End-to-End Pipeline for Trigger Detection on Hit and Track Graphs

Tingting Xuan[1], Yimin Zhu[1], Giorgian Borca-Tasciuc[2], Ming Xiong Liu[3], Yu Sun[2], Cameron Dean[3], Yasser Corrales Morales[3], Zhaozhong Shi[3], Dantong Yu[4]

[1] Stony Brook University
[2] Sunrise Technology Inc.
[3] Los Alamos National Laboratory
[4] New Jersey Institute of Technology
{tingting.xuan,yimin.zhu}@stonybrook.edu, {giorgian.borca-tasciuc,yu.sun}@sunriseaitech.com,
{mliu,ctdean,ycorrales,zhaozhongshi}@lanl.gov, dantong.yu@njit.edu

Abstract

There has been a surge of interest in applying deep learning in particle and nuclear physics to replace labor-intensive offline data analysis with automated online machine learning tasks. This paper details a novel AI-enabled triggering solution for physics experiments in Relativistic Heavy Ion Collider and future Electron-Ion Collider. The triggering system consists of a comprehensive end-to-end pipeline based on Graph Neural Networks that classifies trigger events versus background events, makes online decisions to retain signal data, and enables efficient data acquisition. The triggering system first starts with the coordinates of pixel hits lit up by passing particles in the detector, applies three stages of event processing (hits clustering, track reconstruction, and trigger detection), and labels all processed events with the binary tag of trigger versus background events. By switching among different objective functions, we train the Graph Neural Networks in the pipeline to solve multiple tasks: the edge-level track reconstruction problem, the edge-level track adjacency matrix prediction, and the graph-level trigger detection problem. We propose a novel method to treat the events as track-graphs instead of hit-graphs. This method focuses on intertrack relations and is driven by underlying physics processing. As a result, it attains a solid performance (around 72% accuracy) for trigger detection and outperforms the baseline method using hit-graphs by 2% higher accuracy.

Introduction

Physics Background

sPHENIX is a high energy nuclear physics experiment situated on the Relativistic Heavy Ion Collider (RHIC) and it is still under construction at Brookhaven National Laboratory. The goal of the experiment is to probe the Quark-Gluon Plasma, which is a state of matter where atomic nuclei melt under extremely hot and dense conditions prevailing in the Universe at about 20 μs after the Big Bang. Figure 1 shows the details of the sPHENIX detectors.

In high-energy nuclear and particle physics experiments, a trigger should be able to decide whether an event is of our interest with high efficiency. Among the proton+proton collisions at the sPHENIX experiments, the triggering criteria for interesting events, which are heavy charm quark events, are based on the patterns of the charm-hadron decays into a pair of oppositely charged kaon and pion particles. The particles of interest decay on very short time scales, typically a few nanoseconds or less. However, as the particles are produced close to the speed of light, they can travel several millimeters before decaying. Particle tracks can be extrapolated back in space to see if they coincide with the beam collision point or not. The detector is essentially a high-speed camera, continuously taking 250-megapixel images for proton+proton collisions happening at approximately 10MHz. Then the triggering system inspects image patterns to decide whether to keep or discard each image. The triggering system can greatly reduce data volume that needs to be collected from physics experiment.

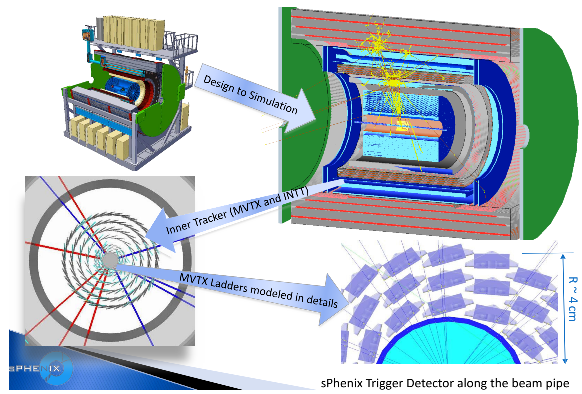

Figure 1: sPHENIX Track Detector and Trigger Design.

The traditional Minimum Biased trigger for studies of fully-reconstructed beauty hadrons is very inefficient because the purity of the filtered data is very low. Currently, no fully-reconstructed beauty hadrons measurement is reported at RHIC. No efficient deterministic method identifies heavy flavor decay, and the occurrence of events has a significant amount of randomness, providing an ideal playground for applying Machine Learning techniques.

