

Reconstructing Particle Tracks in One Go with a Recursive Graph Attention Network

Jay Chan

Scientific Data Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA

E-mail: jaychan@lbl.gov

Abstract. Track reconstruction is a crucial task in particle experiments and is traditionally very computationally expensive due to its combinatorial nature. Recently, graph neural networks (GNNs) have emerged as a promising approach that can improve scalability. Most of these GNN-based methods, including the edge classification (EC) and the object condensation (OC) approach, require an input graph that needs to be constructed beforehand. In this work, we consider a one-shot OC approach that reconstructs particle tracks directly from a set of hits (point cloud) by recursively applying graph attention networks with an evolving graph structure. This approach iteratively updates the graphs and can better facilitate the message passing across each graph. Preliminary studies on the TrackML dataset show physics and computing performance comparable to current production algorithms for track reconstruction.

1 Introduction

Track reconstruction is an essential task in particle experiments. The traditional tracking algorithm, the Combinatorial Kalman Filter [1, 2, 3], is computationally expensive and does not scale well with the event size. Recent developments have focused on applying graph neural networks (GNNs) to the tracking procedure [4, 5, 6, 7] and demonstrated that the GNN-based tracking algorithms can achieve linear scaling with the event size [4, 8].

These GNN-based tracking algorithms can be classified into two approaches: edge classification (EC) [4, 5, 6] and object condensation (OC) [7]. In the EC approach, the GNN is trained to remove edges (based on the learned edge scores) that connect hits belonging to different particles. This will then be followed by a graph segmentation algorithm, which assigns each connected component as a track candidate. In the OC approach, the goal of the GNN is to learn the node representation in a latent space, where hits belonging to the same particles are attracted to one another. One can then perform a clustering for the hits in the learned latent space and assign each cluster as a track candidate.

While effective, most of these methods require a graph construction step before performing GNN. Since it is unrealistic to construct a fully connected graph ($N_{\text{edges}} \sim N_{\text{nodes}}^2$), multiple methods, such as metric learning [4] and module map [5] have been developed to construct a graph connecting the hits that are likely to belong to the same particles. In metric learning, embedding for each node is learned by a multi-layer perceptron (MLP). Edges are then constructed, connecting the nearest neighbors in the embedding space. Here, the embedding for each node is purely based on the features of that node and does not depend on how the node interacts with others. Due to the lack of awareness of other nodes, the metric-learning-based graph construction tends to produce a graph that contains many fake edges (low edge purity). One often needs to use another network that takes edge features as inputs and

further reduces the graph size by removing edges [4]. In the module map approach, one constructs the graph based on all possible connections between detector modules. This is currently resource-intensive and also tends to produce low-purity graphs.

It is important to note that graph construction efficiency plays a crucial role in the performance of later stages for both EC and OC approaches. In the EC approach, any graph construction inefficiency (the missing true edges in the constructed graph) is directly propagated to the graph segmentation stage. In the OC approach, it is possible to recover missing true edges after the clustering. However, graph construction inefficiency still leads to information loss during the GNN message passing step. Furthermore, low purity (due to fake edges) results in noisy information during the message passing. Both cases affect the effectiveness of the message passing step.

In this work, we propose an OC-based method that embeds the graph construction into a Recursive Graph Attention (RGAT) architecture, which takes point clouds as inputs and iteratively constructs the graphs based on the updated embedding. The message passing in the RGAT is then based on the updated graph in each iteration. This method allows to gradually enhance the constructed graph efficiency in each iteration and thus can improve the effectiveness of the message passing. We test the method with the TrackML dataset [9, 10] and obtain promising physics and computing performance comparable to the existing track reconstruction algorithms.

2 Method

The proposed end-to-end tracking pipeline consists of a Recursive Graph Attention (RGAT) step and a clustering step. The RGAT takes a set of hits (a point cloud) as inputs and outputs the node embedding for each node. We adopt a similar approach to GRAVNET [11], where the graph attention acts on dynamically built KNN graphs. In comparison with Ref. [11, 12], our attention weight is learned from a dedicated edge network, and the aggregation is based on the learned edge representation rather than the neighboring node representation. The nodes are then clustered in the node embedding space and assigned to different track candidates. In our study, we adopt Density-Based Spatial Clustering of Applications with Noise (DBSCAN) [13] as the clustering algorithm.

