A New sPHENIX Heavy Quark Trigger Algorithm Based on Graph Neural Networks

Yimin Zhu
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
yimzhu@cs.stonybrook.edu

Tingting Xuan
Department of Applied Mathematics & Statistics
Stony Brook University
Stony Brook, NY 11794
tingting.xuan@stonybrook.edu

Giorgian Borca-Tasciuc
Department of Computer Science
Stony Brook University
Stony Brook, NY 11794
giorgian.borca-tasciuc@stonybrook.edu

Yu Sun
Sunrise Technology Inc.
Center of Excellence Wireless and Information Technology
Stony Brook, NY 11794
yu.sun@sunriseaitech.com

Abstract

Triggering plays a vital role in high energy nuclear and particle physics experiments. Here we propose a new trigger algorithm design for heavy charm quark events in proton+proton (p+p) collisions in the sPHENIX experiment at the Relativistic Heavy Ion Collider (RHIC). This trigger algorithm selects a charm event created in p+p collision by identifying the topology of a charm-hadron (D^0) decays into a pair of oppositely charged kaon and pion particles. Classical approaches are based on statistical models, relying on complex hand-designed features, and are both cost-prohibitive and inflexible for discovering charm events from a large background of other collision events. The proposed neural network based trigger algorithm takes into account unique high level features of charm events, using a stack of images that are embedded in a deep neural network. By incorporating two state-of-the-art graph neural networks, ParticleNet and SAGPool, we can learn high-level physics features and perform binary classification with simple geometrical track information. Our model attains nearly 75% accuracy and only requires moderate resources. With a small number neurons and simple input, our model is designed to be compatible for hardware acceleration and thereby enables extremely fast decision modules for real-time p+p collision events in the upcoming sPHENIX experiment at RHIC.

Fourth Workshop on Machine Learning and the Physical Sciences (NeurIPS 2021).

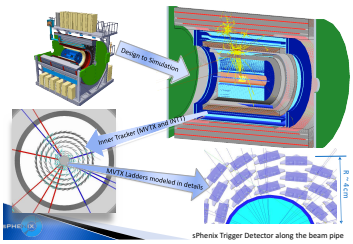

Figure 1: A simulated event display in the sPHENIX experiment. The lower left corner shows the cross-sectional view of the fast silicon detectors that is located in the center of the sPHENIX experiment and plays a vital role in controlling the quality and volume of physics event data. The lower right corner shows that sPHENIX's trigger consists of three layers of the MVTX. Each sensor stave contains two-dimensional $0.5K \times 9K$ cylindrical pixel detector plane, and there are 48 of them in total. The trigger is essentially a high-speed camera, continuously taking 250-megapixel images for p+p collisions happening at ~10MHz, and inspecting image patterns to decide whether to keep or discard each individual image.

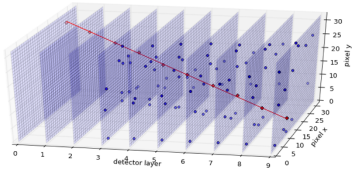

Figure 2: The pixels of hits are extremely sparse for a single p+p collision event. The meaningful connections among pixels are the consequence of particle movement across the 3D toy Detector with ten layers (channels). The red track captures an interesting particle generated from heavy ion collisions.

1 Introduction

sPHENIX trigger algorithm Design: In high-energy nuclear and particle physics experiments, a trigger is a system that uses a specific set of criteria to rapidly decide which collision events to keep when only a small fraction (typically less than 1%) of the total collision events can be recorded. Due to real-world limitations in computing power, data storage capacity, and logging rates, the trigger algorithm is essential for identifying the events of interest to record during the high-rate p+p collisions for later analysis. According to Adare et al. [2015], the Monolithic-Active-Pixel-Sensor based silicon Vertex detector system (MVTX), designed for the sPHENIX experiment at Relativistic Heavy Ion Collider (RHIC), is a precision 3D tracking device to generate a stack of 3D hit images of charged particle produced in the proton+proton (p+p) collisions. The p+p collisions will happen at a rate as high as about 10MHz, while the sPHENIX Data Acquisition System (DAQ) can only take up to a 15kHz worth of collision events. A trigger algorithm is required to reject non-interesting collision events in order to record important physics events with high efficiency within the sPHENIX DAQ bandwidth. Events that contain heavy quarks in the initial production in p+p collisions are of great interest and importance for the sPHENIX physics program. Nevertheless, the majority of such events can not be detected by the existing calorimeter-based trigger algorithms. This paper proposes to develop an intelligent Graph Neural Network (GNN) based trigger algorithm that automates the learning of event representations and features, and predicting the key physics properties of such events, i.e., particle tagging. Our existing embedded deep neural network system provides a matching solution to recognize which stack of images from the MVTX and other sPHENIX detectors have the sought pattern. In this paper, we develop a two-stage GNN-based trigger algorithm to decide whether to record the events or not, with the processed track information retrieved from the captured 3D sparse images by the sPHENIX detectors.