In this paper, we report our work-in-progress machine learning enabled software triggering system that will be deployed and commissioned in 2023. We use validated simulation results to demonstrate the feasibility of using machine learning to learn physics event embeddings, model intrinsic interactions among particles generated in physics events, and classify events as "triggers" verse "backgrounds".

Challenges and Contributions

There are two main challenges in designing and implementing an online event selection algorithm. The first is to provide an end-to-end solution that uses raw detector readout to make trigger decisions for data collection. The second is to design a neural network compatible with detector readout and capable of learning a broad spectrum of physics properties, using low-level hits to build up to the high-level trigger decision. We overcome these two challenges by using a three-stage pipeline, attaining impressive performance in online trigger prediction. The pipeline has three stages: 1) clustering the pixel readings into hits, 2) connecting the hits that belong to identical particles to reconstruct tracks, and 3) dynamically constructing track-graphs and making the graph-level decisions. The main contributions for this paper are as follows:

1. Our event selection pipeline only needs the MVTX fast detector readouts and undertakes multiple tasks that usually appear in offline data analysis and require accessing the reconstructed event data from various (slow) detectors that are not available during real-time data taking. Our work confirms the feasibility of moving these offline analysis tasks to online systems for intelligent data collection.
2. The pipeline is highly effective in reducing event readouts in all three stages: the clustering algorithm removes redundant hits, the tracking algorithm eliminates noise hits with no meaningful connections, and the event building algorithm creates smaller track-graphs than the hit-graphs in the baseline algorithm.
3. We propose a novel method to treat the events as track-graphs instead of hit-graphs. The track-graphs are fully aware of the physics principle and align well with physics data analysis, identify interesting physical events with great accuracy, and outperform the baseline methods hit-graphs by more than 2% in terms of prediction accuracy.

Related Works

Graph Pooling Techniques

Pooling is a critical step in Convolution Neural Networks (CNNs). To generalize CNNs to Graph Neural Networks (GNNs), we believe an appropriate pooling mechanism compatible with network topology can effectively extract graph representations. Graph pooling is still in an exploration stage, and only a handful of efforts in literature are related to this topic.

Three types of pooling methods exist: topology-based, global, and hierarchical pooling. 1) For topology-based pooling, Dhillon (Dhillon, Guan, and Kulis 2007) proposed to eliminate the time-consuming eigendecomposition and instead cluster graph nodes because of the mathematical equivalence between the objective of general spectral clustering and a weighted kernel k-means algorithm. Recent works (Defferrard, Bresson, and Vandergheynst 2016; Rhee, Seo, and Kim 2017) also use this clustering approach as the graph pooling layers in their GNNs. 2) Global pooling, for example, Set2set (Vinyals, Bengio, and Kudlur 2015) and SortPooling (Zhang et al. 2018) essentially uses the summation, maximum, or minimum to pool all graph nodes for graph representation. 3) Hierarchical pooling has been receiving attention recently. Ying et al. proposed DiffPool (Ying et al. 2018), a differentiable graph pooling method that learns assignment matrices and generates hierarchical representations of graphs in an end-to-end fashion. Lee introduced SAGPool (Lee, Lee, and Kang 2019) that considers both the node features and graph topology and uses a graph-convolution-based self-attention mechanism to achieve hierarchical graph pooling.

Machine Learning in Particle Physics

Guest et al. summarize recent efforts in using advanced statistical analysis, machine learning, and deep learning methods to analyze event data in Large Hadron Collider particle physics experiments (Guest, Cranmer, and Whiteson 2018). The majority of these efforts focus on removing noise, reconstructing tracks between different detectors, and identifying whether tracks are generated by electron, photon, or τ lepton (Kazeev 2020). These algorithms gain impressive success in identifying low-level (local-level) physics properties, such as hit identification and tracking reconstruction. There are multiple attempts on event selection and other high-level physics tasks (Casa and Menardi 2018; Baldi, Sadowski, and Whiteson 2014).

