Master thesis
Data Science

Radboud University

Translating Hits to Tracks: Transforming High Energy Physics Experiments

Author:
Nadezhda Dobreva
s1033115

First supervisor/assessor:
Dr. Sascha Caron
scaron@nikhef.nl

Second assessor:
Dr. Inge Wortel
inge.wortel@ru.nl

April 9, 2024

Abstract

Track reconstruction is a crucial part of High Energy Physics (HEP) experiments. Traditional methods such as the Kalman Filter are utilized successfully, but scale poorly and are not easily parallelizable, leaving room for improvement. This makes machine learning and deep learning appealing alternatives. Following the success of Transformers in the field of language processing, we investigate the feasibility of training a Transformer to translate detector signals into track parameters. We perform the association of hits to tracks using an encoder-only model that regresses track parameter values for each hit in an event, followed by clustering in the resulting track parameter space.

The proposed architecture is benchmarked on simplified datasets generated by the recently developed simulation framework REDuced VIrtual Detector (REDVID) as well as subsets of the TrackML data. The results show promise for the application of this deep learning technique on more realistic data for track reconstruction, and we suggest improvements to make it a potential faster alternative to currently utilized algorithms.

Keywords: High-energy physics · Machine learning · Transformer · Hit clustering · Track reconstruction · Track finding

Contents

Chapter 1

Introduction

The Large Hadron Collider (LHC) at CERN is the world's largest and highest-energy particle accelerator [21]. The experiments conducted in LHC involve colliding beams of particles in a particle detector. The collision of two particles generates secondary particles, the movement of which leaves signals on the detectors within the chamber. Track reconstruction refers to the task of rebuilding the trajectories of the secondary particles from the detector signals (i.e., hits). It is a challenging task that plays a vital part in High Energy Physics (HEP) experiments. It helps determine properties of the particles such as momentum and charge, which in turn are used to analyze and identify exotic particles, and even test fundamental laws of physics [49].

The traditional method to conduct the task of reconstructing tracks involves combinatorial Kalman Filters (KF): an algorithm that finds all possible combinations of hits and makes candidate clusters from those that form a sufficiently long track [48]. However, this approach scales quadratically or worse with detector occupancy [11] and is inherently sequential. Especially with the hardware update of the LHC as it enters its High-Luminosity stage in the near future, the number of interactions and generated secondary particles per collision will be increased tenfold [49], motivating the need to move away from combinatorial algorithms and work on a solution that can run faster and in parallel, e.g., using deep learning. Neural networks (NN) are furthermore suitable for the task because of their ability to model complex pattern recognition problems with non-linear dynamics.

A number of methods utilizing NNs have already been explored for trajectory reconstruction. We add to this body of work by investigating the performance of an architecture never before considered for the task of track reconstruction: namely, the Transformer. Transformers are models that find patterns and meaning by tracking relationships in sequential data, using the attention mechanism which enables them to focus on certain parts of the data and ignore other currently irrelevant parts [50]. Their appeal is manifold: they are nonsequential models, indifferent to input order, and able to work well with variable length inputs. These are all properties fitting the task at hand. Furthermore, during inference the time and space complexity of the Transformer model can be brought to sub-quadratic in the number of tracks, which makes Transformers good candidates for a solution faster than Kalman Filters.

We explore a number of approaches to utilizing the Transformer architecture, but this paper focuses on one in particular: using the model for translation, similarly to its original purpose, but in a HEP context. We attempt to translate detector information into physics information, by predicting continuous values in an attempt to directly regress the track defining parameters of the hits. This is followed by a clustering step in the track parameter space to group the hits with similar track parameters together. Thus, we tackle the question of how suitable the Transformer Regressor is for associating hits to tracks, as part of the track reconstruction task. To answer this, we investigate the physics and time performance and their scalability on a number of simulated datasets with increasing complexity and size.

In the next sections we outline the background of HEP experiments and the tracking problem in particular (Chapter 2), elaborate the inner workings of the Transformer (Chapter 3), review related work (Chapter 4), explain our methods, simulated data, and pipeline design choices (Chapter 5), present the experimental setup and results, and discuss their implications (Chapters 6 and 7), and draw conclusions (Chapter 8).

Chapter 2

Particle Track Reconstruction

HEP experiments refer to particle collision events: beams of high energy subatomic particles get accelerated nearly to the speed of light and their interaction upon contact gets recorded for analysis. The data allows physicists to gain knowledge about the behaviour of the particles, and reveal new physics beyond the Standard Model[1]. The world's largest and most powerful tool for particle acceleration is the LHC at CERN.

2.1 Large Hadron Collider

Two types of collisions are investigated at LHC: proton-proton and ion-ion. Beams of particles get accelerated down a 27 kilometer long beam pipe in opposite directions and then forced to collide (a.k.a. an *event*). The collision creates new unstable particles that fly outwards, and can decay and produce sprays of even more particles. These outgoing secondary particles are recorded by complex detector systems. There are four detectors installed at LHC, namely ATLAS [15], ALICE [14], CMS [16], and LHCb [17].

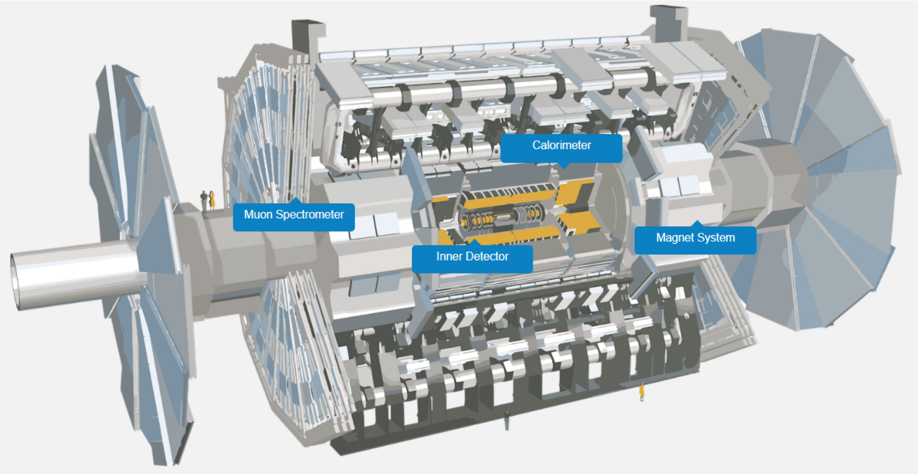


Figure 2.1: A schematic of the ATLAS detector and its components. Image credit: CERN.

[1] The Standard Model of particle physics refers to the theory describing the known fundamental forces and classifying all known elementary particles [24].

For instance, the ATLAS detector consists of multiple subsystems situated concentrically in layers around the collision point [4]. A visualization of the detector is shown in Figure 2.1. The inner detector is the first point of detection at ATLAS. It consists of layers of over a hundred million minuscule silicon pixels, which detect the small energy deposits left behind by charged particles passing through them after bursting out of the collision point. The calorimeter is designed to absorb the particles formed at collision, effectively measuring the energy they lose, and bring them to a stop with layers of high-density material. The outer layer of the detector is the muon spectrometer, made up of muon detectors that identify and measure the momentum of muons – one of the only particles that cannot be stopped by the calorimeter. ATLAS also has a huge magnet system that bends the trajectories of the decaying particles, so that their momentum and charge can be measured as precisely as possible.

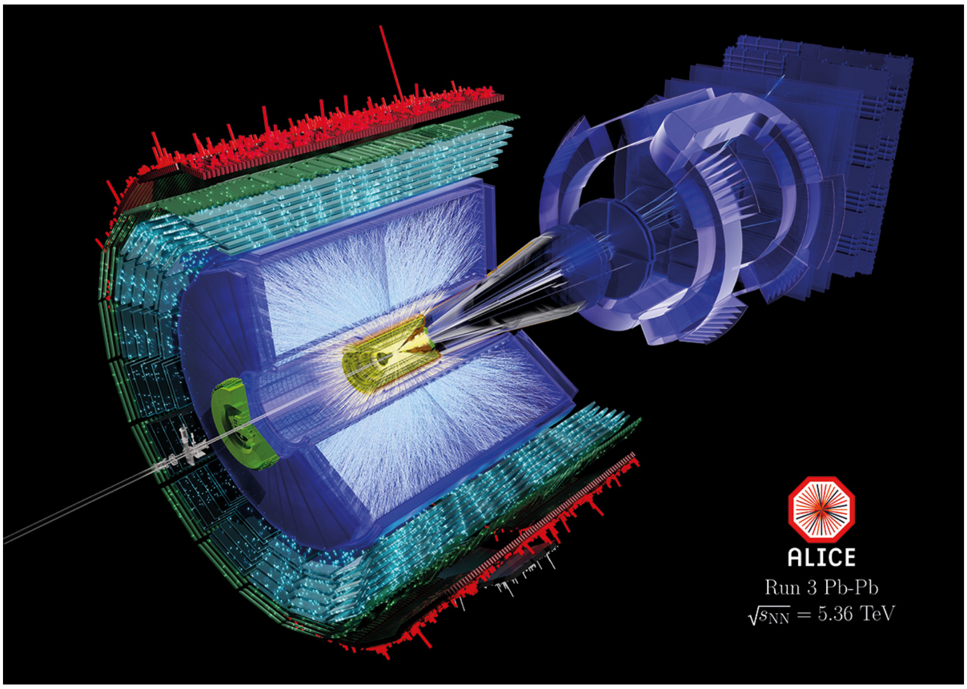


Figure 2.2: Sketch of an event at LHC: the collision and generation of secondary particles which pass through the detector layers. Image credit: ALICE-PHO-GEN-2022-009-7.

When in operation, the LHC produces tens of millions of collisions per second, but due to storage constraints only a few thousand of them are stored. There is a real-time data selection algorithm, the "trigger," that decides whether the collision generated is of interest and should be saved [41]. The particles' *hits* on the detector sensors are stored: e.g., in the case of the inner detector, the hits are the coordinates in physical space of the silicon pixels which recorded an energy deposit during the event. A visualization of an event at the detector ALICE is presented in Figure 2.2.

Based on this data, researchers can conduct analysis with many different purposes, e.g., to identify the generated particles, and discover new ones. One important task of HEP experiments is tracking.

2.2 Track Reconstruction

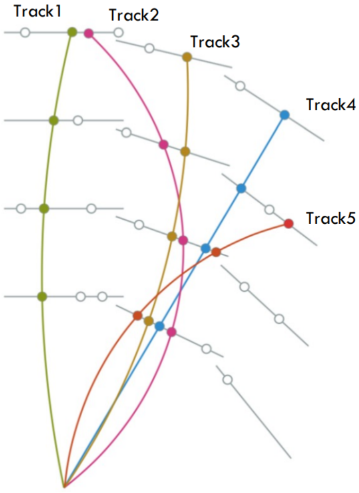


Figure 2.3: A simplified example of tracking: all hits (dots) recorded by the adjacent chunks of detector layers (gray lines), with those grouped into reconstructed tracks colored accordingly. Image credit: [7].

The subatomic particle track reconstruction challenge, a.k.a. *tracking*, involves the reconstruction and tracing of a particle's trajectory, given the sensory data from the layers of detectors (see Figure 2.3). Tracking is crucial to the study of HEP collision experiments and the nature of generated particles as it enables calculating the momentum of a particle, and is present in all major LHC experiments. It can happen at trigger time (online), or based on the stored data (offline).

The first challenging aspect of the tracking problem is the large size of recorded data: in total there are thousands of hits captured by the complex detector system. Since the detectors comprise physical silicon sensors, the hits get discretized, which can be seen as noise in their recording. In the case of multiple particles passing through the same pixel in the same event, only one hit will be recorded. Thus, the particles do not have well-separated hits, and some are even missing. Furthermore, under the influence of the magnetic field, the momentum of the generated particles is modified and their trajectories curve, becoming roughly helical [36]. The particles do not always originate from the same place: in the case of a secondary particle decaying into other particles, the point of origin is not the same as the collision point. Lastly, each event produces an arbitrary amount of particles, meaning that the number of recorded hits per collision varies. All of these complications make the task rather difficult. It will become even more complex in the next stage of the LHC: the High Luminosity stage. With its deployment, there will be a higher frequency of events, a higher amount of collisions collected, a higher amount of tracks generated, more pile-up and more contamination by hits from previous and following events. The hardware of the LHC is currently being upgraded to capture these changes.

Usually, the tracking task is split into two main steps [48]. First is the more complex step of track finding – a pattern recognition problem concerned with grouping together hits likely originating from the same particle. Then track fitting: the derivation of *track parameters* from each group of hits, where track parameters are the basic parameters which describe the trajectory of a particle[2]. The model with best overall performance for both steps is the Kalman Filter [2] and it is the approach currently being utilized at the LHC. However, this makes the tracking challenge notoriously compute-intensive at scale.

2.3 Kalman Filters

The KL is an algorithm which determines the internal state of a linear dynamic system by recursively processing discrete measurements, with random perturbations present in the

[2] For example, a linear track in 2D space is uniquely described by its slope, so that is its track parameter.

measurements and the system itself [34]. Both stages of tracking are based on KL [9], but the track finding phase makes use of a specific version, called the combinatorial KL (CKL). It starts with a track seed (a recorded hit) and extrapolates and updates it with the information of the following hits on that track. The next hit is selected out of a pool of hit candidates which are all evaluated and scored against each other, so that the most fitting one is chosen (e.g., using geometrical properties). The extrapolation and update procedure are repeated for each possible candidate hit, and to make this step more efficient in terms of computing time, the set of candidates is limited wherever possible by applying different filters. The track finding stage with the help of a CKL is also illustrated in Figure 2.4.

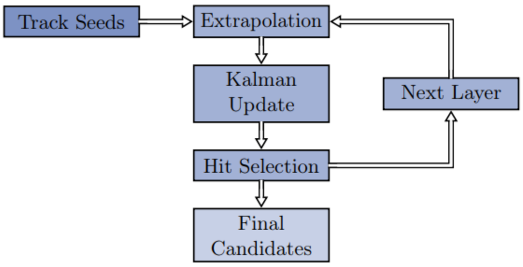


Figure 2.4: The procedure of the CKL used for track finding [9]. The extrapolation, Kalman update and hit selection step are repeated for each hit on the next layer, and this procedure repeats as long as there are remaining detector layers the track candidate can extrapolate to.