2.1 RGAT

The RGAT architecture is illustrated in Fig. 1. The d -dimensional input node features $\nu \in \mathbb{R}^d$ are first encoded:

$$x \in \mathbb{R}^{d_x} = f_{\text{ENC}}(\nu), \quad (1)$$

where f_{ENC} is the MLP node encoder. The encoded node features are then taken as inputs by a number of iterations where we first update the graph node embeddings and then the edges by connecting the K-Nearest Neighbors (KNN) in the node embedding space. In the first iteration ($i = 0$), each node is projected into a latent space (denoted as h_0 -space) by an MLP node network:

$$h_0 \in \mathbb{R}^{d_h} = f_0(x). \quad (2)$$

A MLP node decoder f_{DEC} then acts on h_0 and outputs a node embedding in p_0 -space:

$$p_0 \in \mathbb{R}^{d_p} = f_{\text{DEC}}(h_0). \quad (3)$$

This is then followed by a KNN in p_0 which constructs the first graph G_0 .

In each of the next iterations ($i \geq 1$), multiple graph-attention-based [12] message passing steps are performed to obtain the updated node representation $h_i \in \mathbb{R}^{d_h}$. The same node decoder f_{DEC} is then used to obtain the updated node embedding in p_i -space:

$$p_i \in \mathbb{R}^{d_p} = f_{\text{DEC}}(h_i), \quad i \geq 1, \quad (4)$$

which is followed by a KNN that updates the graph $p_i \rightarrow G_i$.

In each message passing step, we first learn an edge representation ($e_{ij} \in \mathbb{R}^{d_e}$, where i denotes the current RGAT iteration and j denotes the current message passing step), and an edge weight $w_{ij} \in \mathbb{R}$ from the two connecting nodes for each edge:

$$e_{ij} = \begin{cases} f_e^j(x^S, x^T), & i \geq 1, j = 0 \\ f_e^j(h_{ij}^S, h_{ij}^T, e_{i(j-1)}), & i \geq 1, j \geq 1 \end{cases}, \quad (5)$$

and

$$w_{ij} = \begin{cases} f_w^j(x^S, x^T), & i \geq 1, j = 0 \\ f_w^j(h_{ij}^S, h_{ij}^T, e_{i(j-1)}), & i \geq 1, j \geq 1 \end{cases}, \quad (6)$$

where S and T denote the source and target nodes of an edge, respectively. h_{ij} is the current representation of a given node. Note that for $j \geq 1$, in addition to the two connecting nodes, the edge networks f_e^j and f_w^j also take the previous edge representation $e_{i(j-1)}$ as inputs.