Extremely Fast Heavy Quark Event Trigger Design for Nuclear Physics Detectors with Deep Learning Systems A fast real-time trigger algorithm in the sPHENIX experiment needs to identify specific types of physics production events in p+p collisions.

Deep neural networks have bought tremendous advancements in the domains of data processing. These data sets typically share common characteristics, such as well-defined data structures, consistent signal to noise ratio (SNR), and regular neighborhood. On the other hand, physics experiments do not have these regularities. They often use sparse detector images in Figure 2 to minimize processing costs on blank pixels. As a result, they cannot benefit from the popular convolutional neural network (CNN) designed for natural pictures to indiscriminately treat sparse pixels as "noise" instead of signals. Recently, there is growing interest in extending deep learning approaches to graphs that model pair-wise relationships among data entities. Graph data is more difficult to handle, because each node can have an arbitrary number of neighbors. Motivated by the CNN, we will design novel GNN algorithms on raw event data to learn both local and global features and infer the local properties (tracks and jets) and global identifiers (event tagging).

The paper's contributions are that it identifies interesting physics events with great accuracy and outperforms the state-of-the-art methods that requiring sophisticated offline physics models and

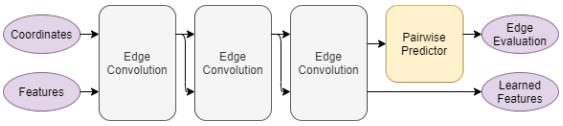

Figure 3: Cluster of Tracks

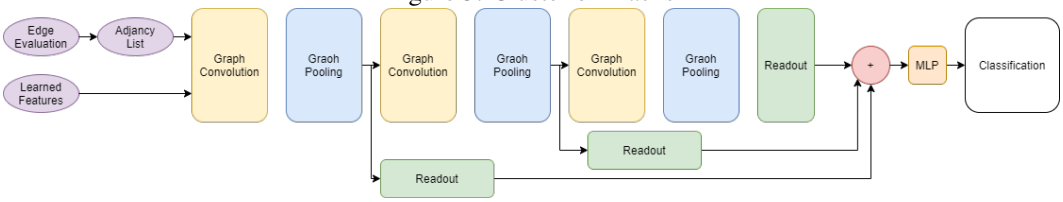

Figure 4: Trigger Prediction

reconstruction. Moreover, our model has a small number of parameters and is compatible with resource-constrained hardware accelerators. The model relies only on geometric properties that are collected by a silicon detector with extremely low latency.

2 Model Architecture

We design a two-stage model. Our first stage clusters track based on their secondary vertex. Our second stage performs trigger predictions from the clustered tracks. Instead of working with raw hits, our GNN treats each track as a graph node and learns the track configuration in an event. A track-based (as opposed to hit-based) graph affinity matrix is much smaller, and track-level physics characteristics can be directly used to tag events in the physics processing pipeline.