In the track fitting stage, a standard KL algorithm is used but the overall procedure is similar. The KL unit calculates a rough estimate of the track using an initial seed hit, then propagates the track parameters and their uncertainties from one detector layer to the next, updating them with information from the rest of the hit measurements. The unit is iteratively applied on all hits associated with the track for a good estimation of the trajectory. To obtain an even better estimation, the result of a former run of KL can be used as the seed for the next one to smooth out the track parameter estimate.

Variations of Kalman Filtering have been used at LHC for decades [29, 22, 9] due to its robustness and excellent physics performance. However, its combinatorial nature in the track finding stage and inherent sequential execution make KL unsuitable for the upcoming High Luminosity stage of the LHC, as it scales very poorly with the recorded number of hits. The currently utilized algorithm in-house is tested on a simulated data of events with 200 points of origin of tracks (i.e., pile-up) [13], which is roughly the same complexity as the largest data we evaluate our proposed approaches on in this paper. The KL pipeline takes 214.3 HS06 × seconds for a single event, which translates to around 12s CPU time[3]. A proposed optimized algorithm with a tighter track selection in the track finding stage achieves 7x speedup, needing 1.8s CPU time per event [13].

[3] The CPU time is multiplied with the HS06 factor of 17.8 for single thread running, since the used CPU is Intel Xeon E5-2620v2. HS06 is a (now no longer used) HEP-wide benchmark for measuring CPU performance, meant to provide a consistent and reproducible measurements to describe experiment requirements. More information about it can be found on the official website: https://w3.hepix.org/benchmarking/HS06.html.

Chapter 3

Transformers

The Transformer is a deep learning architecture, which rose to fame with its success in tasks related to natural language processing [25, 12], and has since found applications in many other areas as well [31, 43]. The traditional Transformer has an encoder-decoder architecture [50] as seen in Figure 3.1.

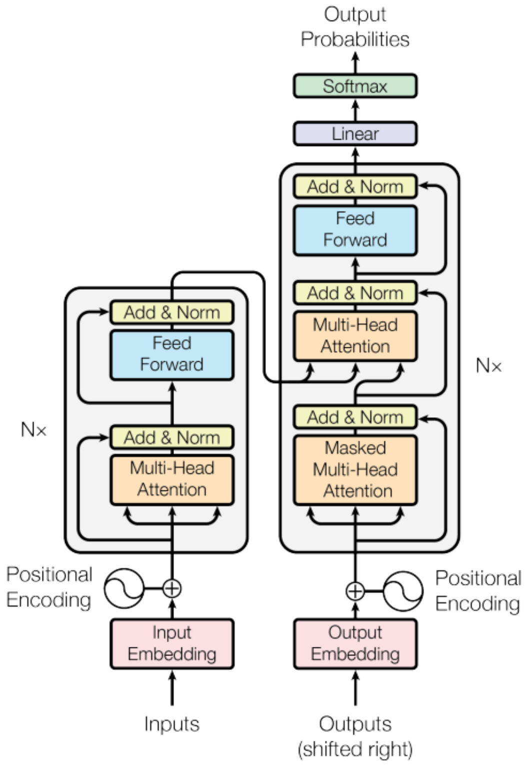


Figure 3.1: The Transformer architecture, as originally introduced [50].

The model's input sequence consists of *tokens* $x_1, x_2, ..., x_n$ and the initial embedding layer creates their representations by mapping each token to a continuous numerical value. Optionally, positional encoding is applied, which essentially ensures the position of each representation in the input does not get lost. Next, the continuous values are fed into the encoder of the Transformer, which comprises a number of identical encoder layers. Each encoder layer consists of the multi-head attention operation, normalization and a feed forward network (i.e., composed of fully connected layers).

The attention mechanism allows for the model to tend to certain parts of the data and ignore other, less relevant ones, to embed the sequence context into the token representations. It can be described as operations done on three vectors: the key (K), value (V) and query (Q) vectors, which are all derived from the input to the encoder block in the first encoder layer, and from the output of the previous layer for every other encoder layer. Q represents the elements for which we want to compute the attention scores with respect to the other elements in the sequence. K is responsible for providing information about the other elements in the sequence, and that helps determine which of them are important in relation to the current element when computing the attention scores. Lastly, V contains learned representations of the content of each element in the input. In the scaled dot-product attention, Q and K are multiplied to capture the dependencies within the input sequence, creating attention weights, which are then used to capture the relevant information content from V. The attention function is displayed in Figure 3.2 and is given by the following equation:

$$Attention(Q,K,V) = softmax(\frac{QK^T}{\sqrt{d_k}})V$$

where d_k is the dimensionality of the lower-dimension linear projection of K.

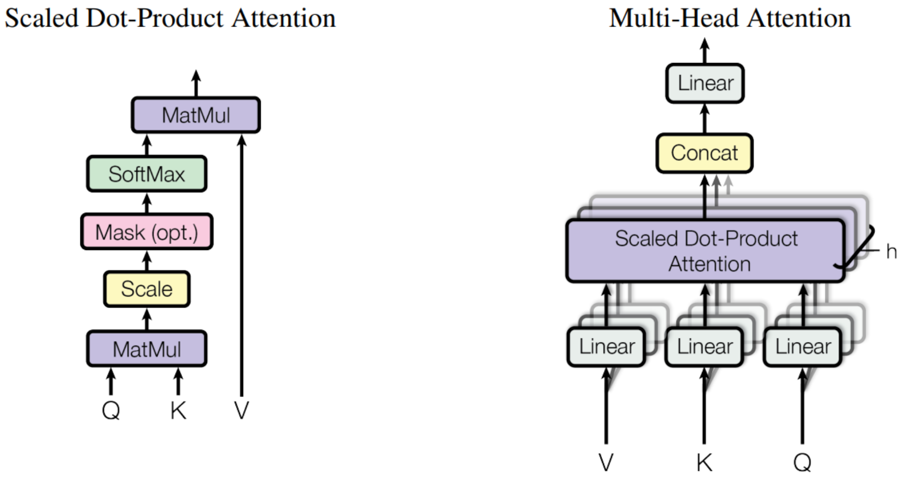


Figure 3.2: The attention function (left), and multi-head attention (right) as originally introduced [50].

Typically, attention is computed on a set of queries at the same time, making K, V and Q matrices. Multi-head attention (MHA) allows for jointly attending to information from different representation subspaces at different positions in the data. The following equation summarizes MHA:

$$MultiHead(Q,K,V) = Concat(head_1, ..., head_H)W$$

where H is the number of heads, W is a parameter matrix, $W \in R^{hd_k \times d_{model}}$ and d_{model} is the dimensionality of the embedding. The attention heads are used in parallel with dimensionality $d_k = \frac{d_{model}}{H}$ – this optimization reduces the computational cost, making it comparable to that of single-head attention. Overall, the shape of the K, V, Q matrices is then $B \times H \times N \times \frac{d_{model}}{H}$, with batch size B, number of attention heads H, and sequence length N.

The decoder has a similar architecture to the encoder, with the additional operation of multi-head attention over the output of the encoder stack. Its purpose is to generate sequences and it is auto-regressive, meaning that it makes use of its previously generated symbols as additional input. The decoder takes a start token, its previous outputs and the encoder output as input, and generates the sequence's next token as output. The multi-head attention layers work differently to prevent it from conditioning on future tokens. For that, masking is used. Masking can also be utilized in the encoder, e.g., for ensuring the attention mechanism does not attend to padding values. As our proposed approach uses an encoder-only architecture, the decoder structure and workings are less relevant.

It is important to note that the attention mechanism creates an $N \times N$ matrix for each attention head (as seen in Figure 3.2). For every attention head, there is a matrix of size $N \times N$, and each encoder layer has a number of attention heads, resulting in $E \times H \times N \times N$ values stored, with E being the number of encoder layers in the stack. This leads to a quadratic memory and time complexity of the Transformer and can restrict application for very long sequences [45], which has motivated research into optimization techniques that reduce the cost of attention computations. Flash attention is one such method [18]: It splits Q, K and V into smaller blocks, loads them into fast static RAM, and only then computes the attention matrices with respect to these blocks. Each block's output is scaled by an appropriate factor then added up, which leads to the same correct result as the normal attention mechanism ends up with. This approach boosts performance, with the authors reporting flash attention as 3x faster and 20x more memory efficient than exact attention.

Chapter 4

Literature Review

4.1 Transformers in HEP

Transformer models have already been utilized for HEP applications. For example, they have been proposed as suitable architectures for jet tagging: the task of identifying the type of elementary particle which has initiated the “jet” of secondary particles in a collision. The Transformer is given the set of particles and an interaction matrix between them as input, and produces classification scores [44]. Furthermore, Transformers are also considered for track fitting [5]. The track finding stage is executed by first applying the Density-Based Spatial Clustering of Applications with Noise (DBSCAN) algorithm to group hits into clusters of adjacent ones, and then creating the minimum spanning tree of each cluster to order the hits and break the cluster into smaller track segments. Each primary track segment is fed to the Transformer model, and it maps every hit in that segment to the closest 3D point to it from the actual particle trajectory. For that task, the Transformer is shown to outperform a Recurrent Neural Network and a KL-based model.

To our knowledge, there is no published research about the usage of Transformers for the track finding stage of the track reconstruction task.

4.2 Alternative Tracking Solutions

Parallel Kalman Filter. The inherent issue of KL is that it is a sequential algorithm, but substantial research has been conducted into parallelizing KL and using multiple-core architectures to speed up the current algorithm [3, 8]. For example, one proposed solution is to have different threads working together as they advance groups of track candidates layer by layer [34]. As a result, KF does not advance one track candidate at a time through the detector until all possible tracks have been evaluated, but instead there is the global set of track candidates found at one layer which gets propagated forward through the layers until the final one is reached. This approach leads to a speedup of 3x over the default KL algorithm, with the physics performance being comparable to the state-of-the-art.

Hough transform. Another algorithm that can be used for the detection of particle trajectories is the Hough transform (HT). Although this is a mathematical approach, its use case is rather similar to the one we design for the Transformer in this paper. HT can be used to extract trajectories and straight or curved lines within a digital image representation of the hit space, by transforming the particle space into a space of parameters (Hough space) [23]. The algorithm works on the basis that if a series of points lie on a straight line, the corresponding lines in the plane transform will intersect in a single point. Thus, given a few hits originating from the same particle, in Hough space a family of straight lines is generated with an intersection point which can then be used to recreate the line in real space, a.k.a. the track. The algorithm computes a straight line for each hit multiple times, with different angles with respect to the origin, and once the Hough space is populated, selection of candidate tracks happens.

This analytic approach has shown to perform well, but its downside is that storage and computation requirements can easily explode. This can be mitigated by implementing HT on FPGAs – field-programmable gate arrays, i.e., integrated circuits reconfigured for a specific task, acting as hardware accelerators [46]. Naturally, our own proposed approach would also benefit from such an implementation, as FPGAs offer high speed and latency that increases linearly with input data size, although their limited amount of available resources can make porting a Transformer model to FPGA challenging [27].

TrackML Kaggle challenge. In search for machine learning solutions to the tracking problem, CERN hosted the TrackML Particle Tracking Challenge on Kaggle in 2018. A custom dataset was created for that challenge, simulating the High Luminosity LHC conditions[1]. There are almost 6,000 submitted solutions, and from the top three submissions only one relies on an artificial neural network, while the rest make use of mathematical modeling and statistics [7]. The algorithm with highest accuracy takes 8 minutes per event, while the fastest algorithm submitted (also not a ML approach) takes 0.56 seconds CPU time per event [6].

Currently, the state-of-the-art often makes use of the TrackML data for experimentation. However, due to the computationally expensive nature of model training, authors opt to reduce the data in one way or another. For instance, considering only data associated with the inner detector [19, 40], and sometimes even filtering for limited momentum values. There is no consensus for a common data reduction protocol, meaning that the data used by different authors for training and evaluation differs and it is hard to perform a direct comparison between results of different approaches.

GNNs. When it comes to deep learning solutions, one method that has received a lot of attention for this application is using Graph Neural Networks (GNNs). They are usually utilized to predict whether or not edges between the vertices (i.e., hits) present actual physical trajectories. Typically, a graph is generated based on an event with all hits becoming nodes, connected by edges based on some constraint[2]. Then, a GNN is trained to assign weights to edges or prune unlikely ones, such that a fitting connectivity between the vertices is found to represent the trajectories of the particles in that event [19, 28, 35].

One recent approach that has shown success [39, 40] is as follows: initially, the graph is constructed by connecting hits from different detector layers that satisfy certain geometric constraints. Then a fully connected NN is used to estimate the weights of all edges and those with weight lower than a specified threshold are pruned. Next, object condensation occurs: a

[1] More about it in Chapter 5.

[2] For example, geometric constraints or a pre-processing algorithm such as Hough Transform [49].

scheme that enables clustering the hits of the same track in a learned space, and regressing the properties of the reconstructed objects [33]. The clustering is done by Density-Based Spatial Clustering of Applications with Noise (DBSCAN), an iterative clustering algorithm. The performance of this pipeline is assessed on the inner detector hits from the TrackML data. Lieret et al. [40] have defined multiple custom metrics to assess the physics performance, which we also make use of – more information on them and the scores obtained by the GNN pipeline are given in Chapter 6, where we compare our approach to this one. No time performance is reported but the authors identify that the usage of Transformer models would lead to a significant reduction in inference time.

Chapter 5

Methodology

5.1 Systematic Approach Towards Model Design

Usually in the cases of ML or DL models developed for HEP tasks, the model design is based on practices in other fields and the human expert knowledge of this field. We use a systematic approach in the design of a viable solution to the problem to ensure its efficiency and performance [42]. This is enabled through complexity reduction of the problem (i.e., using simplified data) and the iterative increase in its complexity. The simplification allows for efficient evaluation of different ML architectures, and narrows down the initial search space.

5.2 Proposed Transformer Approach

We explore three Transformer architectures utilized in different ways for the tracking task. The first model has the full encoder-decoder architecture, and autoregressively rebuilds tracks one hit at a time, with one or two initial hits provided as seeds. The other two models are used specifically for the task of track finding, and thus associate hits with track or particle labels. One is an encoder-only Transformer used as a classifier: it maps every hit in an event to a class label, where the class labels are determined a priori by binning the track parameter space into quantile-based equally-sized bins. The other is also an encoder-only Transformer, which regresses track parameters per hit in an event. This paper focuses on the last approach, as that is the author's largest contribution to the project. More information on the design of the other two methods can be found in Appendix A.