End-to-End Trigger Detection Pipeline

The problem we solve is a classification problem. It starts with the raw data recorded by the detectors, which only contains the indices of the lightened pixels on the detectors and the corresponding coordinates. The goal of the problem is to classify the event as either a trigger event or a background event. The output of the pipeline is the estimated probability for the event being a trigger event.

Our pipeline is consistent with the physics analysis workflow and it reflects the interaction between particles and detectors. It consists of three stages:

1. Clustering pixels into hits using their geometric information.
2. Connecting hits to form tracks.
3. Constructing event graph dynamically and performing graph-level classification for triggers.

Our pipeline extracts the information and reduces the event size step by step. We aggregate the information from pixels to hits, from hits to tracks, and from tracks to a fixed-length vector representation of the whole event. Each of these steps tries to predict critical physics properties that benefit trigger identification in traditional physics methods. Figure 2 details our pipeline with an example event.

Pixel Clustering

A particle might strike a cluster of pixels when it flies through the detector with different energies and momenta.

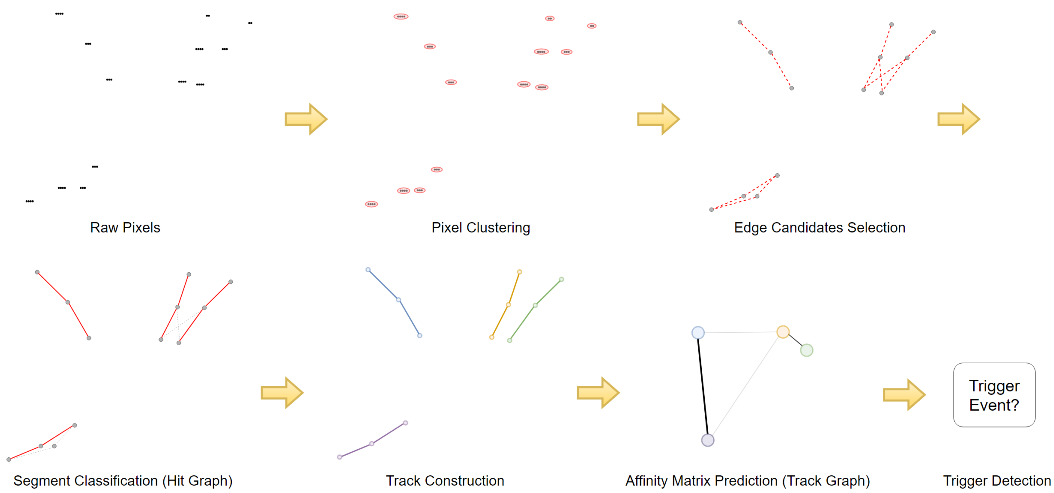

Figure 2: Trigger Detection Pipeline with All the Steps.

These pixels must be grouped and represented by a cluster medoid. We propose a deterministic algorithm to segment the contiguous pixels into separate clusters. Many traditional clustering algorithms incur an $O(n^2)$ time complexity due to needing to calculate the pair-wise distance for clustering. However, we take advantage of the fact that the pixels activated by the same particle must be adjacent in the detector space and design a connected component algorithm with a hash table that hashes pixel indices to memory locations for the efficient search for neighboring pixels. The clustering algorithm has a moderate time complexity of $O(n)$. The clustering algorithm takes in the set of pixels as input and outputs a set of hits. The total number of hits after clustering varies between events.

Track Reconstruction

Once we clean up the hits and remove redundant pixels in each event, we reconstruct the tracks by connecting the hits across different detector layers to recover the trajectories of the particles. By connecting hits in the order of a particle's interaction with detector layers, from inner layer to outer layer, we recover the trajectory of particles. We model the input as the cloud of hits and dynamically construct a graph with nodes (hits) and edges (track segments) to represent a collision event. Then we perform information propagation and convolution with the constructed graphs.