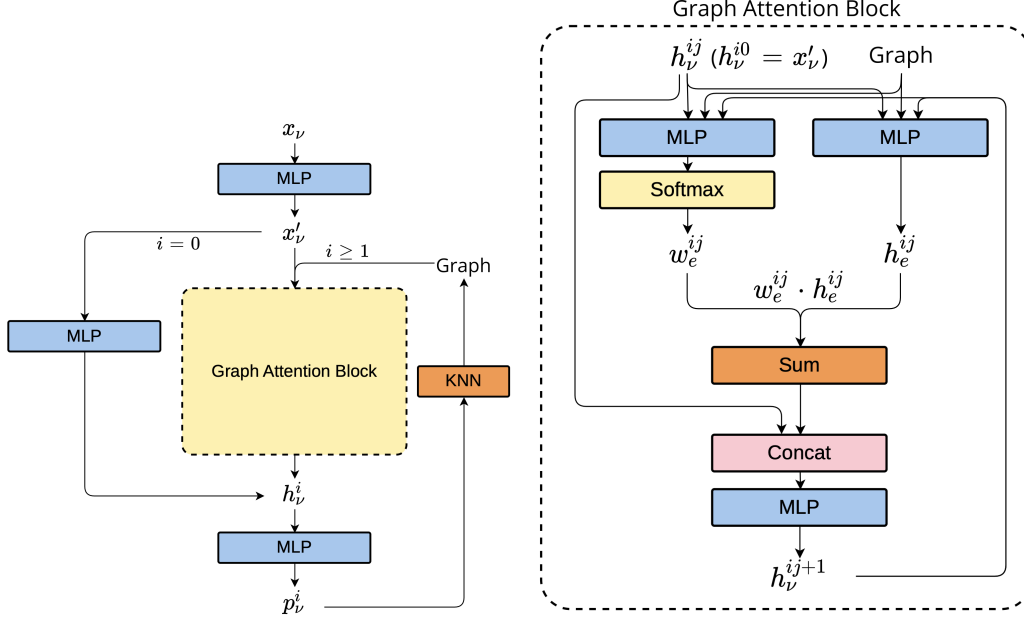


Figure 1: The RGAT architecture. i corresponds to each RGAT iteration, and j corresponds to each message passing step. An RGAT iteration generally consists of a graph attention block and a KNN. The first iteration does not perform graph attention and the last iteration does not perform KNN.

The node representation h_{ij} is updated by a node network f_v^j in each message passing step, which takes as the inputs the weighting sum of all connecting edges for each node:

$$a_{ij} = \sum_{\epsilon \in E} \text{softmax}(w_{ij}^\epsilon) e_{ij}^\epsilon, \quad (7)$$

$$h_{ij} = \begin{cases} f_v^j(x, a_{i0}), & i \geq 1, j = 0 \\ f_v^j(h_{i(j-1)}, a_{ij}), & i \geq 1, j \geq 1 \end{cases}, \quad (8)$$

where E corresponds to all connecting edges for a given node. The h_{ij} from the last message passing step ($h_{i(-1)}$) is then taken as h_i that enters Eq. 4.

2.2 Loss Function

The RGAT model is optimized to minimize the loss L :

$$L = \mathbb{E}_{\epsilon \in \text{true edges}}(l_\epsilon) + \mathbb{E}_{\epsilon \in \text{random edges}}(l_\epsilon) + \mathbb{E}_{\epsilon \in \text{KNN edges}}(l_\epsilon), \quad (9)$$

where l_ϵ is the individual loss term for a given representative edge connection:

$$l_\epsilon = y_\epsilon d_\epsilon^2 + (1 - y_\epsilon) \max^2(0, m - d_\epsilon), \quad (10)$$

where the first term is an attractive loss that brings together nodes belonging to the same particles, and the latter is a repulsive hinge loss that separates nodes belonging to different particles. y_ϵ is the truth

label for each edge, which is 1 for an edge connecting hits coming from the same particle (true edge), and 0 otherwise (fake edge). d_ϵ is the Euclidean distance between two nodes in the p_j -space in the last RGAT iteration (p_{-1}), and m is a constant value, which is set to 1 in our study. The representative edges are pooled from all true edges, randomly selected edges and all KNN edges, corresponding to the three terms in Eq. 9. Since the majority of randomly selected edges are fake edges, the combination of true and randomly selected edges ensure that we optimize the model based on both true and fake edges. KNN edges represent the most true-like edges (edges connecting the nodes that are the nearest). Sampling from these edges allow us to emphasize the fake edges that look like true edges.

2.3 DBSCAN

p_{-1} is taken as inputs of DBSCAN, which then outputs a track label λ for each node:

$$\{\lambda_\nu\} = \text{DBSCAN}(\{p_{-1}\}) | \nu \in V, \quad (11)$$

where V represents all nodes in a graph, i.e., all hits in a collision event. All meaningful track labels are positive integers, and all nodes that share the same meaningful track label form a track candidate. For DBSCAN, it is possible that a node does not belong to any cluster. In this case, the λ is assigned as 0 and the node is considered as a noise hit. Two essential parameters for DBSCAN are the radius ϵ and the minimum number of nodes `MIN_SAMPLES`. Note that DBSCAN is only run for the inference and it not used in the training.