5.2.1 Transformer Regressor

The proposed Transformer Regressor is a sequence-to-sequence, encoder-only model which takes as input the hits of a single event and produces track parameters per hit. Thus, it maps the coordinates of every hit to a regressed tuple of continuous values that is further specified for the data being used. Since the recorded hits in an event are permutation invariant, the Transformer does not make use of positional encoding. Due to the variable length of the input, when providing the data to the model in batches, we pad all events in the batch up to a maximum number of hits an event can have. Consequently, we also use masking, to enable the attention mechanism to ignore the padding values.

The Transformer has an embedding layer (i.e., a fully connected layer) which increases the dimensionality of the data for increased flexibility: We break up the information for easier processing of the model. Next there is a stack of encoder layers, and lastly, a fully connected layer generates the output by reducing the dimensions. The loss function that the model optimizes is the Mean Squared Error (MSE) loss.

Because of the large amount of memory the attention mechanism demands and the constraints of the available GPU memory, the size of the events that we can work with is limited. We also identify that for larger events, training a single model is extremely costly in terms of time and computational resources. Therefore, for the largest dataset utilized, we take measures to reduce the memory footprint of the attention computation, and speed up training and hyperparameter tuning. We implement and utilize Flash attention instead of exact attention which, as already discussed in Chapter 3, greatly reduces memory usage. We choose this optimization technique in particular because it not only leads to memory that grows linearly with sequence length, but also speeds up training.

Moreover, as Flash attention relies on data being of type float-16, we also make use of mixed precision training[1]. The model's weights, biases and gradients as well as the data passed to it are of type float-16 wherever possible, except during the operations which can greatly benefit from the data being of type float-32 (i.e., values computed by large reductions such as batch normalization) [1]. In addition to this being necessitated by Flash attention, using lower precision in ML computation is another proven way of dealing with the bottleneck of GPU memory [26].

5.2.2 Clustering

The output of the Transformer model is a regressed set of track parameters for each hit in the event, so grouping these regressed values results in a grouping of the hits as well. We make use of clustering to achieve this: a method for producing groups of points such that the data similarity within the group is maximized, and minimized across groups. Therefore, the input to the clustering algorithm is the predicted track parameters and its output is a cluster label for each track parameter, and so a cluster label for each hit in the event.

Since there can be a variable amount of tracks (i.e., clusters) in a single event, we must use a clustering algorithm which does not need the number of clusters pre-specified. DBSCAN is already identified as a suitable technique in related work, and we make use of a modified version of that algorithm: Hierarchical Density-Based Spatial Clustering of Applications with Noise (HDBSCAN). DBSCAN identifies places of high density and expands clusters from them, but it assumes the data contains clusters of similar density [20]. It works with a single parameter related to the global density: ϵ, which denotes the maximum distance between two samples for them to be considered in the same neighborhood. HDBSCAN makes no assumptions about the distribution of the data and clusters, which makes it a more appealing alternative. It essentially performs DBSCAN over varying ϵ values and integrates the results

[1] This is a reliance imposed by the used framework, PyTorch. Mixed precision training is facilitated using the Automatic Mixed Precision library, also part of PyTorch. More about it can be found here: https://pytorch.org/tutorials/recipes/recipes/amp_recipe.html.

to find the best clustering [10]. Consequently, HDBSCAN is more robust and can find clusters at varying densities.

Experimentation with alternative techniques showed that Agglomerative clustering is also a promising approach, leading to similar physics performance. However, the Agglomerative clustering algorithm is known to scale very poorly with number of data points (time complexity of $O(n^3)$) [30]. On the other hand, HDBSCAN has time complexity of $O(n^2)$ [47], which, given our objective of developing a time-efficient alternative to KF, makes HDBSCAN even more appealing.

5.3 Simulations

HEP experiments are not performed on demand, so frequently simulations are utilized for the generation of datasets. These data help evaluate tracking solutions, or study and analyze physics phenomena. For a deep learning based pipeline, simulations are particularly necessary for the creation of train and test datasets. In this work, we make use of two simulators for data of different complexity.

5.3.1 REDVID Data

The first datasets are generated using the REDVID simulator – a highly customizable tool that allows for the creation of events and the derivation of the "recorded" hits in them and their track defining parameters. REDVID can be configured to apply different levels of detail when simulating HEP events [42]. The most basic level operates within a two-dimensional (2D) geometric space, and there are two major levels in three-dimensional (3D) geometric space, with various fine-grained adjustments. This enables us to generate multiple datasets of increasing complexity to test our solutions to the task. They are all summarized in Table 5.1. Each dataset comprises 100k events and has added noise to the hit coordinates.

	3D-lin:3	3D-lin:1-20	3D-lin:10-50	3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100
Tracks per event	3	1-20	10-50	3	1-20	10-50	50-100
Max hits per event	27	180	450	27	180	450	900

Table 5.1: The 7 REDVID-generated datasets used, with increasing complexity. The naming convention used is as follows: xD-y:n where x is the dimensionality, y is the type of track (linear vs helical), and n is the (possibly variable) number of tracks in a single event.

The 3D data we use is based on linear and non-linear track definitions, resulting in recorded hits in the cylindrical coordinate system. A visualization of a couple of events is given in Figure 5.1. The linear tracks are described by two parameters: the angles θ and ϕ in spherical coordinate system, derived from the cylindrical coordinates of the track (r, θ_{cc}, z) in the following manner:

$$\phi = \theta_{cc}$$
$$\theta = \frac{\pi}{2} - arctan(\frac{z}{r})$$

We identify the symmetry property of ϕ: the angle's domain is $[-\pi, \pi]$ and points with $\phi = -\pi$ and $\phi = \pi$ are treated as maximally different by error measures such as Mean Squared Error

(MSE), although they should in fact be treated as if they are the same. In order to bypass this issue of rotational invariance[2], we use the trigonometrical functions $sin(\phi), cos(\phi)$ in the preprocessing step[3]. Consequently, the Transformer model maps a sequence of hit coordinates (r, θ_{cc}, z) to a sequence of the track parameters $\theta, sin(\phi), cos(\phi)$. This usage of trigonometrical functions also has the added advantage of bounding the values to the range of $[-1, 1]$ which might be advantageous for the model's learning.

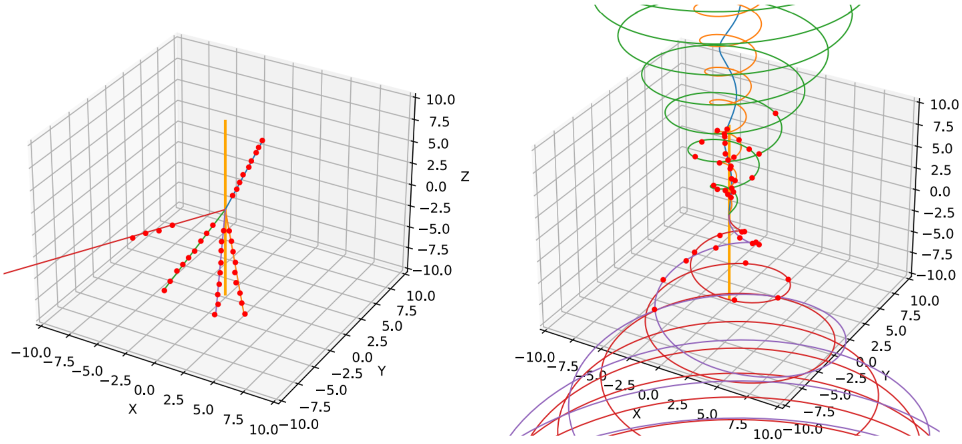


Figure 5.1: Visualized events with 5 tracks each, with the red dots representing the recorded hits. Left: linear tracks, right: helical tracks.

The non-linear 3D tracks have a helical form and are given in the cylindrical coordinate system:

$$r = r_0 + a \cdot t$$

$$\theta = \theta_0 + d \cdot t$$

$$z = z_0 + b \cdot t$$

where a, d, b are the radial, azimuthal, and pitch coefficient respectively and t is the free variable. These coefficients determine the change in pitch, azimuth, and radius of the track in an attempt to emulate the magnetic field's effects. This is simulated by the helices being extended or squished (visualized in Figure 5.1). It is precisely these three coefficients that uniquely define each particle's trajectory and that we consider for track parameters. No preprocessing steps are needed for this data, meaning that the Transformer maps a sequence of hit coordinates (r, θ_{cc}, z) to a sequence of the track parameters a, d, b.

5.3.2 TrackML Data

The TrackML data [32] created for the Kaggle challenge of the same name is the other dataset we make use of. As already mentioned in Chapter 4, this huge dataset simulates the conditions in the High-Luminosity stage of the LHC.

[2] Although θ is also an angle, this issue is not present for that track parameter, as its domain is $[0, \pi]$.

[3] For more information on the need to use $sin(\phi)$ and $cos(\phi)$, please refer to Appendix B.1.

The dataset consists of 3D hit coordinates in Cartesian coordinate system with the global z axis defined along the beam direction [7]. There are 10000 tracks per event on average, and between 70000 and 135000 hits per event. Besides the larger number of tracks, this dataset is more complex than the REDVID-generated data in multiple ways: it involves pile-up, meaning that not all particle tracks originate from the same point – there are 200 different origin points for the tracks. There are also noisy hits belonging to no track. Lastly, the tracks can be helical or linear, and of varying lengths.

This dataset contains weights associated with each hit and the sum of all hits' weights in an event sums up to 1. The hit weights are used in the calculation of the TrackML score, which assesses track reconstruction – more about it in the next chapter.

Following our approach of iteratively increasing the complexity of the problem, we generate multiple subsets of the TrackML data, summarized in Table 5.2. The smaller datasets are created by randomly sampling particles and the hits they generated from every event. The dataset by design contains only 8745 events, so in order to increase the amount of training data, we sample 5 times from every event, resulting in 43725 events in total. Visualization of an event from each dataset is shown in Figure 5.2.

	TML:10-50	TML:50-100	TML:200-500
Tracks per event	10-50	50-100	200-500
Max hits per event	700	1200	5000

Table 5.2: The three TrackML-derived datasets with increasing complexity, used for training and evaluating models.

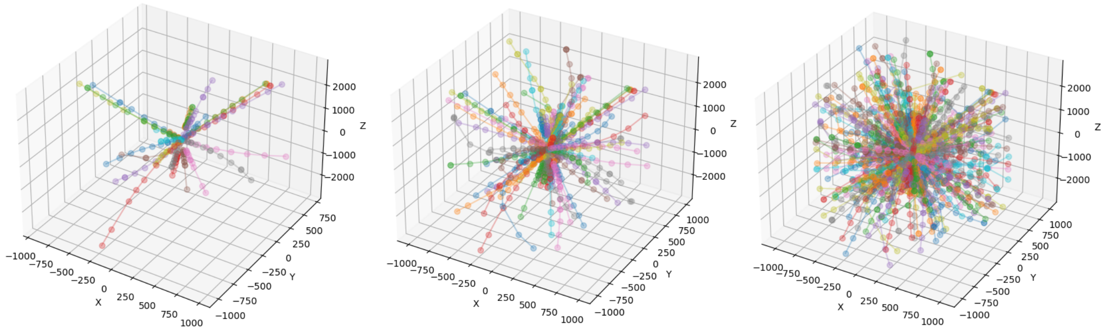


Figure 5.2: Events from TML:10-50 (left), TML:50-100 (center), and TML:200-500 (right).

In addition to these datasets, used for training and evaluation, we create a few more TrackML subsets with the purpose to assess the generalizability and scalability of the models. As we do not train models on these data, we do not need a large number of events so each of these datasets contains only 2245 events. The datasets used for generalizability testing are described in Table 5.3. Besides those, we create eight more datasets, which have a fixed amount of tracks per event and more consistent steps of increase between the input size, in order to calculate a truthful estimation of the pipeline's time complexity. The datasets used for scalability are TML:1000, TML:2000, TML:3000, TML:4000, TML:5000, TML:6000, TML:7000, and TML:8000.

	TML:500-1000	TML:2000-5000	TML:all
Tracks per event	500-1000	2000-5000	6200-13500
Max hits per event	9500	55000	135000

Table 5.3: The three additional, larger TrackML-derived datasets used for assessing generalizability. TML:all is a dataset in which the events are in their original size.

The available track-related parameters in the dataset ground truth information are 10, but we only make use of four of them as necessary to uniquely identify tracks[4]. We use the initial momentum (in GeV/c) along each global axis (p_x, p_y, p_z), but it is transformed into the spherical coordinate system using the following equations:

$$\theta = arccos(\frac{p_z}{\sqrt{p_x^2 + p_y^2 + p_z^2}})$$

$$\phi = arctan2(\frac{p_y}{p_x})$$

Similarly to the 3D linear REDVID data, to avoid the angle symmetry issue, we map ϕ to $sin(\phi)$ and $cos(\phi)$. The last track parameter is the particle charge, as multiple of the absolute electron charge, $q \in \{-1, 1\}$. And so the Transformer must regress $\theta, sin(\phi), cos(\phi), q$ meaning that it has 3D input and 4D output. The only pre-processing step taken is normalization of the input data, due to the varying scales of the x, y, z coordinates.

[4] This is a decision based on some experimentation with different combinations and numbers of parameters, but not necessarily the most optimal choice of track parameters.

Chapter 6

Experiments and Results

6.1 Experimental Setup and Design

A different model is trained for the 10 datasets utilized. Each of the models is trained for up to 500 epochs, with an early stopping condition. We make use of the Adam optimizer, and a learning rate of 0.001. Of each dataset, 70% is used for training, 15% as a validation set for the hyperparameter optimization, and 15% as a test set for evaluating the performance of the fully-trained model. The train and validation loss of a few models are given in Figure 6.1 (and the rest are included in Appendix B.3). We can see that the models are learning, with the loss dropping and improving significantly. It is curious that the validation loss is consistently lower than the train loss. This could be due to fact that dropout is used during training and not during validation. We believe it is not a sign of overfitting, given that the test loss of the fully trained model (reported in Table 6.1) is always lower than or equal to the train and validation loss.

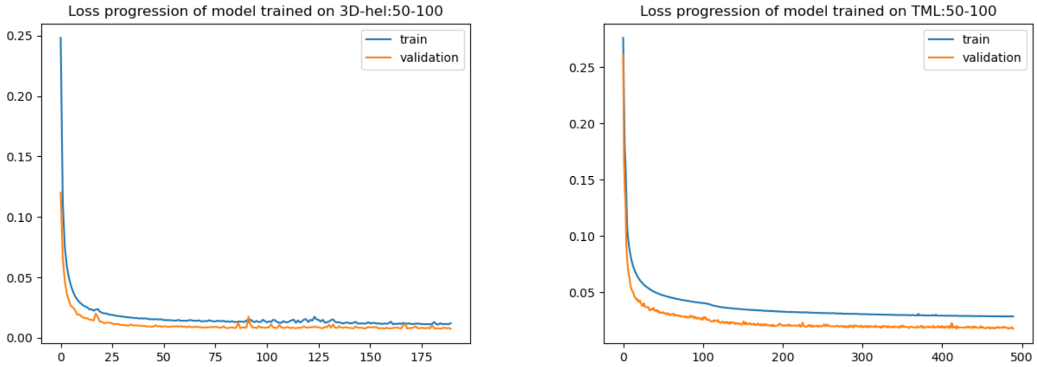


Figure 6.1: The train and validation loss evolution of the models trained on 3D-hel:50-100 (left) and TML:50-100 (right) over the epochs.