We adopt the Message-Passing GNN proposed by (Farrell et al. 2018) and divide the track reconstruction into two steps. The first step is to connect hits belonging to consecutive layers into hit pairs. To prevent excessive connections, we apply geometric constraints and down select the hit pairs into edge candidates. After selecting the edge candidates, the tracking problem is simplified to a binary edge classification problem (link prediction) to predict real track segments. In the second step, we iteratively apply the powerful graph convolution operations to propagate hit information among nodes via candidate edges and enrich each hit representation with multi-hop neighborhood information as shown in

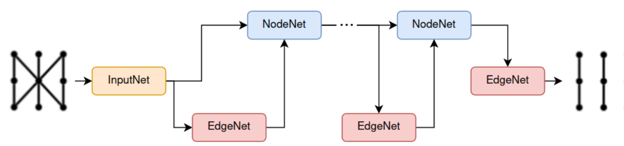

Figure 3: Graph Neural Network for Tracking

Figure 3. Finally, we use a Multilayer Perceptron (MLP) on the two nodes connected by each candidate edge and predict whether the candidate edge belongs to the ground truth track segments. The GNN ensures that the graph-level information is learned during the information diffusion processing, and the pairwise link prediction is performed in the context of the entire event graph.

Trigger Detection

After the previous step, we have built the track segments between hits and we are able to classify events as a trigger or non-trigger using a baseline GNN that treats an event as a graph with hits as nodes and track segments as edges. But from the physics perspective, trigger detection requires more sophisticated features than the hit-level information used in the baseline models. For example, physicists often need to analyze the relation between tracks and whether they are from the same origin points. Inspired by this physics process, we build intertrack links and perform graph classification on a track-based graph (tracks as nodes). It is more consistent with the physics logic to solve the trigger detection problem compared to the hit-graph. Also, the track-based graph is much smaller, benefiting our goal for fast inference and moderate storage space.

We propose two methods to build tracks. Given all predicted track segments between hits, we apply depth-first traverse to find all the connected hits and chain them into a track. Alternatively, we select the hits in a track by finding the longest non-cyclical path in the connected component. We also add other geometric features, such as edge lengths

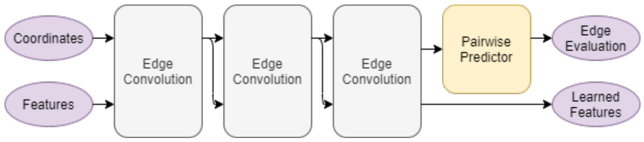

Figure 4: Cluster of Tracks

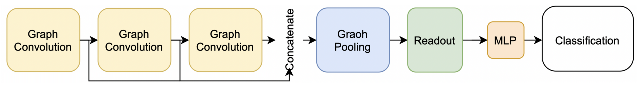

Figure 5: Trigger Prediction (global)

and angles, to help the model perform better.

Dynamic adjacency matrix generation for tracks using ParticleNet. Given a collection of n tracks for one event, we need to identify whether any pair of tracks share the same original vertex. We define a connection between tracks if they have the same original vertex. Here, the original vertices are dependent variables and only available for defining loss functions during the training stage. We improve upon the ParticleNet (Qu and Gouskos 2020) and combine a balanced Cross-Entropy Loss and a weighted Laplacian Loss to predict the track adjacency matrix and to construct a track-graph.

Figure 4 shows the clustering network based on ParticleNet. ParticleNet is a neural network architecture that consists of EdgeConv (Wang et al. 2019) and operates directly on particle clouds. The EdgeConv defines a message passing between node x_i and its neighborhood:

$$x_i' = \square_{j=1}^{k} h_\Theta(x_i, x_{ij}), \quad (1)$$

where $x_i \in \mathbb{R}^F$, k is the number of neighboring points of each point, h is the edge function on learnable parameters Θ and $\square$ is an aggregation function. Two hyperparameters characterize an EdgeConv block: the number of neighbors k and the number of channels C in each block.

ParticleNet substitutes the neighbors' feature vector in the Equation 1 and calculates the difference between a node and its central point

$$h_\Theta(x_i, x_{ij}) = \tilde{h}_\Theta(x_i, x_{ij} - x_i)$$

, where $\tilde{h}_\Theta$ in the EdgeConv block is implemented by a MLP with neuron weight Θ. The aggregation function is to calculate the mean of inputs. The stackable edge convolution blocks ensure the deep network learns the features of point clouds hierarchically and approximates the proximity of tracks dynamically.

Finally, we apply a pairwise predictor that takes the learned node embeddings from ParticleNet and predicts the connectivity of all tracks pairs. Given m points as input, it outputs a $m \times m$ matrix where each element is the predicted linkage probability between two tracks.