2.4 Models and Implementation

All MLPs in the RGAT model consist of 3 hidden layers, each with a width of 128 neurons. Each intermediate layer in these networks uses a SiLU [14] activation function and batch normalization. The last layer of f_{DEC} uses a Tanh activation function, and the output is further normalized $p' = \frac{p}{|p|}$. For the following studies, we set d_x , d_h and d_e to 128, while d_p is set to 24. The RGAT model consists of 5 RGAT iterations ($i \leq 4$), and each GAT block consists of 8 message passing steps ($j \leq 7$). We consider $k = 10$ for all KNN steps in the training.

All neural networks are implemented using PyTorch [15] and PyTorch Lightning [16] in the ACORN framework [17] and simultaneously optimized with Adam [18] with a learning rate of 2×10^{-4} . The training uses a batch size of 1 event and is performed for 200 epochs.

DBSCAN is implemented using Rapids cuML [19].

3 Experimental Setup and Evaluation

In the following, we test our tracking pipeline with the TrackML dataset, which is a simulation of a generic tracking detector under the HL-LHC pileup condition. For simplicity, we consider only particles with transverse momentum $p_T > 1$ GeV. This means hits coming from particles with $p_T < 1$ GeV and noise hits are removed from our simulated datasets, reducing the number of hits by a factor of $\mathcal{O}(10)$.

We train the RGAT model with 12 input node features $\nu \in \mathbb{R}^{12}$ (described in Ref. [4]). We first evaluate the RGAT edge-wise performance on the KNN graphs built in the p_{-1} -space. We define two metrics for the KNN edge-wise performance, including ‘‘edge-wise efficiency’’ Eff_{KNN} and ‘‘edge-wise purity’’¹ Pur_{KNN} :

$$\text{Eff}_{\text{KNN}} = \frac{N_{\text{KNN}}^{y=1}}{N^{y=1}}, \quad (12)$$

$$\text{Pur}_{\text{KNN}} = \frac{N_{\text{KNN}}^{y=1}}{N_{\text{KNN}}} = \frac{N_{\text{KNN}}^{y=1}}{N_{\text{hits}} \cdot k}, \quad (13)$$

where $N^{y=1}$ is the number of all true edges, N_{KNN} is the number of edges counted from all KNN graphs, which is equal to $N_{\text{hits}} \cdot k$, and $N_{\text{KNN}}^{y=1}$ is the number of true KNN edges. The RGAT performance is the best when Eff_{KNN} and Pur_{KNN} are closest to 1 (they cannot be larger than 1). Note that $N_{\text{KNN}}^{y=1}$ cannot exceed k per node or the number of true edges for each source node:

¹ ‘‘Efficiency’’ and ‘‘purity’’ are also commonly known as ‘‘recall’’ and ‘‘precision’’.

$$N_{\text{KNN}}^{y=1} \leq \sum_{\nu^S} \min(k, N_{\nu^S}^{y=1}). \quad (14)$$

This thus determines the upper bounds of Eff_{KNN} and Pur_{KNN} . As shown in Fig. 2, we plot Eff_{KNN} and Pur_{KNN} as a function of k . Eff_{KNN} increases and approaches 1 as k increases, while Pur_{KNN} decreases as k increases. Upper bounds of Eff_{KNN} and Pur_{KNN} are also shown in the figure. We observe that both Eff_{KNN} and Pur_{KNN} are very close to the upper bounds for any k , which indicates that the RGAT performance is close to optimal. Note that the KNN graphs are only used as the inputs for the message passing and training loss, and they do not correspond to the final track candidates which are obtained using DBSCAN. Thus, the performance on the KNN graphs does not directly translate to the tracking performance. However, it allows us to evaluate the RGAT model performance with effects decoupled from the later stage of DBSCAN.