3D-lin:3	3D-lin:1-20	3D-lin:10-50	3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100	TML:10-50	TML:50-100	TML:200-500
0.00006 (±0.00005)	0.00006 (±0.00002)	0.00007 (±0.00002)	0.0007 (±0.005)	0.003 (±0.005)	0.005 (±0.004)	0.007 (±0.003)	0.006 (±0.008)	0.017 (±0.014)	0.05 (±0.01)

Table 6.1: The average MSE and its standard deviation for each trained Transformer Regressor model, calculated for the test data of each utilized dataset.

A summary of the models' hyperparameters is provided in Appendix B.2. A brief optimization with a small set of hyperparameter values is conducted with the help of Grid Search, the implementation of which was facilitated using the Weights & Biases website for hyperparameter tuning[1]. Our code can be found at this project's repository[2]. We use the PyTorch deep learning framework for the implementation of the Transformer.

It is important to note that Flash attention is a relatively new concept, which means that it is not yet fully integrated into largely utilized libraries such as PyTorch. In PyTorch, Flash attention is currently in beta mode and some of its functionalities are not yet implemented, e.g., usage of attention masks or variable length inputs while Flash attention is enabled. Because of this, the batch size communicated to the data loaders is 1 (to remove the need for padding and thus masking) when training with data large enough to necessitate the usage of Flash attention. Despite this limitation, we implement manual batching to avoid stochastic learning. The parameters are updated in a backward pass only every B many forward passes, with B being the batch size. For the reported models, only one trained on TML:200-500 makes use of Flash attention.

For the clustering algorithm, we utilize sklearn's implementation of HDBSCAN. The metric for calculating the distance between points is simply euclidean distance. The algorithm has two main parameters to tune: the minimum number of samples in a group for it to be considered a cluster (i.e., smaller groups are seen as noise), and the number of samples in the neighborhood of a point for it to become a core point. We treat these two parameters as hyperparameters to be tuned per dataset; the optimal values are presented in Appendix B.2.

All of our experiments are conducted on the Dutch supercomputer, managed by SURF: Snellius. It offers fast processors and GPUs, a lot of memory space and the ability to process large datasets[3]. More specifically, the training and evaluation of each model was done on an NVIDIA A100 Tensor Core GPU, with 40GB memory. We also make use of Intel(R) Xeon(R) Platinum 8360Y CPU @ 2.40GHz.

6.2 Performance Metrics

To evaluate the performance of our proposed method, we make use of a few custom metrics. Firstly, the TrackML score, designed for the TrackML challenge [32]. It is consistent with the various metrics used in physics reconstruction, and essentially combines the Jaccard version of counting pairs and set matching [7]. In the definition of this metric, reconstructed tracks

[1] The WandB website is a free to use AI developer platform which offers tools for ease of training and fine-tuning models. More information can be found on the website itself: `https://wandb.ai/site`.

[2] The repository is publicly accessible: `https://github.com/nadiand/hep_transformer`

[3] More about Snellius can be found on its official website: `https://www.surf.nl/en/services/snellius-the-national-supercomputer`.

with four or more hits are considered, and at least 50% of a reconstructed track's hits must originate from the same truth particle for that track to contribute to the scoring. The score of a track is the sum of the correctly assigned hits' weights. No penalty for incorrectly assigned hits is necessary as a wrongly associated hit automatically reduces the score for the track it should have been associated to.

Since the REDVID simulations do not generate particles, but tracks, as part of the true data, we have replaced the true particle with the true track for the evaluation of the models trained on that data. Furthermore, we consider the weight value 1 for all hits.

The other three physics performance metrics we utilize are taken from Lieret et al. [40]:

- Perfect match efficiency ϵ^{perf}: The fraction of reconstructed tracks that include all hits of a single particle from the ground truth and no other hits, normalized to the number of particles.
- LHC-style match efficiency ϵ^{LHC}: The fraction of reconstructed tracks in which at least 75% of the hits belong to the same particle, normalized to the number of reconstructed tracks.
- Double majority match efficiency ϵ^{DM}: The fraction of reconstructed tracks in which at least 50% of the hits belong to the same particle and it has less than 50% of its hits outside of the reconstructed track, normalized to the number of particles.

Note that the four aforementioned metrics are calculated for a given hit-track association. Therefore, they do not evaluate only the Transformer Regressor, but the overall track finding pipeline (Transformer and HDBSCAN as a whole). Consequently, a high score indicates both perform well, while a low score means at least one does not perform well. We go into more detail about this and attempt to directly measure the performance of the separate parts of the pipeline later in this chapter.

6.3 Results

6.3.1 Physics Performance

The TrackML and efficiency scores of the Transformer Regressor for the seven REDVID-generated datasets and the three TrackML-derived datasets are presented in Table 6.2. The TrackML scores of the other two Transformer architectures investigated in our team, on some of the aforementioned datasets, are included in Table 6.3. Note that the Transformer Classifier has not been trained on TML:200-500 as of the time of writing, while the Autoregressive Transformer is too time-consuming to be trained and optimized on that dataset. Furthermore, the Autoregressive Transformer builds a track starting with two seed hits that are already known to originate from the same particle, which gives its scores a small boost.

As we can see, with the increase of data complexity, and most importantly number of tracks and hits per event, the TrackML score gradually drops for all three models. The encoder-only approaches both have relatively high scores despite the reduction due to data complexity. The Autoregressive Transformer appears to be least suited for the task of track finding as a stand-alone model, given its significant drop in performance for the TrackML-derived data and the poor scalability in terms of training speed.

	3D-lin:3	3D-lin:1-20	3D-lin:10-50	3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100	TML:10-50	TML:50-100	TML:200-500
TrackML score	0.996	0.984	0.97	0.994	0.946	0.92	0.85	0.932	0.862	0.7 (0.67)
ϵ^{perf}	0.98	0.98	0.94	0.98	0.87	0.78	0.6	0.78	0.65	0.4 (0.36)
ϵ^{DM}	0.99	0.987	0.97	0.99	0.96	0.94	0.89	0.91	0.87	0.75 (0.72)
ϵ^{LHC}	0.996	0.99	0.98	0.996	0.97	0.96	0.92	0.97	0.93	0.82 (0.79)

Table 6.2: Summary of the ϵ^{perf}, ϵ^{DM}, ϵ^{LHC} and TrackML scores obtained by the Transformer Regressor models for the 10 used datasets. For the models trained on TML:200-500, the result with Flash attention is in parentheses.

	3D-lin:10-50	3D-hel:10-50	3D-hel:50-100	TML:10-50	TML:200-500
Autoregressive Transformer	0.93	0.85	0.85	0.26	n/a
Transformer Classifier	0.98	0.96	0.93	0.93	n/a
Transformer Regressor	0.97	0.92	0.85	0.93	0.7 (0.67)

Table 6.3: Summary of the TrackML scores obtained by the Autoregressive Transformer and the Transformer Classifier models for five of the utilized datasets. The scores of the Transformer Regressor are included for ease of comparison.

The results indicate that even in the most complex data, the pipeline involving Transformer Regressor still reconstructs a number of perfect tracks. Given the high ϵ^{LHC} and ϵ^{DM} values, we see that the majority of hits in a reconstructed track consistently originates from one particle, and most particles have an associated reconstructed track. Similarly to the TrackML score, the efficiency is reduced the more complex the data is but even for TML:200-500, ϵ^{LHC} and ϵ^{DM} are quite high.

Although we make use of the metrics defined by Lieret et al. [40], a direct comparison to their results is not possible, because they not only use a different subset of the data, but also limit their score calculation to a particular set of particles – those with larger momentum[4], i.e., less curvature. However, their scores can still provide insight into where our proposed approach fits in a broader context. Lieret et al. obtain $\epsilon^{DM}_{p_T>0.9} = 96.4\%, \epsilon^{LHC}_{p_T>0.9} = 98\%, \epsilon^{perf}_{p_T>0.9} = 85\%$.

It is worth mentioning that when exact attention is used, the scores are slightly higher: e.g., the model trained on TML:200-500 with exact attention achieves TrackML score higher by 0.03 in comparison to the model using Flash attention. This is most likely because of the loss of precision when using float-16, imposed by Flash attention, vs. float-32. However, due to the great speed-up that Flash attention provides and the necessity to use it for data larger than the one reported on, we focus on the results obtained with Flash attention.

6.3.2 Generalizability

While a model is trained on and optimized for a particular dataset, thanks to the Transformer's ability to accept variable length input, we can assess the generalizability of our models to other datasets. For that, we evaluate each model on the other datasets with the same characteristics, i.e., a Transformer Regressor trained on 3D helical data gets scored for

[4] This filter is imposed by only considering the particles with transverse momentum $p_T > 0.9$ GeV/c, where p_T is a projection of the momentum vector.

all four REDVID-generated helical datasets. The results are given in Tables 6.4, 6.5, and 6.6. For the linear data all models have comparable performance on all three datasets. This trend is also observed for helical data, for all models except the one trained on 3D-hel:3. For the TrackML data, the drop in performance when evaluating on larger data than the one used for training is more exaggerated, likely because of the data complexity (e.g., pile-up). We also test the model trained on TML:200-500 on the larger three TrackML-derived datasets. The efficiency scores on these larger datasets are included in Appendix B.4.

		Tested on:		
		3D-lin:3	3D-lin:1-20	3D-lin:10-50
Trained on:	3D-lin:3	0.996	0.978	0.947
	3D-lin:1-20	0.997	0.984	0.963
	3D-lin:10-50	0.99	0.98	0.97

Table 6.4: The different models trained on linear REDVID datasets evaluated on all three linear datasets.

		Tested on:			
		3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100
Trained on:	3D-hel:3	0.994	0.67	0.2	0.04
	3D-hel:1-20	0.99	0.946	0.86	0.7
	3D-hel:10-50	0.95	0.96	0.92	0.83
	3D-hel:50-100	0.97	0.96	0.92	0.85

Table 6.5: The different models trained on helical REDVID datasets evaluated on all four helical datasets.

		Tested on:					
		TML:10-50	TML:50-100	TML:200-500	TML:500-1000	TML:2000-5000	TML:all
Trained on:	TML:10-50	0.932	0.79	0.46	–	–	–
	TML:50-100	0.86	0.862	0.53	–	–	–
	TML:200-500	0.8 (0.76)	0.85 (0.81)	0.7 (0.67)	(0.45)	(0.2)	(0.06)

Table 6.6: The different models trained on TrackML-dervied datasets evaluated on all defined TrackML datasets: the three used for training and the three used for generalizability testing. For the models trained on TML:200-500, the result with Flash attention is in parentheses. The larger datasets can only fit in memory when Flash attention is enabled.

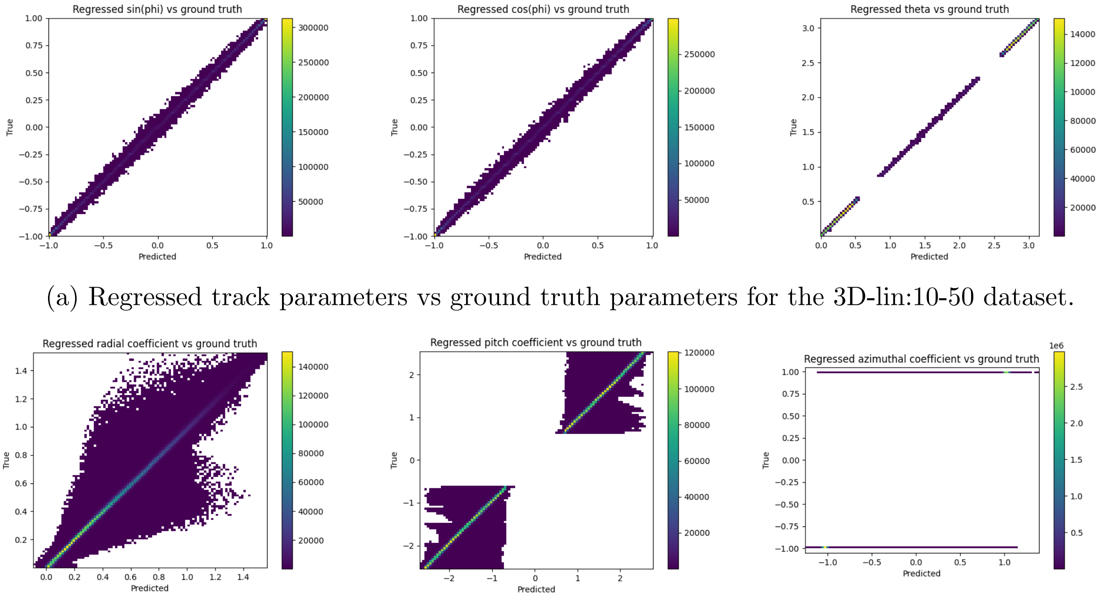
For TML:500-1000 the scores are still rather high for a dataset up to 5x larger than what the model was trained for. For the larger two datasets the results become very low as is expected: the scores clearly show the need for training a more complex Transformer for larger events.

6.3.3 Transformer vs Clustering Performance

To gain insight into the Transformer model's performance as a regressor of track parameters, we analyze the trained models separately from HDBSCAN. We calculate the standard deviation of the predicted parameters from the true ones: the measurements for the models trained on the TrackML subsets are presented in Table 6.7. The errors of the Transformer are also visualized using heatmaps in Figure 6.2 for three of the models, to illustrate the performance on datasets with differing complexities. The rest of the models' heatmaps are included in Appendix B.5 and the standard deviation of the linear and helical REDVID datasets are reported in Appendix B.4.