Trigger prediction via SAGPool. We design a graph-level prediction model that classifies the input event graphs

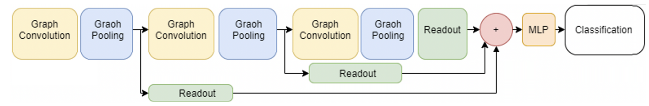

Figure 6: Trigger Prediction (hierarchical)

into the trigger and non-trigger events based on the predicted inter-track connection and track features from the previous stage. Figures 5 and 6 show the architecture of event-level prediction using global and hierarchical architecture. The algorithm employs a pooling method that aggregates any number of node features into an event representation with a predefined size. Among multiple choices of pooling method, we adopt SAGPool (Lee, Lee, and Kang 2019) to categorize each event due to its simplicity and efficiency.

SAGPool uses graph convolutions to calculate the self-attention scores, and considers both node features and graph topology while incurring only moderate time and spatial computational complexity. Self attention with graph convolution, allows the pooling method to consider both node features and graph topology to yield hierarchical representations with a reasonable complexity of time and space. SAGPool utilizes the widely used graph convolution (Kipf and Welling 2016) defined on the normalized graph Laplacian $L = I - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$. The forward propagation in layer l is defined as:

$$h^{l+1} = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} h^l \Theta_{MLP}),$$

where h^l represents the input node features of the l-th layer, σ is the activation function, $\tilde{A} \in R^{N \times N}$ is the adjacency matrix with self-loop (i.e. $A + I_N$), $\tilde{D} \in \mathbb{R}^{N \times N}$ is the degree matrix of $\tilde{A}$, $\Theta_{MLP} \in \mathbb{R}^{F \times F'}$ is the weight matrix of the feature transformation network, N is number of nodes, F is the input feature dimension and F' is the output feature dimension.

SAGPool uses self-attention on the layer transformation on node features before exporting the output to the next layer. The self-attention score Z is:

$$Z = \sigma(GNN(X, A)) = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A} \tilde{D}^{-\frac{1}{2}} X \Theta_{att}),$$

where $X \in \mathbb{R}^{N \times F}$ is the input features, $\Theta_{att} \in \mathbb{R}^F$ contains the learnable attention parameters of the pooling layer. SAGPool utilizes graph convolution to obtain self-attention scores Z on each node and apply the attention score to retain the most significant nodes in the input graph nodes for pooling. As a result, SAGPool takes into consideration the graph features and topology. Here, the pooling ratio $p \in (0, 1]$ is used to select $\lceil pN \rceil$ nodes with the top Z scores.

Further Improvements

Distance Metric in ParticleNet

To search the nearest neighbor, we need to have a definition for distance. A track consists of several hits and is represented by a concatenation of the hit coordinates. Hence, the direct distance between the two tracking vectors is not necessarily the closest distance between the two formed lines.

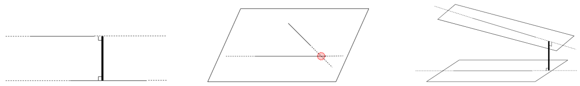
(a) Parallel Lines (b) Non-parallel Coplanar lines (c) Non-parallel Non-coplanar lines

Figure 7: Intertrack Distance is defined as the shortest distance between two line in three dimensions.

We adopt inter-track displacement in the physics analysis as our distance metric in ParticleNet. Figure 7 shows the inter-track distance under different circumstances.

Loss Function for Track Adjacency Matrix

The objective of ParticleNet is to minimize the error in link prediction. We first apply the Cross-Entropy Loss to penalize the prediction error. The Cross-Entropy Loss is sensitive when two tracks have a small distance and renders the link prediction with high variance. Since the displacement between the primary and secondary vertices in most events is no more than a few hundred microns, we adopt the weighted graph Laplacian Loss as a regularization term in the objective function to penalize false inter-track connections. In graph theory, a graph can be represented by adjacency matrices without any loss of information. The loss function is:

$$\mathcal{L} = \mathcal{L}_{CE}^{*} + \alpha \cdot \mathcal{L}_{Laplacian}$$

where $\mathcal{L}_{CE}^{*}$ is calculated on a balanced training dataset and α is the weight factor for Laplacian loss. $\mathcal{L}_{CE}$ for the binary classification of n tracks is:

$$\mathcal{L}_{CE} = -\frac{1}{n}\sum_{i=1}^{n}[y_i \cdot \log \hat{y}_i + (1 - y_i) \cdot log(1 - \hat{y}_i)]$$

Our dataset is balanced for trigger/nontrigger events. With each graph, a link between two tracks indicates that the two tracks share the same primary or the secondary vertex in Figure 7b. The number of edges is far greater than that of spurious edges. To ensure a balanced penalty on real and spurious edges in the events with decay in Figure 7c, we apply $\mathcal{L}_{CE}$ on the subgraph with the same number of edges from two classes and denote it as $\mathcal{L}_{CE}^{*}$. For non-decay events, we use all the edges.