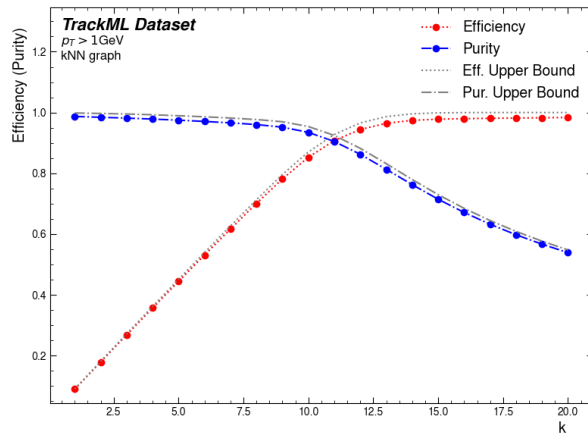


Figure 2: Edge-wise efficiency and purity in the KNN graphs as a function of k . The upper bound for each metric is also shown in the plot.

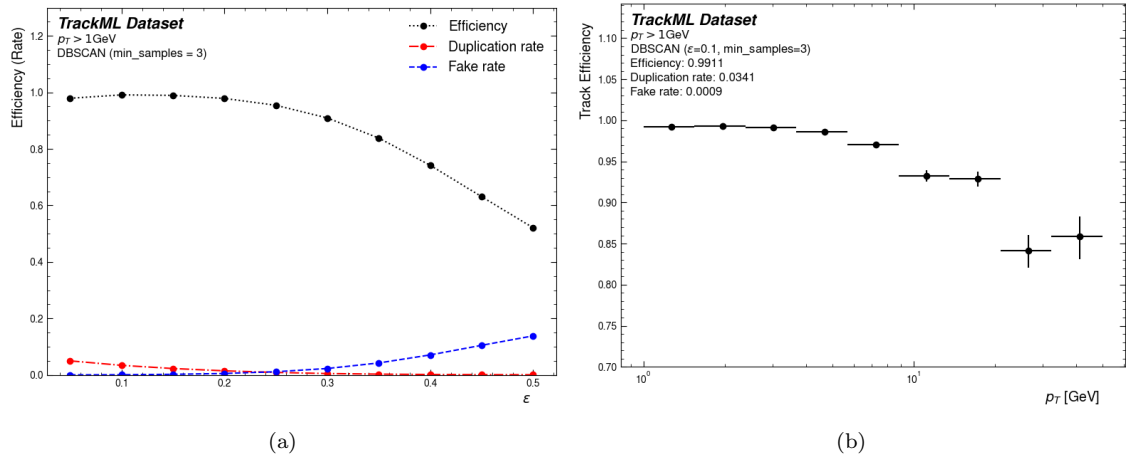


Figure 3: (a) Track efficiency, duplication rate and fake rate with track candidates obtained from the DBSCAN clusters as a function of ϵ . (b) Track efficiency at $\epsilon = 0.1$ as a function of particle p_T .

METHOD	EFFICIENCY	DUP. RATE	FAKE RATE
EC	0.9898	0.0421	0.0012
OC	0.9902	0.0328	0.0015
RGAT	0.9911	0.0341	0.0009

Table 1: Track performance with RGAT compared with methods that require pre-constructed input graphs, including the edge classification approach (EC) and the object condensation approach (OC). $\epsilon = 0.1$ is used for the DBSCAN in all scenarios.

The track performance is then evaluated on all track candidates extracted from the DBSCAN clusters. A track candidate is matched to a particle if more than 50% of the track hits come from that particle. Three metrics are then defined as follows:

- Efficiency: The fraction of particles to which at least 1 track candidate is matched.
- Duplication rate: The fraction of track candidates that are matched to the same particles.
- Fake rate: The fraction of track candidates that are not matched to any particle.

Fig. 3a shows the three metrics as a function of ϵ , which is a parameter of DBSCAN. In particular, when $\epsilon = 0.1$, we obtain a track efficiency of 0.9911, duplication rate of 0.0341 and fake rate of 0.0009. As shown in Fig. 3b, we also plot the track efficiency across different particle p_T bins. We observe a higher track efficiency in lower p_T , and lower efficiency in higher p_T , which is likely due to imbalanced training data statistics across p_T (i.e. there are fewer particles with higher p_T). Increasing the training statistics or training with artificial weights could improve performance in high p_T region, which we leave for future work.