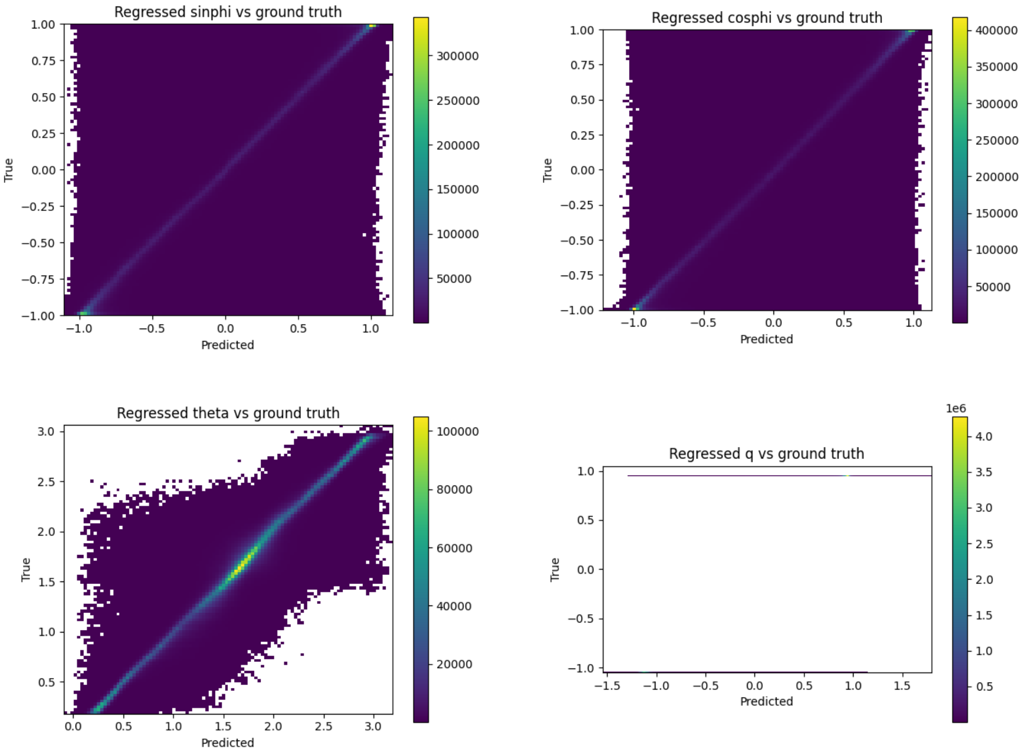
	TML:10-50	TML:50-100	TML:200-500
$\sigma(\theta)$	0.06	0.06	0.08 (0.11)
$\sigma(sin(\phi))$	0.06	0.07	0.15 (0.17)
$\sigma(cos(\phi)))$	0.06	0.08	0.16 (0.19)
$\sigma(q)$	0.13	0.24	0.32 (0.52)

Table 6.7: Standard deviation of the track parameters regressed by the models trained on the TrackML-derived datasets. For the models trained on TML:200-500, the result with Flash attention is in parentheses.



(a) Regressed track parameters vs ground truth parameters for the 3D-lin:10-50 dataset.

(b) Regressed track parameters vs ground truth parameters for the 3D-hel:50-100 dataset.



(c) Regressed track parameters vs ground truth parameters for the TML:200-500 dataset. The model uses Flash attention.

Figure 6.2: Visualization of the regressed track parameters by the Transformer plotted against the actual track parameter values from the ground truth. White background signifies there are no points in that region. The ideal case is a thin diagonal line.

The heatmaps clearly indicate that the the model is learning the mapping of hits to track parameters, with the complex TML:200-500 being most difficult to learn. Despite that, even for that dataset, we can see that the majority of the parameters fall on the diagonal. The standard deviation is overall low for all trained models, the highest one being for the TML:200-500 model again. These findings are in agreement with the reported scores above. Moreover, it is interesting to note that the distribution of the errors the models make for each track parameter has the shape of a Laplace distribution[5], with a sharp peak at its mean and a slower, prolonged decay. This, together with what we see from the heatmaps, shows that the models' standard deviation is due to the effect of outliers, i.e., it sometimes makes very large mistakes but most are quite small.

Next, we investigate the behaviour of the clustering algorithm on its own by assessing its performance on the pure or noisy ground truth of dataset TML:10-50. Table 6.8 provides the results of the evaluation of the clustering when the actual track parameters as derived from the ground truth are used as input with noise of varying magnitude added to it. This is simulated by sampling random noise from a Laplace distribution with standard deviation $\sigma \in \{0, 0.01, 0.05, 0.1, 0.15\}$ and adding it to all of the track parameters. For reference, 1% noise for θ would mean about $\sigma(\theta) = 0.03$, and for $sin(\phi)$ it is about $\sigma(sin(\phi)) = 0.02$.

[5] Histogram of the errors of each track parameter for the model trained on TML:10-50 are included in Appendix B.5.

σ	0	0.01	0.05	0.1	0.15
TrackML score	1	0.97	0.77	0.55	0.39
ϵ^{perf}	0.96	0.92	0.67	0.42	0.25
ϵ^{DM}	0.97	0.95	0.78	0.56	0.4
ϵ^{LHC}	0.99	0.98	0.85	0.71	0.6

Table 6.8: The TrackML score and the efficiency scores obtained when clustering the ground truth of TML:10-50 test set with varying magnitude of noise added to it. The same amount of noise is added to every track parameter.

As we can see, even small values of standard deviation ($\sigma < 0.1$) from the real parameter values can lead to a large reduction in the scores. This sensitivity of the algorithm essentially acts as a bottleneck to the scores and performance we can realistically achieve with the overall pipeline.

6.3.4 Time Complexity

To assess the time performance of our approach, we measure the GPU time, and CPU time during inference. We profile the Transformer prediction step and the clustering step separately and present the measurements in Table 6.9. For the models trained on TML:200-500, we report the speed with exact attention first and that with Flash attention in parentheses. It is important to highlight the contribution of Flash attention. The Transformer with Flash attention takes 3.6ms on average for a single event from TML:200-500, while the model using exact attention on the same dataset takes 52.8ms – a speedup of 15x. For a larger data size it is impossible to see the speedup given by Flash attention, as the attention matrices cannot fit in memory when exact attention is used, and thus the model cannot be evaluated.

	3D-lin:3	3D-lin:1-20	3D-lin:10-50	3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100	TML:10-50	TML:50-100	TML:200-500
CUDA time	2.2	2.2	2.4	2.2	2.2	2.3	4.2	2.3	3.2	52.8 (3.6)
CPU time	1.5	3.6	8.3	1.5	3.8	8.7	18.6	5.6	16.3	68.8 (72.2)

Table 6.9: Average inference speed per event measured in CPU and GPU time. The Transformer's execution is captured in the GPU time recording, while the clustering algorithm runs on the CPU. For the models trained on TML:200-500, the result with Flash attention is in parentheses.

The size of these reduced-complexity datasets is rather small and we are likely measuring the overhead only and cannot truthfully gauge the time complexity of the pipeline. Furthermore, in realistic scenarios, the data size would be much larger, necessitating the need for Flash attention. Thus, we record the speed of the model trained on TML:200-500 with Flash attention, on the eight extra datasets made specifically for scalability tests. The results are presented in Figure 6.3. In addition, we record that the inference time on a full TrackML event is 0.5s GPU time and 4.8s CPU time.

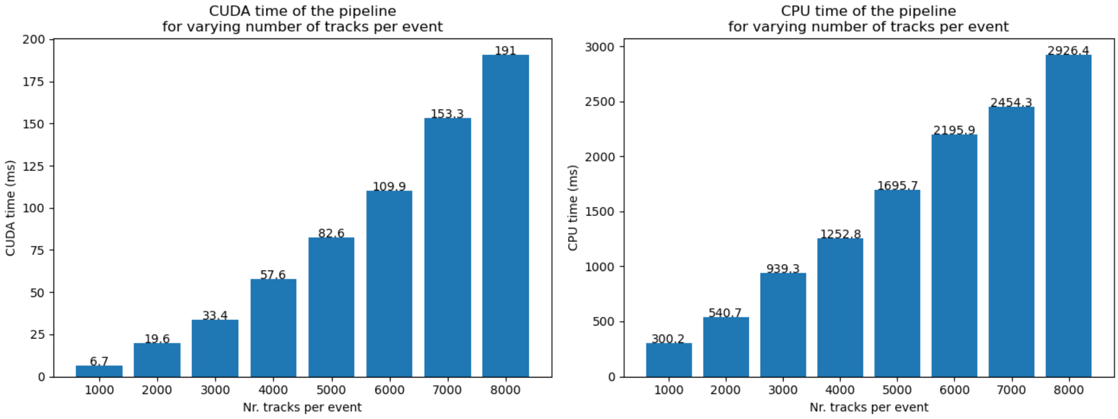


Figure 6.3: Average inference speed in milliseconds per event measured in CPU and GPU time, of the model trained on TML:200-500 using Flash attention on the eight datasets created for scalability. The Transformer's execution is captured in the GPU time recording, while the clustering algorithm runs on the CPU.

The time complexity of both the Transformer model and the clustering algorithm appear to be sub-quadratic, and even roughly linear.

Lastly, we report on the time needed for training one of our models. Flash attention speeds up the training procedure by 5.7x, with the model on TML:200-500 with exact attention taking 40 minutes wall-clock time for a single training epoch with batch size of 16, while the model trained with Flash attention needs only 7 minutes.

Note that the task of moving data to GPU is not included in the measurements, as we assume the data is already situated there when the algorithm is ran. However, we still report the CPU time it takes to load the data to GPU: between 0.025ms (for TML:10-50) and 4.3ms (for TML:all) per event, depending on the data size.

Chapter 7

Discussion and Future Work

Physics performance. The reported results indicate the usage of Transformers for the task of track finding is promising. Most consistently good results are recorded for the Transformer Classifier, while the Autoregressive Transformer seems to be least suitable for the task. However, it is possible that its performance can be boosted by further optimization and extending the pipeline in different ways, listed later in this section.

The Transformer Regressor in particular achieves rather high scores even on the largest dataset used: $\epsilon^{DM} = 0.72, \epsilon^{LHC} = 0.79$ and TrackML score of 0.67, despite being not fully optimized. According to the generalizability test, feeding the model with a slightly larger event can reduce the performance slightly but not fully tank it, which is a great feature, given that the recorded number of hits vary between event. The lower ϵ^{perf} score is not necessarily of large consequence, as a cluster containing the majority of hits from one particle might still provide a good estimation of the particle's track parameters in the track fitting stage. Moreover, adding another network to the pipeline focused on refining the clusters by removing wrongly associated hits, can improve the purity of the grouped hits and facilitate the job of the subsequent track fitter. This type of refinement at a later stage can be beneficial for all three Transformer approaches. For more information on this potential refining network, refer to Appendix C.3.

A further improvement of the track finding effectiveness is possible, for instance by decomposing the problem domain based on the location of the hits within the coordinate system or based on the track parameter space, subdividing the task. We can then train a specialized Transformer for each subproblem, which can lead to a performance boost due to the decreased sequence length and complexity within the data, and increased resolution within that subdomain. More information on this can be found in Appendix C.1.

The size of our models is rather small, with the majority having less than 100,000 parameters and the largest one having less than 1 million. The GNN proposed by Lieret et al. for instance has almost 2 million parameters [40]. Since in our experiments with TML:200-500, increasing the model size so far always lead to a larger score, we believe that given more resources, increasing the model complexity would lead to a significant boost in performance and is thus a direction worth exploring. On the other hand, the compactness of the models can be further made use of for increasing the speed of the models. Previous work [27, 38] has already shown that small Transformer architectures can be accelerated with the usage of FPGAs, and that models with millions of parameters can be compressed and implemented to FPGA without significant accuracy drop.

Time complexity. The Transformer scales sub-quadratically with the size of the input, largely thanks to Flash attention which greatly speeds up computation. Although the clustering algorithm also exhibits the same time complexity, it slows down the pipeline and is currently CPU-bound. In future work, using a GPU implementation of HDBSCAN is worth investigating.

It is unfortunately impossible to directly compare the inference speed of our model to the approaches utilizing Kalman Filters currently in use at LHC, as that would require adapting the algorithm to the specific datasets, but as already established, current in-house algorithms take around 12s CPU time per event on data comparable to TrackML. Our proposed pipeline requires 0.5s GPU time and 4.8s CPU time per event on the full TrackML data – however, the speed is not directly comparable as we utilize a different CPU. The speedup offered by our pipeline is similar to the speedup of parallelizing KF. When taking into account usage of the Transformer only (e.g., as is the case for the Transformer Classifier), the optimized fast KF mentioned in Chapter 2 and the fastest identified solution by the TrackML challenge would be outperformed.

Our proposed solution can be further sped up: domain decomposition coupled with parallel execution of multiple Transformers will bring down the inference time to that of a smaller event. For example, if a full event is broken down into a few subevents, comparable to the TML:2000 data, the pipeline would need 0.02s GPU time and 0.5s CPU time – significantly less than any other state-of-the-art solution. The execution time will be even further reduced for the Transformer Classifier which does not need the clustering step. This makes the proposed architecture a good alternative to the Kalman Filter in terms of speed.

Thanks to Flash attention, the Transformer models can be trained rather quickly. This is also very beneficial in addition to fast inference, as it enables for fast retraining in case, for example, the hardware at the collider is modified and the recorded data type changes, necessitating the Transformer be trained anew. The generalizability test results indicate that fine-tuning the model could also be enough, if the change in size between train and test data is not too large, and the rest of the conditions (i.e., track parameters, pile-up) remain the same.

Transformer design limitations. Perhaps the biggest challenge for the Transformer Regressor model is the discovery of track parameters that sufficiently define a track and can be learned by the model. What coordinate system they should be in, dealing with angle symmetry and varying scales, different weighting of the tracks' contribution to the loss, etc. are some examples of things to consider. Furthermore, there is available physics knowledge about the problem that the model is currently not exploiting to the fullest. The helical and linear tracks, as already shown in Chapter 5, can be defined via formulas which could, in future work, be introduced to the model in some way. For instance, supplying the attention mechanism with an extra matrix which enables the Transformer to derive more physics-informed connections within the data. The loss function could also be modified to better fit the problem. We have already experimented with creating a custom loss that utilizes angle difference (see Appendix C.2), but that idea would need further work.

Another direction for future research, related to both track parameter choice and physics information, is tackling the issue of pile-up. Currently we do not take any measures to aide the Transformer Regressor in differentiating tracks originating at different points, but that can be remedied by for instance including the initial vertex position of the particle in the track parameters. We believe this may improve the physics performance for larger data.