The Laplacian Loss is defined by the Laplacian matrix $L = D - A$, where D is the degree matrix and A is the adjacency matrix. The Laplacian matrix is symmetric and positive semi-definite, which satisfies the bilinear term $x^T Lx \geq 0$ for $x \in \mathbb{R}^{|v|}$ (Hamilton 2020). The proposed graph Laplacian regularization helps ensure that two interconnected tracks have the same secondary vertex:

$$\mathcal{L}_{laplacian} = trace(O^{\top}(\tilde{D} - \tilde{A})O)$$

where $O \in \mathbb{R}^{n \times 3}$ is the ground truth coordinate matrix of the secondary vertex of each track, $\tilde{D}$ is the degree matrix, and $\tilde{A}$ is the learned affinity matrix consisting of predicted links by ParticleNet.

Experiment Results

Our experiment utilizes a simulation dataset (Sample data files can be found at: https://github.com/sPHENIX-Collaboration/HFMLTrigger). The simulation is generated by the PYTHIA8.3 package (Sjöstrand, Mrenna, and Skands 2008) and the physical quantities are measured by the sPHENIX detector with GEANT4 simulation toolkit (Agostinelli et al. 2003; Allison et al. 2006). PYTHIA8 is a software package simulating QCD processes in small collision systems and has been widely validated at many colliders (Aguilar et al. 2022; P. Skands 2014; Skands 2010). GEANT4 is a package that simulates the passage of particles and radiation through matter. For this project, only the three-layer MVTX and two-layer INTT detectors are simulated as they are the only detectors capable of operating fast enough to achieve the goal of this project. They are also the two innermost detectors and so simulating further detectors would needlessly utilize time and resources. It should also be noted that as the collisions occur within the RHIC beam pipe, this object is also simulated along the lengths of the detectors. The simulated data is expected to match with the real data at a high level which has been extensively validated from previous physics experiments. Our model will be used to filter data in the real experiment that will start taking data in February 2023. Experiments are run on an NVIDIA Titan RTX and GeForce RTX 3090.

Tracking

We sample the events and built a balanced training data of trigger events and background events. Table 2 shows that the GNNs have impressive performance with more than 90% accuracy in identifying tracks. Furthermore, the accuracy is highly dependent on the number of layers and hidden dimensions of the MLPs in GNNs. A larger number of parameters might lead to higher accuracy while incurring more considerable variance in prediction, more complex parameter space, longer training time, and higher costs for inference compared with the case of a lower dimension. To generate the best tracking for the next step, we choose the hidden dimension to be 32, four MLP layers, and four iterations of message passing across all hits.

Trigger Detection

For the experiment settings, we use 600,000 events for training and 200,000 events for validation to induce the trigger prediction model. We adopt the Adam optimizer and 50 epochs for all the training experiments.

Track adjacency matrix generation by ParticleNet. The modified loss function for ParticleNet has a weighted Laplacian loss. An optimal value for the Laplacian Loss Factor must be determined. Figure 8 shows that the performance improves with properly tuned weights for the Laplacian loss. Performance without Laplacian loss is represented by the blue dotted line. When the Laplacian weight factor is $\leq$ 1e-6, both the Auc score and Accuracy outperform the model trained without the Laplacian loss. We chose a Laplacian weight factor of 1e-6 for the downstream task.