For comparison, we also perform track reconstruction with both the EC and OC approaches where a pre-constructed graph is required as an input for the graph attention network. For these pre-constructed graph-based approaches, we adopt the metric learning method [4] for graph construction. We simply take the same architecture as in the first RGAT iteration ($i = 0$), which is an MLP, as the neural network used in metric learning and perform KNN with $k = 10$. The graph attention network has the same architecture as the graph attention block in the RGAT model for both EC and OC approaches. A comparison of RGAT result with methods that require a pre-constructed input graph is shown in Tab. 1. We observe that RGAT outperforms the pre-constructed graph-based methods.

Finally, Fig. 4 shows the training and inference time when running on an NVIDIA A100 Graphical Processing Unit (GPU) versus the total number of hits for each event. The averaged total computing time per event is around 0.33 seconds for training and 0.28 seconds for inference. The total computing time is also broken down into different components, including Graph Attention, KNN, calculation of loss, backward propagation and DBSCAN. The majority of the computing time comes from graph attention and KNN.

4 Conclusions

In this paper, we proposed a new object condensation-based approach to particle track reconstruction. The approach takes point clouds as inputs and builds graphs on the fly for each recursive graph attention iteration. The constructed graphs are iteratively updated, improving message-passing efficacy. We demonstrated this approach with the TrackML dataset and obtained excellent performance in both KNN graphs and reconstructed track candidates. The result is encouraging and requires future work to further improve both the computational and physics performance. For example, we observed lower track efficiency for particles with higher p_T , which could be improved by increasing training statistics of higher p_T particles. In addition, KNN has a significant contribution to the computing time. It will be important to reduce the time consumption with an alternative method, such as approximate nearest neighbors and radius nearest neighbors.

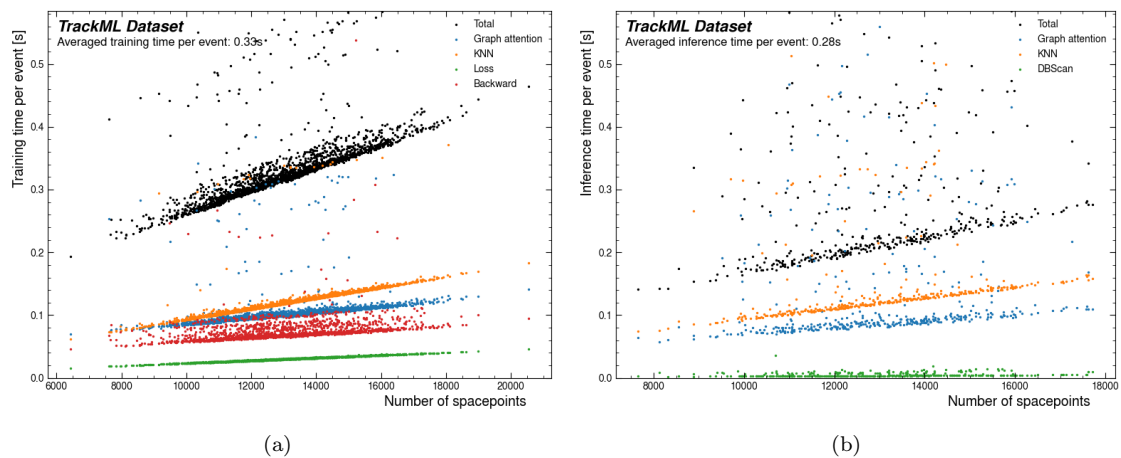


Figure 4: Training (a) and inference (b) time for each event versus the number of hits. The computing time is also broken down different components, including Graph Attention, KNN, calculation of loss, backward propagation and DBSCAN.

Acknowledgements

This work was supported by the DOE HEP Center for Computational Excellence Lawrence Berkeley National Laboratory under B&R KA2401045.