Given the possibly very different nature of the track parameters, e.g., θ and ϕ being real values and q being binary, it is possible that the chosen loss (MSE) does not fit the task too well and leads to the Transformer under performing. One direction for future work would be to train multiple networks, one for each track parameter – consequently, the model predicting q can be a binary classifier. This would further enable the models to estimate their uncertainty and output it along the track parameter, which can be utilized in the clustering stage, by ignoring the hits with predicted track parameters that the Transformer is very uncertain about.

Clustering. The usage of clustering is suboptimal as it disables us from training the pipeline end-to-end. One unfavourable side effect of this is the need to hypertune the clustering parameters at inference time once the model is deployed, as illustrated by the fact that the best results for different datasets are not achieved with the same parameter values.

Furthermore, clustering is very sensitive to the performance of the Transformer. There must be almost perfect regression with no overlap between different particles' track parameters (i.e., no noise) for the clustering to perform very well. As our experiments show, even a small amount of standard deviation from the ground truth can lead to a large reduction in scores, and our largest model has standard deviation between $0.1 - 0.5$ for each track parameter. Consequently, the imperfections in the Transformer's performance become exaggerated by the clustering algorithm. It is possible that training a network to output the uncertainty of its predictions can be beneficial to improve the clustering performance. Alternatively, we could also utilize deep net clustering or learnable clustering where the grouping happens in a latent space of a higher dimension instead of in the track parameter space.

One possible way for future work to alleviate the exaggeration introduced by the clustering algorithm would be to make use of a second network which refines the clusters one by one and improves their purity by filtering out the false hits, as mentioned above. Alternatively, an autoregressive model can attempt to add to hit groups more hits which were supposed to be part of the group but were missed by the first network. One could also attempt to make the clustering algorithm more suitable for the tracking task by incorporating physics information. For instance, the metric used for calculating the distance between hits can be modified to include angle difference. This is all discussed in more depth in Appendix C.

Track fitting. As already made explicit, our proposed approach does not cover the full task of track reconstruction. We tackle the challenge of track finding, while the less complex track fitting step that comes after is out of scope for this paper. The Transformer Regressor already derives the track parameters, but the regressed values are per hit so they cannot be directly used. Furthermore, they are estimates instead of actual track defining parameters. In future work, the pipeline can be extended to cover the track fitting stage, by adding another architecture that takes the groups of hits and produces a single set of track parameters that define the trajectory the hits lie on. One possible way this can be achieved is using another Transformer Regressor, similarly to [5].

Chapter 8

Conclusion

This paper presents a Transformer-based approach to the tracking challenge in HEP. We strove to conduct a proof of concept, showing the viability of using a Transformer architecture for regression in the task of track finding, with simplified data of increasing complexity. Despite the reliance on clustering, our results show that the proposed model and overall pipeline have high physics performance. Furthermore, with its sub-quadratic scaling, our approach has the possibility to achieve a significant speedup over the traditional Kalman Filters and GNN-based solutions, especially if certain improvements to the pipeline are applied, such as domain decomposition. This is the case even more for the model utilized as a classifier. In conclusion, the Transformer appears to be suitable for the association of hits to tracks, and further research into its usage, especially on more realistic data, is vital.

Bibliography

[1] Train With Mixed Precision, NVIDIA Docs. `https://docs.nvidia.com/deeplearning/performance/mixed-precision-training/index.html#training_pytorch`. Accessed: 2024-03-31.

[2] Ai, X., Gray, H. M., Salzburger, A., and Styles, N. A non-linear kalman filter for track parameters estimation in high energy physics. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 1049* (2023), 168041.

[3] Ai, X., Mania, G., Gray, H. M., Kuhn, M., and Styles, N. A gpu-based kalman filter for track fitting. *Computing and Software for Big Science 5*, 1 (2021), 20.

[4] Airapetian, A., Dodonov, V., Micu, L., Axen, D., Vinogradov, V., Akerman, D., Szeless, B., Chochula, P., Geich-Gimbel, C., Schacht, P., et al. *ATLAS detector and physics performance: Technical Design Report, 1.* No. CERN-LHCC-99-014. ATLAS-TDR-014, 1999.

[5] Alonso-Monsalve, S., Sgalaberna, D., Zhao, X., McGrew, C., and Rubbia, A. Artificial intelligence for improved fitting of trajectories of elementary particles in dense materials immersed in a magnetic field. *Communications Physics 6*, 1 (2023), 119.

[6] Amrouche, S., Basara, L., Calafiura, P., Emeliyanov, D., Estrade, V., Farrell, S., Germain, C., Gligorov, V. V., Golling, T., Gorbunov, S., et al. The tracking machine learning challenge: throughput phase. *Computing and Software for Big Science 7*, 1 (2023), 1.

[7] Amrouche, S., Basara, L., Calafiura, P., Estrade, V., Farrell, S., Ferreira, D. R., Finnie, L., Finnie, N., Germain, C., Gligorov, V. V., et al. *The tracking machine learning challenge: accuracy phase.* Springer, 2020.

[8] Berkman, S., Krutelyov, V., Hall, A. R., McDermott, K., Lantz, S. R., Vourliotis, E., Kortelainen, M., Gravelle, B., Yagil, A., Elmer, P., et al. arxiv: Speeding up the cms track reconstruction with a parallelized and vectorized kalman-filter-based algorithm during the lhc run 3. Tech. rep., 2023.

[9] Braun, N. *Combinatorial Kalman filter and high level trigger reconstruction for the Belle II experiment.* Springer, 2019.

[10] Campello, R. J., Moulavi, D., and Sander, J. Density-based clustering based on hierarchical density estimates. In *Pacific-Asia conference on knowledge discovery and data mining* (2013), Springer, pp. 160–172.

[11] Cerati, G., Elmer, P., Lantz, S., McDermott, K., Riley, D., Tadel, M., Wittich, P., Würthwein, F., and Yagil, A. Kalman filter tracking on parallel architectures. In *Journal of Physics: Conference Series* (2015), vol. 664, IOP Publishing, p. 072008.

[12] Chernyavskiy, A., Ilvovsky, D., and Nakov, P. Transformers:"the end of history" for natural language processing? In *Machine Learning and Knowledge Discovery in Databases. Research Track: European Conference, ECML PKDD 2021, Bilbao, Spain, September 13–17, 2021, Proceedings, Part III 21* (2021), Springer, pp. 677–693.

[13] Collaboration, A., et al. Fast track reconstruction for hl-lhc. Tech. rep., Tech. Rep. ATL-PHYS-PUB-2019-041, CERN, Geneva, 2019.

[14] Collaboration, T. A. The ALICE experiment at the CERN LHC. *Journal of Instrumentation* (2008).

[15] Collaboration, T. A. The ATLAS Experiment at the CERN Large Hadron Collider. *Journal of Instrumentation* (2008).

[16] Collaboration, T. C. The CMS experiment at the CERN LHC. *Journal of Instrumentation* (2008).

[17] Collaboration, T. L. The LHCb Detector at the LHC. *Journal of Instrumentation* (2008).

[18] Dao, T., Fu, D., Ermon, S., Rudra, A., and Ré, C. Flashattention: Fast and memory-efficient exact attention with io-awareness. *Advances in Neural Information Processing Systems 35* (2022), 16344–16359.

[19] DeZoort, G., Thais, S., Duarte, J., Razavimaleki, V., Atkinson, M., Ojalvo, I., Neubauer, M., and Elmer, P. Charged particle tracking via edge-classifying interaction networks. *Computing and Software for Big Science 5* (2021), 1–13.

[20] Ester, M., Kriegel, H.-P., Sander, J., Xu, X., et al. A density-based algorithm for discovering clusters in large spatial databases with noise. In *kdd* (1996), vol. 96, pp. 226–231.

[21] Evans, L., and Bryant, P. Lhc machine. *Journal of Instrumentation 3*, 08 (aug 2008), S08001.

[22] Frühwirth, R. Application of kalman filtering to track and vertex fitting. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment 262*, 2-3 (1987), 444–450.

[23] Gabrielli, A., Alfonsi, F., and Del Corso, F. Hough transform proposal and simulations for particle track recognition for lhc phase-ii upgrade. *Sensors 22*, 5 (2022), 1768.

[24] Gaillard, M. K., Grannis, P. D., and Sciulli, F. J. The standard model of particle physics. *Reviews of Modern Physics 71*, 2 (1999), S96.

[25] Gillioz, A., Casas, J., Mugellini, E., and Abou Khaled, O. Overview of the transformer-based models for nlp tasks. In *2020 15th Conference on Computer Science and Information Systems (FedCSIS)* (2020), IEEE, pp. 179–183.

[26] Gupta, S., Agrawal, A., Gopalakrishnan, K., and Narayanan, P. Deep learning with limited numerical precision. In *International conference on machine learning* (2015), PMLR, pp. 1737–1746.

[27] Jiang, Z., Yin, D., Khoda, E. E., Loncar, V., Govorkova, E., Moreno, E., Harris, P., Hauck, S., and Hsu, S.-C. Ultra fast transformers on fpgas for particle physics experiments. *arXiv preprint arXiv:2402.01047* (2024).

[28] Ju, X., Murnane, D., Calafiura, P., Choma, N., Conlon, S., Farrell, S., Xu, Y., Spiropulu, M., Vlimant, J.-R., Aurisano, A., et al. Performance of a geometric deep learning pipeline for hl-lhc particle tracking. *The European Physical Journal C 81* (2021), 1–14.

[29] Kalman, R. E. A new approach to linear filtering and prediction problems.

[30] Karthikeyan, B., George, D. J., Manikandan, G., and Thomas, T. A comparative study on k-means clustering and agglomerative hierarchical clustering. *International Journal of Emerging Trends in Engineering Research 8*, 5 (2020).

[31] Khan, S., Naseer, M., Hayat, M., Zamir, S. W., Khan, F. S., and Shah, M. Transformers in vision: A survey. *ACM computing surveys (CSUR) 54*, 10s (2022), 1–41.

[32] Kiehn, Moritz, Amrouche, Sabrina, Calafiura, Paolo, Estrade, Victor, Farrell, Steven, Germain, Cécile, Gligorov, Vava, Golling, Tobias, Gray, Heather, Guyon, Isabelle, Hushchyn, Mikhail, Innocente, Vincenzo, Moyse, Edward, Rousseau, David, Salzburger, Andreas, Ustyuzhanin, Andrey, Vlimant, Jean-Roch, and Yilnaz, Yetkin. The trackml high-energy physics tracking challenge on kaggle. *EPJ Web Conf.* (2019).

[33] Kieseler, J. Object condensation: one-stage grid-free multi-object reconstruction in physics detectors, graph, and image data. *The European Physical Journal C 80* (2020), 1–12.

[34] Lantz, S., McDermott, K., Reid, M., Riley, D., Wittich, P., Berkman, S., Cerati, G., Kortelainen, M., Hall, A. R., Elmer, P., et al. Speeding up particle track reconstruction using a parallel kalman filter algorithm. *Journal of Instrumentation 15*, 09 (2020), P09030.

[35] Lazar, A., Ju, X., Murnane, D., Calafiura, P., Farrell, S., Xu, Y., Spiropulu, M., Vlimant, J.-R., Cerati, G., Gray, L., et al. Accelerating the inference of the exa. trkx pipeline. In *Journal of Physics: Conference Series* (2023), vol. 2438, IOP Publishing, p. 012008.

[36] Lebrun, P. Superconductivity and cryogenics for future high-energy accelerators. Tech. rep., 2007.

[37] Lee, J., Lee, Y., Kim, J., Kosiorek, A., Choi, S., and Teh, Y. W. Set transformer: A framework for attention-based permutation-invariant neural networks. In *International conference on machine learning* (2019), PMLR, pp. 3744–3753.

[38] Li, B., Pandey, S., Fang, H., Lyv, Y., Li, J., Chen, J., Xie, M., Wan, L., Liu, H., and Ding, C. 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.

[39] Lieret, K., and DeZoort, G. An object condensation pipeline for charged particle tracking at the high luminosity lhc. *arXiv preprint arXiv:2309.16754* (2023).

[40] Lieret, K., DeZoort, G., Chatterjee, D., Park, J., Miao, S., and Li, P. High pileup particle tracking with object condensation. *arXiv preprint arXiv:2312.03823* (2023).

[41] Mello, A. G., Dos Anjos, A., Armstrong, S., Baines, J., Bee, C., Biglietti, M., Bogaerts, J., Bosman, M., Caron, B., Casado, P., et al. Overview of the high-level trigger electron and photon selection for the atlas experiment at the lhc. In *14th IEEE-NPSS Real Time Conference, 2005.* (2005), IEEE, pp. 5–pp.

[42] Odyurt, U., Swatman, S. N., Varbanescu, A.-L., and Caron, S. Reduced simulations for high-energy physics, a middle ground for data-driven physics research. *arXiv preprint arXiv:2309.03780* (2023).

[43] Parisotto, E., Song, F., Rae, J., Pascanu, R., Gulcehre, C., Jayakumar, S., Jaderberg, M., Kaufman, R. L., Clark, A., Noury, S., et al. Stabilizing transformers for reinforcement learning. In *International conference on machine learning* (2020), PMLR, pp. 7487–7498.

[44] Qu, H., Li, C., and Qian, S. Particle transformer for jet tagging. In *International Conference on Machine Learning* (2022), PMLR, pp. 18281–18292.

[45] Ren, H., Dai, H., Dai, Z., Yang, M., Leskovec, J., Schuurmans, D., and Dai, B. Combiner: Full attention transformer with sparse computation cost. *Advances in Neural Information Processing Systems 34* (2021), 22470–22482.

[46] Shawahna, A., Sait, S. M., and El-Maleh, A. Fpga-based accelerators of deep learning networks for learning and classification: A review. *ieee Access 7* (2018), 7823–7859.

[47] Stewart, G., and Al-Khassaweneh, M. An implementation of the hdbscan* clustering algorithm. *Applied Sciences 12*, 5 (2022), 2405.

[48] Strandlie, A., and Frühwirth, R. Track and vertex reconstruction: From classical to adaptive methods. *Reviews of modern physics 82*, 2 (2010), 1419.

[49] Tsaris, A., Anderson, D., Bendavid, J., Calafiura, P., Cerati, G., Esseiva, J., Farrell, S., Gray, L., Kapoor, K., Kowalkowski, J., et al. The hep. trkx project: deep learning for particle tracking. In *Journal of Physics: Conference Series* (2018), vol. 1085, IOP Publishing, p. 042023.

[50] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, Ł., and Polosukhin, I. Attention is all you need. *Advances in neural information processing systems 30* (2017).