Input	Model	Track Generation	Hidden dim	#Parameters	Accuracy	AUC	Recall	Precision
hit-graph	GNN-Sum	-	16	1969	68.6%	74.89%	68.60%	68.65%
hit-graph	GNN-Sum	-	8	826	68.69%	74.7%	68.68%	68.72%
hit-graph	DiffPool-1Layer	-	32	3183	69.81%	76.42%	69.81%	69.81%
hit-graph	DiffPool-2Layer	-	16	5957	69.94%	76.39%	69.94%	69.97%
hit-graph	SAGPool	-	32	12342	67.90%	73.79%	67.90%	67.91%
track-graph	SAGPool	Connected Component	16	4190	71.00%	77.86%	73.19%	70.11%
track-graph	SAGPool	Connected Component	32	14262	71.46%	78.88%	73.08%	70.72%
track-graph	SAGPool	Connected Component	256	773510	71.93%	79.18%	75.27%	70.58%
track-graph	SAGPool	Longest Path	256	773510	71.86%	79.22%	71.40%	72.07%

Table 1: Trigger Detection Performance

Hidden dim	Layers	#Parameters	Accuracy
8	2	457	93.2%
8	4	745	94.1%
16	2	1681	94.89%
16	4	2769	96.1%
32	2	6433	95.5%
32	4	10657	96.3%

Table 2: Tracking Graph Neural Network Performance

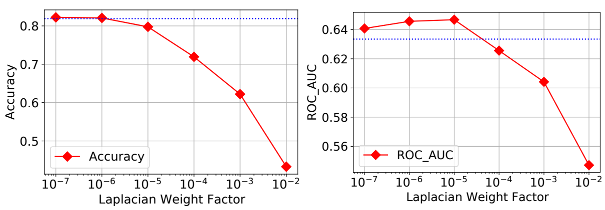

Figure 8: Performance with Different Laplacian Weights

We perform a grid search on the learning rate ($1e-1$, $1e-2$, $1e-3$, $1e-4$) and the number of neighbors (5, 9, 16) to obtain the best model.

We observe that the accuracy of linkage prediction increases as the number of neighbors increases. Another observation is that the model performance depends on the training dataset size. The larger the dataset, the better performance the modified ParticleNet obtains.

There are three EdgeConv blocks in the model, associated with three channel parameters. The final settings for the model to be fed into the next stage are learning rate (0.001), k(16), C_1(64), C_2(128), C_3(256), and dropout(0.1).

Trigger Detection with SAGPool on track-graph. We perform another grid search on hyperparameters in Table 3 to find the best setting for SAGPool at the second stage, where H stands for hierarchical and G stands for global.

Hyperparameter	Range
Learning rate	1e-1, 1e-2, 1e-3, 1e-4
Pooling ratio	0.25, 0.5, 0.75
Dropout ratio	0.5, 0.7
Pooling type	G, H

Table 3: Grid search for SAGPool

Baselines. We compare our result with some baselines.

- **Simple Aggregation.** Simple aggregation simply performs node-wise aggregation on the GNN-updated node embeddings to get a global graph representation. The general aggregation functions include summation, average, maximum, minimum, self-attention.
- **DiffPool Hierarchical Pooling.** Nodes can be aggregated by the bottom-up hierarchical pooling method from (Ying et al. 2018) that preserves the original event structure and generates event-level embeddings.

We evaluate the results of GNN with two types of readout functions (sum and maximum), DiffPool and SAGPool for hidden dimensions 8, 16 and 32.

Our best results after grid search for different models are summarised in Table 1. From the table we can see, the accuracy of trigger detection on track-graphs noticeably surpasses the best result we get on hit-graphs by more than 2%. Most of our experiments on track-graphs reach a similar performance as the one we showed in the table. With a hidden dimension of 32, pooling ratio 0.75, dropout ratio 0.5, SAGPool with hierarchical pooling architecture can reach accuracy of 71.46%. With a hidden dimension of 32, pooling ratio 0.75, dropout ratio 0.5, SAGPool with hierarchical pooling architecture can reach accuracy of 71.00%.

Conclusion

The data generated by the sPHENIX experiment outpaces the existing Data Acquisition Systems and necessitates intelligent event tagging and filtering. This paper addresses this challenge and provides an end-to-end solution for an online sPHENIX trigger development. All existing efforts focus on offline reconstruction and analysis once experiment data is collected and transferred to data centers and clouds. Our endeavor is one of the first attempts to shift many offline tasks, such as noise removal, event reconstruction, and tagging, to the early stage of data collection. This paper proposes a novel idea of treating the events as track-graphs instead of hit-graphs that is consistent with the governing physics of particle collision, formation and decay. It also demonstrates that physics-informed data mining improves the trigger detection accuracy by 2%. Our work provides proof-of-concept for software-based intelligent triggers for future physics experiments and expedites the turn-around from experiment to scientific discovery.