2.1 Cluster Tracks based on ParticleNet

A *track cluster* is a collection of tracks with the same secondary vertex. Given a collection of n tracks from the event reconstruction, we need to identify whether any pair of tracks share the same secondary vertex. Figure 3 shows the clustering system based on ParticleNet (Qu and Gouskos [2020]). That is a neural network architecture based on Dynamic Graph Convolution Neural Network Wang et al. [2019] that operates directly on particle clouds. Unlike Set2Graph by Serviansky et al. [2020] which initially treats all nodes as connected, our custom ParticleNet uses the k nearest neighbours to filter out spurious connections, focusing on those pairwise relationships with minimal distance. Our ParticleNet incorporates the track geometry information and directly learns the high-level features in the hierarchical event structure. It does not suffer from the over-connection issue in Set2Graph and shows superior performance in our experiments. ParticleNet substitutes the neighbours' feature vector in the edge convolution block and calculates the difference to its central point. We implement the edge function in the edge-convolutional block with a multilayer perceptron (MLP) and use mean aggregation. We also adopt the intertrack distance as the distance metric to select the initial k nearest neighbors. The output layer consists of a pairwise link predictor based on an MLP that takes the learned node features and predicts the connectivity probability of each pair of tracks.

Loss Function The objective of ParticleNet is to minimize the error in link prediction. Since the distance between the primary and secondary vertices are 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. Our loss function thus contains two terms: the weighted Cross Entropy loss (CELoss) and the Laplacian loss.

Our dataset is balanced for trigger/nontrigger events. To ensure balance on link prediction, we calculate CELoss on the subgraph with the same number of true and false edges. For the non-decay events, we use all the edges.

Our experimental data shows that the close distance between the primary and secondary vertices leads to a highly sensitive prediction model with an excessive number of mispredicted links. To mitigate the problem, we introduced Graph Laplacian regularization to ensure two interconnected tracks must have the same secondary vertex: $\mathcal{L}_{lap} = trace(O^\top(\hat{D} - \hat{A})O)$, where $O \in \mathbb{R}^{n\times 3}$ is the coordinate matrix of the ground truth secondary vertex of each track, $\hat{D}$ is the degree matrix, and $\hat{A}$ is the learned affinity matrix based on the predicted links by ParticleNet.

2.2 Trigger Prediction via SAGPool

We design a graph-level prediction model that classifies the input event graph into trigger and non-trigger events based on the predicted inter-track connection and track features from the previous stage.

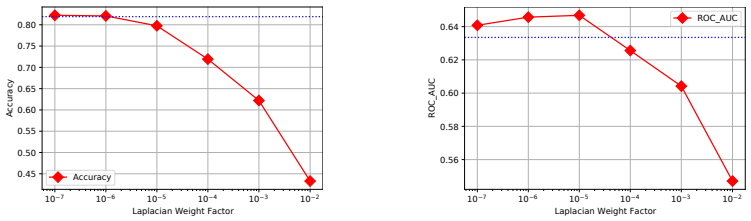

Figure 5: Model Performance with Different Laplacian Weight Factor

Table 1: Performance on Trigger Prediction from Clustered Tracks

Pooling Method	Accuracy	Roc_Auc	Training time	Epoch	Efficiency	Purity
Set2Graph+DiffPool	0.7112	0.6948	3h 06m 58s	50	0.3804	0.0380
ParticleNet+DiffPool	0.7423	0.7277	15h 31m 49s	50	0.4196	0.0419
ParticleNet+SAGPool-H	0.7492	0.8174	6h 1m 55s	50	0.4696	0.0469
ParticleNet+SAGPool-G	**0.7511**	**0.8187**	5h 12m 5s	50	**0.4739**	**0.0474**

Figure 4 shows the architecture of event-level prediction. The algorithm employs a pooling method that aggregates any number of node features into an event representation with a pre-defined size. Among multiple choices of pooling method, we adopt SAGPool proposed by Lee et al. [2019] to categorize each event due to its simplicity and efficiency. In constrast to DiffPool, proposed by Ying et al. [2018], SAGPool uses graph convolution to calculate the self-attention scores and considers both node features and graph topology with only moderate time and space costs. It also utilizes the node selection method proposed by Gao and Ji [2019] to retain a portion of the nodes of the input graph when the input sizes and structures of graphs vary.

SAGPool calculates the self-attention score Z as: $Z = \sigma(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta_{att})$ where σ is the activation function, $\tilde{A} \in R^{N\times N}$ is the adjacency matrix with self-loop connections, $\tilde{D} \in R^{N\times N}$ is the degree matrix of $\tilde{A}$, X is the input features, and Θ_{att} contains the learnable attention parameters of the pooling layer.