Appendix A

Other Two Transformer Methods

Our team explores three Transformer models with different purposes and designs. Below are further details on the other two architectures. Similarly to the Transformer Regressor, a separate model is trained for each data for these two approaches as well.

A.1 Autoregressive Transformer

This Transformer closely resembles the original Transformer architecture [50]. The encoder encodes the hits of a given event, and the decoder autoregressively rebuilds tracks within the same event. It requires a seed (a short starting sequence of hits) from which to build a particular track, and predicts which hits belong on the trajectory in question. Thus, this model builds tracks one by one, instead of using a one-shot approach as the other two Transformers.

The Autoregressive Transformer model uses fixed-query attention [37] in the first encoder stack in order to ensure full positional invariance of the set of input hits. While this model also omits positional encoding in the encoder, the decoder does use positional encoding since for the constructed track the order of hits is relevant. The decoder outputs a vector with (x, y, z) coordinates of the next hit in the track. This is the Transformer with highest amount of parameters of the three, with the model trained on TML:10-50 having 8.7 million parameters.

A.2 Classifier Transformer

This model uses an encoder-only, sequence-to-sequence Transformer architecture for the task of classification. Its input is a sequence of hit coordinates from a single event and its output is a sequence of corresponding class labels for each hit. The class labels are generated in advance by categorising track parameters into quantile-based balanced bins, such that all bins have a comparable amount of hits associated with them. In the case of multiple track parameters, each parameter is binned first, and each unique combination of bins makes up a single class. The binning for each dataset is different, with the linear and helical REDVID datasets using 5000 bins, and the TML:10-50 data using 1800 bins for the reported results.

This Transformer is similar to the Regressor Transformer in architecture, with the only differences being the dimensionality of the output layer, and utilized loss function: Binary Cross Entropy. The largest model is trained on TML:10-50 and has 969671 parameters.

Appendix B

Pipeline Details

B.1 Solving the Rotational Invariance Problem

As described in Chapter 5, whenever the model must regress ϕ as a track parameter, we map ϕ to $\sin(\phi)$ and $\cos(\phi)$. Otherwise, π and $-\pi$ are not treated as equal.

Both *sin* and *cos* are necessary, because using one of them only leads to loss of information. For instance, $sin(\phi) = 0$ for $\phi \in \{0, \pi, -\pi\}$ – meaning that all three angles would be treated as though they are the same. It is the case, however, that $cos(\pi) \neq cos(-\pi)$. Similarly, $cos(\phi) = 0$ for $\phi \in \{\frac{\pi}{2}, \frac{3\pi}{2}\}$ which is undesirable, but $sin(\frac{\pi}{2}) \neq sin(\frac{3\pi}{2})$. Therefore, whenever *sin* is the same, *cos* is different and vice versa, meaning that the two functions together sufficiently and properly differentiate the angle ϕ.

The effect and need of this preprocessing step can be observed in the visualization of regressed $\phi, cos(\phi), sin(\phi)$ plotted against the actual parameter values in Figure B.1. As we can see, when the model regresses only the angle ϕ, it often makes the mistake of predicting $-\pi$ when it should predict π and vice versa. This effect is seen a lot more rarely in the other two plots.

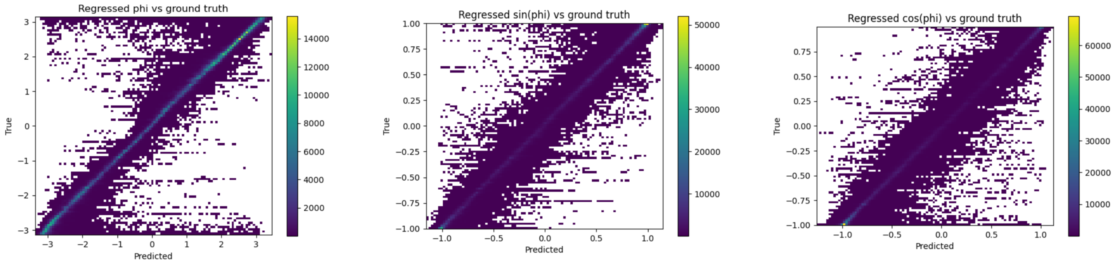


Figure B.1: Visualization of the regressed ϕ plotted against the actual ϕ for a model trained to regress θ, ϕ, q (left), and of the regressed $cos(\phi), sin(\phi)$ plotted against the actual values for a model trained to regress $\theta, sin(\phi), cos(\phi), q$ (middle, right). The models were trained on the TML:10-50 dataset.

B.2 Hyperparameters

We train a separate model for all ten utilized datasets. The structure of the Transformers is optimized per dataset, with the hyperparameters under consideration being embedding dimensionality, number of attention heads and hidden layer dimensionality. All models have 6 encoder layers with a dropout of 0.1 each, with the exception of the model trained on TML:200-500 which has 7 layers. A summary of the rest of the hyperparameters, as well as the total number of parameters per model are given in Table B.1.

These trained optimal models are included in the repository of this project.

Dataset	Embedding dimensionality	Nr. attention heads	Hidden layer dimensions	Total nr. parameters	Nr. epochs trained
3D-lin:3	32	8	128	76451	205
3D-lin:1-20	32	8	128	76451	443
3D-lin:10-50	32	4	128	76451	316
3D-hel:3	32	8	128	76451	112
3D-hel:1-20	32	8	128	76451	239
3D-hel:10-50	64	8	128	201283	490
3D-hel:50-100	64	8	256	300355	189
TML:10-50	32	4	128	76484	482
TML:50-100	32	4	128	76484	489
TML:200-500	128	4	256	928388	49

Table B.1: Hyperparameters of the Transformer Regressor models trained on each of the 10 datasets.

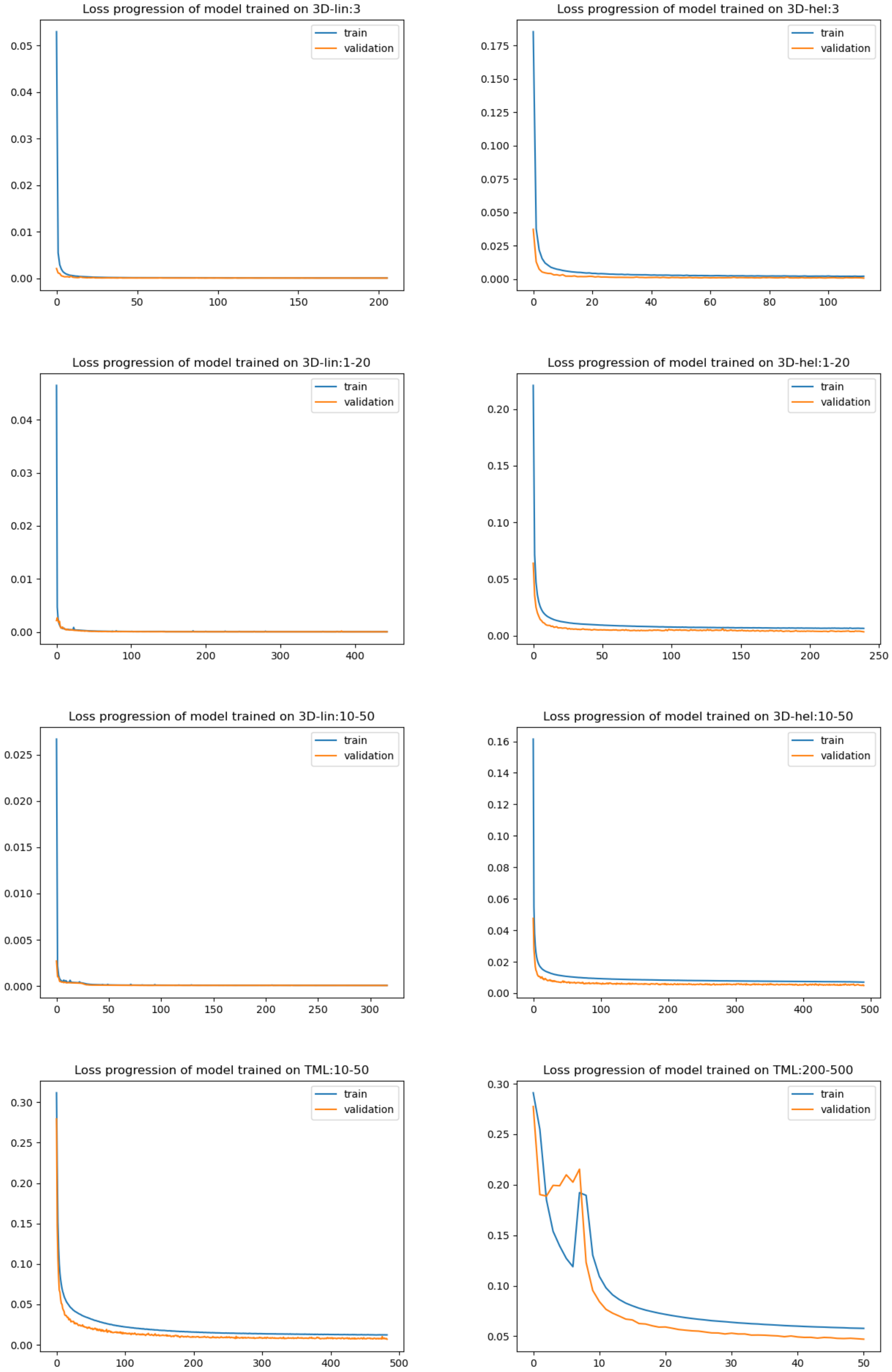
For each model and dataset, HDBSCAN has two parameters to tune: `min_cluster_size` and `min_samples`. A summary of the best found values for these two per dataset are given in Table B.2.

	3D-lin:3	3D-lin:1-20	3D-lin:10-50	3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100	TML:10-50	TML:50-100	TML:200-500
Minimum cluster size	2	4	4	2	5	4	4	5	5	5
Minimum nr. samples	3	2	2	3	2	2	2	2	3	2

Table B.2: The hyperparameters of the HDBSCAN clustering algorithm per dataset.

B.3 Loss Plots

For completeness, we include the plotted train and validation loss evolution of the models not presented in Chapter 6.



Loss progression of model trained on 3D-lin:3

Loss progression of model trained on 3D-hel:3

Loss progression of model trained on 3D-lin:1-20

Loss progression of model trained on 3D-hel:1-20

Loss progression of model trained on 3D-lin:10-50

Loss progression of model trained on 3D-hel:10-50

Loss progression of model trained on TML:10-50

Loss progression of model trained on TML:200-500

Figure B.2: The train and validation loss evolution of the models trained on datasets 3D:lin-3, 3D-hel:3, 3D-lin:1-20, 3D-hel:1-20, 3D:lin-10-50, 3D-hel:10-50, TML:10-50, TML:200-500.

B.4 Additional Tables

	TML:500-1000	TML:2000-5000	TML:all
TrackML score	0.45	0.2	0.06
ϵ^{perf}	0.24	0.07	0.008
ϵ^{DM}	0.64	0.32	0.11
ϵ^{LHC}	0.61	0.31	0.12

Table B.3: The TrackML score and the efficiency scores of the model trained on TML:200-500 with Flash attention when evaluated on the three additional, larger TrackML-derived datasets.

	3D-lin:3	3D-lin:1-20	3D-lin:10-50
$\sigma(\theta)$	0.004	0.003	0.003
$\sigma(sin(\phi))$	0.008	0.009	0.01
$\sigma(cos(\phi))$	0.009	0.009	0.01

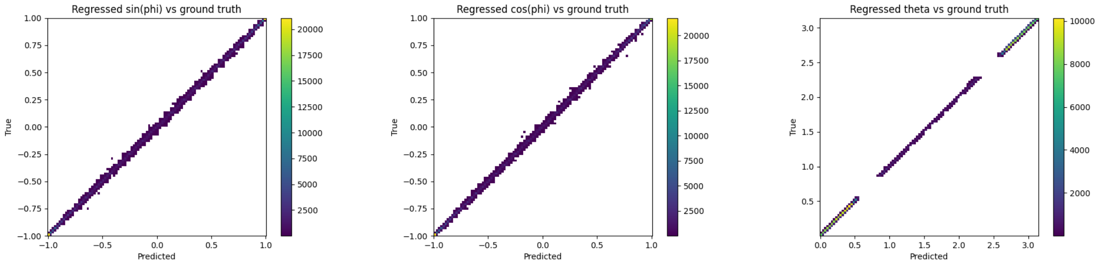
Table B.4: Standard deviation of the track parameters regressed by the models trained on the linear REDVID datasets.

	3D-hel:3	3D-hel:1-20	3D-hel:10-50	3D-hel:50-100
σ(radial coeff)	0.01	0.02	0.02	0.02
σ(pitch coeff)	0.03	0.05	0.07	0.08
σ(azimuthal coeff)	0.03	0.08	0.1	0.13

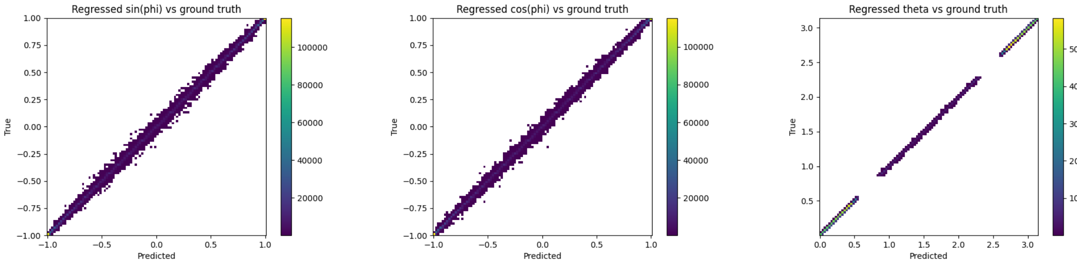
Table B.5: Standard deviation of the track parameters regressed by the models trained on the helical REDVID datasets.

B.5 Additional Figures

For completeness, we include the heatmaps of regressed track parameters vs ground truth of the models not presented in Chapter 6.



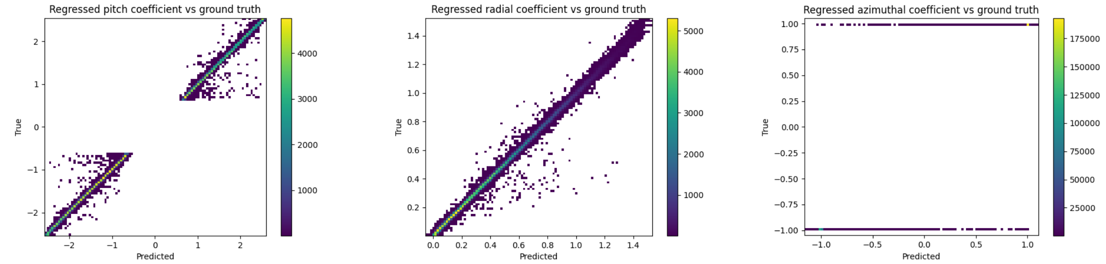
(a) Regressed track parameters vs ground truth parameters for the 3D-lin:3 dataset.


