trained ML models, the following have been used:

- PyTorch: Models are provided in the PyTorch format, which could optionally be used for quantization.
- ONNX: Open Neural Network Exchange (ONNX) is an open-source format for representing ML models, which reveals low-level operations, input/output dimensions, data types, and weight Tensor values (if present) per operation. The ONNX format is utilised for quantization as well.
- AMD Vitis HLS 2022.2: Vitis High-Level Synthesis (HLS) is a tool from AMD (previously Xilinx), enabling C/C++, or OpenCL descriptions of hardware to be synthesised into Register Transfer Level (RTL) implementations targeting FPGAs.
- AMD Vivado 2022.2: Vivado Design Suite is used to integrate and configure the IP designed using Vitis HLS with the Zynq MPSoC and to synthesise the final bitstream.
- PYNQ: Python Productivity for Zynq (PYNQ) is a Python-based development environment designed for AMD's Zynq SoCs. PYNQ enables interaction with the Zynq PL.

3.1 Development flow

As depicted in Figure 1, our deployment workflow is comprised of different steps involving the aforementioned tools.

First, a pre-trained PyTorch model is converted to the ONNX format. The selected model is trained on the first dataset from Section 2.2. Working with an ONNX model requires the model itself, plus the input data shape. Considering the model graph of low-level computational operations (through ONNX format), one can opt for a full or a partial model deployment. A full deployment is conditional to model size and PL resource availability.

Currently, we apply a manual partitioning, which results in a split model, including the slice to be synthesised as a kernel. The output from the preceding layer is passed on to the slice kernel. In return, the output from the kernel is being fed to the following layer, fusing the dataflow between model partitions. Vitis HLS is used to develop the kernel in C/C++. Once functionality is verified with behavioural simulation, the kernel is synthesised into a hardware IP. The resulting IP is imported into Vivado where it is integrated with the MPSoC block and connected with peripherals such as the AXI4 communication bus. Finally, using Vivado, the

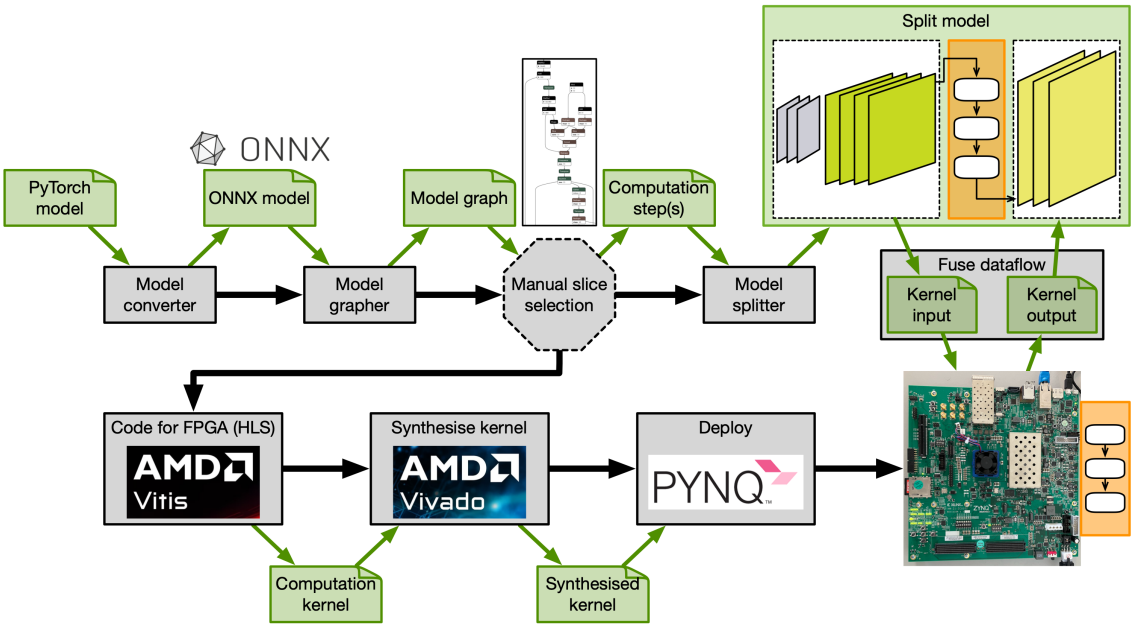

Figure 1: The development flow describing the handling of pre-trained ML models and preparations for selective slice deployment on a FPGA.

complete design is synthesised and a bitstream generated. Note that our current workflow relies on the Vivado IP Flow to integrate our designed IP into the rest of the peripherals using Vivado. Optionally, the (more restrictive) alternative Vitis Kernel Flow can be used to directly synthesise the final output.