3 Experiments

We ran our experiments on an NVIDIA TitanXpGPU. All baselines and our model are implemented with PyTorch by Paszke et al. [2019] and PyTorch Geometric by Fey and Lenssen [2019]. We use experiments to verify our design choices, including the Laplacian regularization and SAGPool node aggregation strategy. There are 2 types of SAGPool pooling structures: SAGPool with global pooling (SAGPool-G) and SAGPool with hierarchical pooling (SAGPool-H).

Data set Our experiment data set consists of simulated sPHENIX p+p collision at 200 GeV c.m. energy with only MVTX detector readout provided by Huang [2018]. The three-layer structure with interleaved detector strips generates short tracks. The input vector for each track consists of the coordinates of the three hits in each detector layer. The length of the edges, the angle between these two edges, the total track length, and the coordinate of the geometric center of all the hits in the graph are calculated as complementary features to ease the downstream learning task.

Laplacian Loss Factor Figure 5 shows that the performance improves with properly tuned weight for the Laplacian loss. The blue dotted horizontal line represents the performance without Laplacian loss. When the Laplacian weight factor is $\leq$ 1e-6, both roc_auc score and accuracy outperform the model trained without the Laplacian loss. We choose the Laplacian weight factor 1e-6 for the downstream task.

Comparison between SAGPool and DiffPool We perform a grid search to find the best pooling ratio (0.25, 0.5, 0.75) and the best dropout ratio (0.5, 0.7). We only show the best setting for each model in Table 1. ParticleNet shows 4% better accuracy compared to Set2Graph proposed by Shlomi et al. [2021]. SAGPool is efficient, converging three times faster than Diffpool, and reaches a peak performance 0.9% better than that of Diffpool. SAGPool with global structure using pooling ratio 0.75 and dropout ratio 0.7 preserves 47% triggers while rejecting 90% events. It has the highest roc_auc score which indicates that it the preferred model to distinguish between triggering and non-trigger events.

4 Limitations

Currently, the node features generated from the first stage are only optimized for link prediction and not for trigger prediction. One alternative solution would be to train these two models in tandem, such that the generated node features are optimized for both link prediction and triggering. We would then need to perform a grid search to find the ratio to weight the loss of the first stage and the loss of the second stage. This grid search on the pipeline trained end-to-end would search for the best trade-off between between trigger accuracy and inter-track clustering performance. Another solution is to perform transductive learning , as proposed by Vapnik [1999]. Moreover, Tingting et al. [2021] provides a method to construct the tracks dataset from raw hits. Our current model can be applied on the generated dataset to make a three-stage model and implement the complete end-to-end solution that inputs raw hits and outputs trigger tags.

5 Conclusion

This paper propose a two-stage model to classify the networks of particle tracks into trigger and non-trigger events. Our model starts with the clouds of the input tracks and cluster the tracks based on their vicinity. Afterwards, it performs the binary classification based on the learned features. Our model can predict triggers with higher accuracy and achieve a significant speedup in training time with small number of model parameters and simple input of geometric events information. It paves the way for a high-speed trigger solution with hardware acceleration in the future.

6 Acknowledgement

This material is based upon work supported by the U.S. Department of Energy, Office of Science, Office of Advanced Scientific Computing Research, Office of Nuclear Physics, under Award Number DE-SC0019518.

Checklist

1. For all authors...
   (a) Do the main claims made in the abstract and introduction accurately reflect the paper's contributions and scope? [Yes]
   (b) Did you describe the limitations of your work? [Yes]
   (c) Did you discuss any potential negative societal impacts of your work? [No]
   (d) Have you read the ethics review guidelines and ensured that your paper conforms to them? [Yes]
2. If you are including theoretical results... [No]
   (a) Did you state the full set of assumptions of all theoretical results?
   (b) Did you include complete proofs of all theoretical results?
3. If you ran experiments...
   (a) Did you include the code, data, and instructions needed to reproduce the main experimental results (either in the supplemental material or as a URL)? [No]
   (b) Did you specify all the training details (e.g., data splits, hyper parameters, how they were chosen)? [Yes]
   (c) Did you report error bars (e.g., with respect to the random seed after running experiments multiple times)? [No]
   (d) Did you include the total amount of compute and the type of resources used (e.g., type of GPUs, internal cluster, or cloud provider)? [Yes]
4. If you are using existing assets (e.g., code, data, models) or curating/releasing new assets...
   (a) If your work uses existing assets, did you cite the creators? [Yes]
   (b) Did you mention the license of the assets? [Yes]
   (c) Did you include any new assets either in the supplemental material or as a URL? [No]
   (d) Did you discuss whether and how consent was obtained from people whose data you're using/curating? [Yes]
   (e) Did you discuss whether the data you are using/curating contains personally identifiable information or offensive content? [No]
5. If you used crowdsourcing or conducted research with human subjects... [No]