References

Agostinelli, S.; et al. 2003. Geant4: A simulation toolkit. *Nucl. Instrum. Meth.*, A506: 250.

Aguilar, M. R.; Chang, Z.; Elayavalli, R. K.; Fatemi, R.; He, Y.; Ji, Y.; Kalinkin, D.; Kelsey, M.; Mooney, I.; and Verkest, V. 2022. PYTHIA 8 underlying event tune For RHIC energies. *Physical Review D*, 105(1).

Allison, J.; Amako, K.; Apostolakis, J.; Araujo, H.; Dubois, P.; et al. 2006. Geant4 developments and applications. *IEEE Trans.Nucl.Sci.*, 53: 270.

Baldi, P.; Sadowski, P.; and Whiteson, D. 2014. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1): 1–9.

Casa, A.; and Menardi, G. 2018. Nonparametric semisupervised classification for signal detection in high energy physics. *arXiv preprint arXiv:1809.02977*.

Defferrard, M.; Bresson, X.; and Vandergheynst, P. 2016. Convolutional neural networks on graphs with fast localized spectral filtering. *Advances in neural information processing systems*, 29: 3844–3852.

Dhillon, I. S.; Guan, Y.; and Kulis, B. 2007. Weighted graph cuts without eigenvectors a multilevel approach. *IEEE transactions on pattern analysis and machine intelligence*, 29(11): 1944–1957.

Farrell, S.; Calafiura, P.; Mudigonda, M.; Anderson, D.; Vlimant, J.-R.; Zheng, S.; Bendavid, J.; Spiropulu, M.; Cerati, G.; Gray, L.; et al. 2018. Novel deep learning methods for track reconstruction. *arXiv preprint arXiv:1810.06111*.

Guest, D.; Cranmer, K.; and Whiteson, D. 2018. Deep learning and its application to LHC physics. *Annual Review of Nuclear and Particle Science*, 68.

Hamilton, W. L. 2020. Graph representation learning. *Synthesis Lectures on Artifical Intelligence and Machine Learning*, 14(3): 1–159.

Kazeev, N. 2020. *Machine Learning for particle identification in the LHCb detector*. Ph.D. thesis, Yandex School of Data Analysis (RU).

Kipf, T. N.; and Welling, M. 2016. Semi-supervised classification with graph convolutional networks. *arXiv preprint arXiv:1609.02907*.

Lee, J.; Lee, I.; and Kang, J. 2019. Self-attention graph pooling. In *International Conference on Machine Learning*, 3734–3743. PMLR.

P. Skands, J. R., S. Carrazza. 2014. Tuning PYTHIA 8.1: the Monash 2013 Tune. *The European Physical Journal C*, 74(3024).

Qu, H.; and Gouskos, L. 2020. Jet tagging via particle clouds. *Physical Review D*, 101(5).

Rhee, S.; Seo, S.; and Kim, S. 2017. Hybrid approach of relation network and localized graph convolutional filtering for breast cancer subtype classification. *arXiv preprint arXiv:1711.05859*.

Sjöstrand, T.; Mrenna, S.; and Skands, P. 2008. A brief introduction to PYTHIA 8.1. *Computer Physics Communications*, 178(11): 852–867.

Skands, P. 2010. Tuning Monte Carlo generators: The Perugia tunes. *Physical Review D*, 82(7).

Vinyals, O.; Bengio, S.; and Kudlur, M. 2015. Order matters: Sequence to sequence for sets. *arXiv preprint arXiv:1511.06391*.

Wang, Y.; Sun, Y.; Liu, Z.; Sarma, S. E.; Bronstein, M. M.; and Solomon, J. M. 2019. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5): 1–12.

Ying, R.; You, J.; Morris, C.; Ren, X.; Hamilton, W. L.; and Leskovec, J. 2018. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*.

Zhang, M.; Cui, Z.; Neumann, M.; and Chen, Y. 2018. An end-to-end deep learning architecture for graph classification. In *Thirty-Second AAAI Conference on Artificial Intelligence*.