Focusing on a single encoder layer, the model is split into three parts: the segment before the first encoder layer, the encoder layer to be implemented and deployed on the FPGA, and the remaining model segment. The encoder kernel block is developed using HLS C/C++ in Vitis. The top-level function takes all the weights of the encoder layer as input. These weights are transferred to the kernel using an AXI4-Lite interface. Once all the parameters are loaded in, the computations for the encoder layer take place. This consists of the Multi-Head Attention (MHA), followed by the first `Add AND Normalisation`, a feed-forward layer, and finally a last `Add AND Normalisation` layer. HLS Pragmas are used to optimise performance.

When designing a kernel for the PL, we are not required to implement ONNX operation blocks individually. In fact, it often reduces CLB and memory utilisation if a selected segment can be developed into a monolithic kernel. To make the transition and porting smooth, we first develop the ONNX model slice using low-level Python code, i.e., NumPy and basic arithmetic.

3.2 Quantization effects

The current implementation makes use of floating-point weights and activations. A common way to increase performance and reduce resource consumption is to employ quantization, often at `INT8`, for both weights and activations. However, this can greatly affect the overall accuracy of the model. ONNX Static quantization available in the ONNX Python package was used to investigate the effect on the accuracy. Table 1 shows the resulting model accuracies at different quantization configurations. We can conclude that quantizing the activations has a bigger impact on the model accuracy, while quantizing just the weights has a lesser impact.

There are more combinations to try for a comprehensive benchmarking, e.g., not quantizing the activations and only using quantized `INT8` weights. ONNX Static quantization does not support this configuration and implementation is more complicated.

Table 1: Model prediction accuracies with different quantization levels applied. The base model has an overall prediction accuracy of 0.97.

Model activations	Model weights	Prediction accuracy
`INT16`	`INT16`	0.90
`INT16`	`INT8`	0.90
`INT8`	`INT16`	0.71
`INT8`	`INT8`	0.70

3.3 Resource consumption

The FPGA has a limited amount of available hardware resources, in particular, Block RAMs (BRAMs), Digital Signal Processing slices (DSPs), Flip-Flops (FFs), and Look-Up Tables (LUTs). Table 2 lists the resource consumption of a single encoder layer kernel compared to the available resources on the ZCU102.

Table 2: ZCU102 FPGA resource utilisation for a single encoder layer, as reported by Vitis.

Resource type	Used	Total available	Utilisation (%)
36 Kb BRAM	347	912	38.04
DSP	67	2 520	2.66
FF	74 184	548 160	13.53
LUT	65 815	274 080	24.01

As it can be seen, the limiting factor is the available BRAM, which has a utilisation of 38.04%. However, the number of used BRAMs can be reduced effectively by considering more DDR memory at the expense of increased memory transfer and latency. Thus, given that the BRAM utilisation can be reduced, the second limiting factor will be the number of LUTs, which is at 24.01% utilisation for a single encoder layer. This suggests that a maximum of 4 encoder layers can be implemented on this particular FPGA and for this model.

4 Conclusion

We have presented a structured development flow for the deployment of pre-trained tracking models on FPGAs. A partial deployment is as feasible as a full deployment. Although not as performant, we consider a partial deployment as valuable as a full one, since it enables model inference to be deployed on more accessible hardware. We observe that the application of optimisations can be more important than the HLS implementation itself. We have also shown how detrimental quantization can be to the model's prediction accuracy. Optimisations and their effects are highly dependent on the model and the HLS implementation.

Acknowledgements

This publication is part of the project ZORRO with project number KICH1.ST02.21.003 of the research programme Key Enabling Technologies (KIC), which is (partly) financed by the Dutch Research Council (NWO).

References

[1] Q. Guo, J. Wan, S. Xu, M. Li and Y. Wang, *HG-PIPE: Vision Transformer Acceleration with Hybrid-Grained Pipeline*, In *Proceedings of the 43rd IEEE/ACM International Conference on Computer-Aided Design*, doi:10.1145/3676536.3676681 (2025).