(a) Did you include the full text of instructions given to participants and screenshots, if applicable?

(b) Did you describe any potential participant risks, with links to Institutional Review Board (IRB) approvals, if applicable?

(c) Did you include the estimated hourly wage paid to participants and the total amount spent on participant compensation?

References

A Adare, S Afanasiev, C Aidala, NN Ajitanand, Y Akiba, R Akimoto, J Alexander, K Aoki, N Apadula, H Asano, et al. An upgrade proposal from the phenix collaboration. *arXiv preprint arXiv:1501.06197*, 2015.

Huilin Qu and Loukas Gouskos. Jet tagging via particle clouds. *Physical Review D*, 101(5), Mar 2020. ISSN 2470-0029. doi: 10.1103/physrevd.101.056019. URL http://dx.doi.org/10.1103/PhysRevD.101.056019.

Yue Wang, Yongbin Sun, Ziwei Liu, Sanjay E Sarma, Michael M Bronstein, and Justin M Solomon. Dynamic graph cnn for learning on point clouds. *Acm Transactions On Graphics (tog)*, 38(5):1–12, 2019.

Hadar Serviansky, Nimrod Segol, Jonathan Shlomi, Kyle Cranmer, Eilam Gross, Haggai Maron, and Yaron Lipman. Set2graph: Learning graphs from sets. *Advances in Neural Information Processing Systems*, 33, 2020.

Junhyun Lee, Inyeop Lee, and Jaewoo Kang. Self-attention graph pooling. In *International Conference on Machine Learning*, pages 3734–3743. PMLR, 2019.

Rex Ying, Jiaxuan You, Christopher Morris, Xiang Ren, William L Hamilton, and Jure Leskovec. Hierarchical graph representation learning with differentiable pooling. *arXiv preprint arXiv:1806.08804*, 2018.

Hongyang Gao and Shuiwang Ji. Graph u-nets. In *international conference on machine learning*, pages 2083–2092. PMLR, 2019.

Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, Alban Desmaison, Andreas Kopf, Edward Yang, Zachary DeVito, Martin Raison, Alykhan Tejani, Sasank Chilamkurthy, Benoit Steiner, Lu Fang, Junjie Bai, and Soumith Chintala. Pytorch: An imperative style, high-performance deep learning library. In H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox, and R. Garnett, editors, *Advances in Neural Information Processing Systems 32*, pages 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf.

Matthias Fey and Jan E. Lenssen. Fast graph representation learning with PyTorch Geometric. In *ICLR Workshop on Representation Learning on Graphs and Manifolds*, 2019.

Jin Huang. sphenix machine learning open data set for tracking and heavy flavor physics. https://github.com/sPHENIX-Collaboration/HFMLTrigger, 2018.

Jonathan Shlomi, Sanmay Ganguly, Eilam Gross, Kyle Cranmer, Yaron Lipman, Hadar Serviansky, Haggai Maron, and Nimrod Segol. Secondary vertex finding in jets with neural networks. *The European Physical Journal C*, 81(6), Jun 2021. ISSN 1434-6052. doi: 10.1140/epjc/s10052-021-09342-y. URL http://dx.doi.org/10.1140/epjc/s10052-021-09342-y.

Vladimir N Vapnik. An overview of statistical learning theory. *IEEE transactions on neural networks*, 10(5):988–999, 1999.

Xuan Tingting, Zhu Yimin, Durao Fedrik, and Sun Yu. End-to-end online sphenix trigger detection-pipeline. *Machine Learning and the Physical Sciences Workshop at the 35th Conference on Neural Information Processing Systems*, 2021.