Regressed sin(phi) vs ground truth

Regressed cos(phi) vs ground truth

Regressed theta vs ground truth

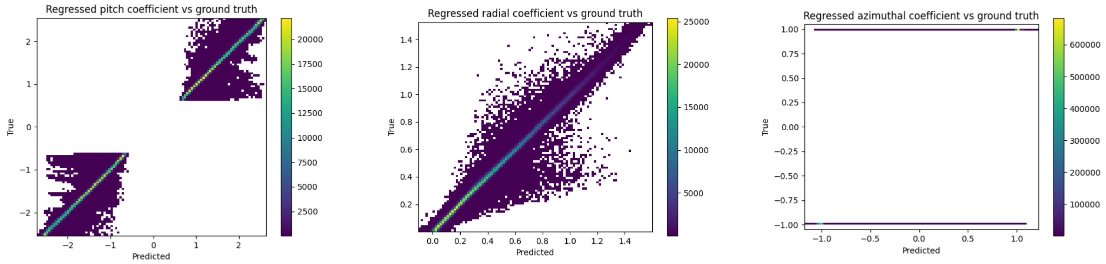
(b) Regressed track parameters vs ground truth parameters for the 3D-lin:1-20 dataset.


Regressed pitch coefficient vs ground truth

Regressed radial coefficient vs ground truth

Regressed azimuthal coefficient vs ground truth

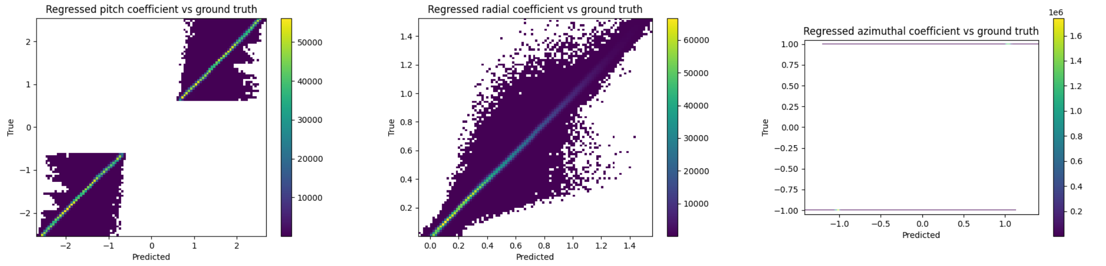
(c) Regressed track parameters vs ground truth parameters for the 3D-hel:3 dataset.


Regressed pitch coefficient vs ground truth

Regressed radial coefficient vs ground truth

Regressed azimuthal coefficient vs ground truth

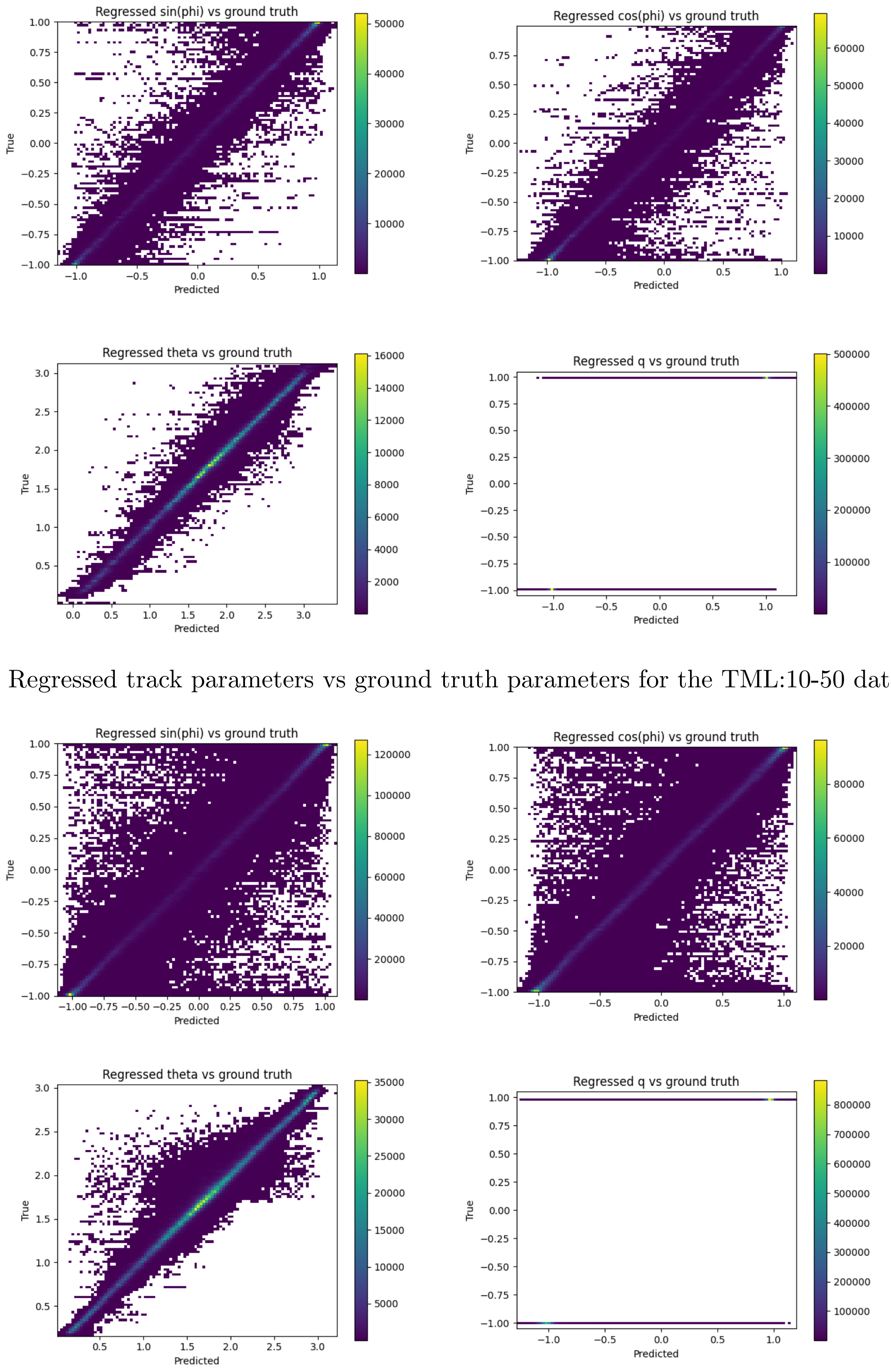
(d) Regressed track parameters vs ground truth parameters for the 3D-hel:1-20 dataset.


Regressed pitch coefficient vs ground truth

Regressed radial coefficient vs ground truth

Regressed azimuthal coefficient vs ground truth

(e) Regressed track parameters vs ground truth parameters for the 3D-hel:10-50 dataset.



(f) Regressed track parameters vs ground truth parameters for the TML:10-50 dataset.

(g) Regressed track parameters vs ground truth parameters for the TML:50-100 dataset.

Figure B.3: Visualization of the regressed track parameters by the Transformer plotted against the actual track parameter values found in the ground truth. White background signifies there are no points in that region. The ideal case is a thin diagonal line.

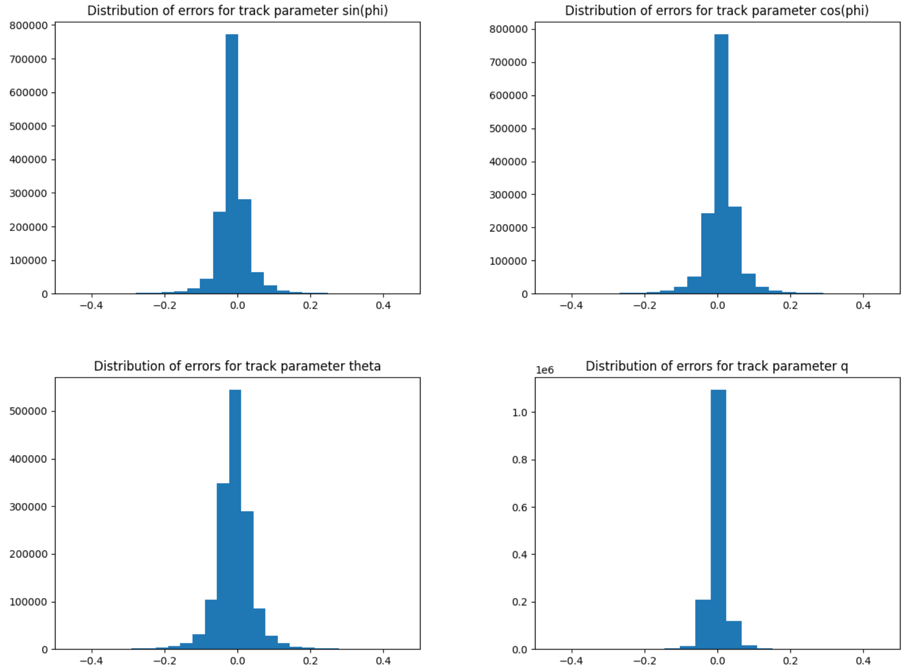


Figure B.4: Histograms of the errors made by the model trained on TML:10-50 for the four track parameters.

Appendix C

Methods for Future Work

Below we describe multiple ideas for potential improvement of the performance of the track finding pipeline. We believe they are worth further investigation in the future, and we have already partly or fully implemented them, but not used them in the main experiments.

C.1 Domain Decomposition

Domain decomposition can narrow down the size of the input, and also enable the model to capture dependencies of higher resolution. The idea is to split the events into subevents of smaller size in the following manner: For every hit in an event, the x, y, z Cartesian coordinates are used to calculate the θ, ϕ coordinates in the cylindrical system. These two are binned each in X bins, making in total $X \times X$ bins of $\theta - \phi$ combinations which we associate hits with. The boundaries of the bins are based on the distributions of the two variables.

Furthermore, we introduce overlap in the subevents, so that if by default a bin in θ encompasses values between 0.5 and 0.7, an overlap of 0.1 would lead to the bin including hits with $\theta \in [0.4, 0.8]$. The exact amount of overlap is determined based on the trade-off between splitting up as few as possible tracks over multiple bins and actually reducing the size of the problem. To assess the efficiency of the split, for every track t we calculate how well it is represented using the following formula: $max(\forall\ b \in bins\ \frac{\#\ hits\ of\ t\ in\ b}{\#\ total\ hits\ of\ t})$, and then take the average value over all tracks.

Everything described above is fully implemented and included in the public repository of this project. This is one of the author's contributions to the project, and it was further developed by another team member to split the track parameter space not into equally sized bins but into bins based on the parameters' distribution. We do not utilize domain decomposition in the experiments reported in this paper, but we believe that it will be very useful when dealing with larger amount of data, with more than 500 tracks per event. Not only will it enable the Transformer to work with simpler data with higher resolution, but it will also allow us to run multiple copies of the model on different subdomains in parallel, speeding up the execution time significantly.

C.2 Incorporating Angle Difference

While attempting to address the issue of the symmetry of ϕ, we explored using the angle difference instead of the absolute difference between values in the calculation of MSE. The proposed new loss function, Angle Difference Loss (ADL), is as follows:

$$\frac{\sum_y ((y - y_{pred} + \pi)\ mod\ 2\pi) - \pi}{\#\ y}$$

This loss function relies on the prediction and true value being angles, but q (one of the track parameters used in TrackML-derived datasets) is not. This leaves room for future work to investigate how ADL can be utilized. Due to time constraints, we do not integrate this loss function into our pipeline, and instead use $sin(\phi)$ and $cos(\phi)$ as track parameters instead of ϕ to bypass the angle symmetry issue. However, we believe that using ADL is rooted in physics and might be more appropriate than using the trigonometrical functions, as it does not introduce an extra parameter to be regressed.

We can make both parts of the overall pipeline be physics informed. As most clustering algorithms, HDBSCAN included, allow for the definition of a custom distance metric, we believe that this formula can be further utilized during clustering. However, to do so, we are left with the same issue to solve as the one of ADL.

C.3 Refining Network

Taking advantage of the speed of our proposed approach, the physics performance can be increased by adding another component to the pipeline. One possible way to do that is use a model similar to the Autoregressive Transformer investigated by our team, which can take a fully-formed cluster of hits as a seed and attempt to identify more hits within the event which lie on the same track to help a more accurate reconstruction.

Another approach is to use a one-shot model, similar to the other two approaches we develop, which does not add hits, but instead removes wrongly assigned ones, thus increasing the purity of each cluster. This can be seen as a classification task, where the model takes a group of hits and classifies them as belonging to that cluster or not. Or the network can provide a probability estimate of the belonging of each individual hit to the track. We investigate a third option: using our own Transformer Regressor again to map a sequence of hits to a sequence of track parameters. However, the Refiner does so per cluster, instead of per event, meaning that it can regress the track parameter values more accurately because of the smaller sequence length and increase in resolution.

This idea is implemented and can be found in our repository. The network is also trained on the ground truth data that the Transformer Regressor gets trained on. At inference time, the Refiner is fed a cluster of hits, which it maps to track parameters. The median track parameter set is found based on these regressed values, and the MSE of the median and each regressed track parameter set is calculated. Finally, according to a cut-off value, the track parameters whose MSE is highest are removed from the cluster. This cut-off value is found empirically, using the validation set of the data. The Refiner becomes part of the pipeline, run right after HDBSCAN to refine the clusters, and its output is used for the calculation of the four metrics.

Although fully implemented, we do not make use of the Refiner in the experiments reported in this paper as due to time constraints we do not optimize the model in terms of hyperparameters and cut-off value, and do not investigate its applicability across different datasets and data complexities. Furthermore, we theorize that removing wrongly assigned hits might not increase physics efficiency a lot according to the utilized metrics, as e.g., the TrackML score is calculated by summing over the weights of correctly assigned hits, meaning that removing the incorrect ones does not always increase the final result. The real benefit to using such a network after the clustering algorithm will be seen once a track fitter model is added the the pipeline, and we can evaluate whether refining the clusters aides in finding the actual track defining parameters. We believe that this direction is worth investigating, including the design of the aforementioned autoregressive type of refiner network.