[2] P. Zhao, D. Xue, L. Wu, L. Chang, H. Tan, Y. Han and J. Zhou, *HEAT: Efficient Vision Transformer Accelerator With Hybrid-Precision Quantization*, IEEE Transactions on Circuits and Systems II: Express Briefs (2025), doi:10.1109/TCSII.2025.3547340.

[3] C. Du, S.-B. Ko and H. Zhang, *Energy Efficient FPGA-Based Binary Transformer Accelerator for Edge Devices*, In *2024 IEEE International Symposium on Circuits and Systems (ISCAS)*, doi:10.1109/ISCAS58744.2024.10558631 (2024).

[4] E. Kabir, J. D. Bakos, D. Andrews and M. Huang, *A Runtime-Adaptive Transformer Neural Network Accelerator on FPGAs*, doi:10.48550/arXiv.2411.18148 (2025).

[5] M. Zhang, J. Cao, K. Shi, K. Zhao, G. Zhang, J. Yu and K. Wang, *FNM-Trans: Efficient FPGA-based Transformer Architecture with Full N:M Sparsity*, In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, doi:10.1145/3649329.3656497 (2024).

[6] S. Wang and H. Zhang, *Efficient FPGA-Based Transformer Accelerator Using In-Block Balanced Pruning*, In *2024 13th International Conference on Communications, Circuits and Systems (ICCCAS)*, doi:10.1109/ICCCAS62034.2024.10651591 (2024).

[7] Z. Li, Y. Lai and H. Zhang, *Energy Efficient FPGA-Based Accelerator for Dynamic Sparse Transformer*, In *2024 13th International Conference on Communications, Circuits and Systems (ICCCAS)*, doi:10.1109/ICCCAS62034.2024.10652850 (2024).

[8] Z. Bai, P. Dangi, H. Li and T. Mitra, *SWAT: Scalable and Efficient Window Attention-based Transformers Acceleration on FPGAs*, In *Proceedings of the 61st ACM/IEEE Design Automation Conference*, doi:10.1145/3649329.3658488 (2024).

[9] Z. He, X. Jin and Z. Xu, *F3: An FPGA-Based Transformer Fine-Tuning Accelerator With Flexible Floating Point Format*, IEEE Journal on Emerging and Selected Topics in Circuits and Systems (2025), doi:10.1109/JETCAS.2025.3555970.

[10] T. A. Collaboration, *The ALICE experiment at the CERN LHC*, Journal of Instrumentation (2008), doi:10.1088/1748-0221/3/08/S08002.

[11] T. A. Collaboration, *The ATLAS Experiment at the CERN Large Hadron Collider*, Journal of Instrumentation (2008), doi:10.1088/1748-0221/3/08/S08003.

[12] T. C. Collaboration, *The CMS experiment at the CERN LHC*, Journal of Instrumentation (2008), doi:10.1088/1748-0221/3/08/S08004.

[13] T. L. Collaboration, *The LHCb Detector at the LHC*, Journal of Instrumentation (2008), doi:10.1088/1748-0221/3/08/S08005.

[14] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser and I. Polosukhin, *Attention is All You Need*, In *Proceedings of the 31st International Conference on Neural Information Processing Systems* (2017).

[15] S. Caron, N. Dobreva, A. Ferrer Sánchez, J. D. Martín-Guerrero, U. Odyurt, R. Ruiz de Austri Bazan, Z. Wolffs and Y. Zhao, *TrackFormers: In Search of Transformer-Based Particle Tracking for the High-Luminosity LHC Era*, The European Physical Journal C (2025), doi:10.1140/epjc/s10052-025-14156-3.

[16] U. Odyurt, S. N. Swatman, A.-L. Varbanescu and S. Caron, *Reduced Simulations for High-Energy Physics, a Middle Ground for Data-Driven Physics Research*, In *Computational Science – ICCS 2024*, doi:10.1007/978-3-031-63751-3_6 (2024).

[17] Kiehn, Moritz, Amrouche, Sabrina, Calafiura, Paolo, Estrade, Victor, Farrell, Steven and et al., *The TrackML high-energy physics tracking challenge on Kaggle*, EPJ Web Conf. (2019), doi:10.1051/epjconf/201921406037.

[18] AMD Technical Information Portal, *Zynq UltraScale+ MPSoC Data Sheet: Overview*, AMD.