References

1. Strandlie A and Frühwirth R. Track and vertex reconstruction: From classical to adaptive methods. *Rev. Mod. Phys.* 2010 May; 82(2):1419–58. DOI: 10.1103/RevModPhys.82.1419. Available from: <https://link.aps.org/doi/10.1103/RevModPhys.82.1419>
2. Aaboud M et al. Performance of the ATLAS Track Reconstruction Algorithms in Dense Environments in LHC Run 2. *Eur. Phys. J. C* 2017; 77:673. DOI: 10.1140/epjc/s10052-017-5225-7. arXiv: 1704.07983 [hep-ex]
3. Chatrchyan S et al. Description and performance of track and primary-vertex reconstruction with the CMS tracker. *JINST* 2014; 9:P10009. DOI: 10.1088/1748-0221/9/10/P10009. arXiv: 1405.6569 [physics.ins-det]
4. Ju X et al. Performance of a geometric deep learning pipeline for HL-LHC particle tracking. *Eur. Phys. J. C* 2021; 81:876. DOI: 10.1140/epjc/s10052-021-09675-8. arXiv: 2103.06995 [physics.data-an]
5. Biscarat C, Caillou S, Rougier C, Stark J, and Zahreddine J. Towards a realistic track reconstruction algorithm based on graph neural networks for the HL-LHC. *EPJ Web Conf.* 2021; 251:03047. DOI: 10.1051/epjconf/202125103047. arXiv: 2103.00916 [physics.ins-det]
6. Caillou S, Calafiura P, Farrell SA, Ju X, Murnane DT, Rougier C, Stark J, and Vallier A. ATLAS ITk Track Reconstruction with a GNN-based pipeline. Tech. rep. Geneva: CERN, 2022. Available from: <https://cds.cern.ch/record/2815578>
7. Lieret K, DeZoort G, Chatterjee D, Park J, Miao S, and Li P. High Pileup Particle Tracking with Object Condensation. 2023 Dec. arXiv: 2312.03823 [physics.data-an]
8. Lazar A et al. Accelerating the Inference of the Exa.TrkX Pipeline. *J. Phys. Conf. Ser.* 2023; 2438:012008. DOI: 10.1088/1742-6596/2438/1/012008. arXiv: 2202.06929 [physics.ins-det]
9. Amrouche S et al. The Tracking Machine Learning challenge : Accuracy phase. *The NeurIPS '18 Competition: From Machine Learning to Intelligent Conversations.* 2019 Apr. DOI: 10.1007/978-3-030-29135-8_9. arXiv: 1904.06778 [hep-ex]

10. Amrouche S et al. The Tracking Machine Learning Challenge: Throughput Phase. *Comput. Softw. Big Sci.* 2023; 7:1. DOI: 10.1007/s41781-023-00094-w. arXiv: 2105.01160 [cs.LG]
11. Qasim SR, Kieseler J, Iiyama Y, and Pierini M. Learning representations of irregular particle-detector geometry with distance-weighted graph networks. *Eur. Phys. J. C* 2019; 79:608. DOI: 10.1140/epjc/s10052-019-7113-9. arXiv: 1902.07987 [physics.data-an]
12. Veličković P, Cucurull G, Casanova A, Romero A, Liò P, and Bengio Y. Graph Attention Networks. *International Conference on Learning Representations*. 2018. Available from: <https://openreview.net/forum?id=rJXMpikCZ>
13. Ester M, Kriegel HP, Sander J, and Xu X. A density-based algorithm for discovering clusters in large spatial databases with noise. *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*. KDD'96. Portland, Oregon: AAAI Press, 1996 :226–31
14. Elfving S, Uchibe E, and Doya K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks* 2018; 107. Special issue on deep reinforcement learning:3–11. DOI: <https://doi.org/10.1016/j.neunet.2017.12.012>. Available from: <https://www.sciencedirect.com/science/article/pii/S0893608017302976>
15. Paszke A, Gross S, Massa F, Lerer A, Bradbury J, Chanan G, Killeen T, Lin Z, Gimelshein N, Antiga L, Desmaison A, Kopf A, Yang E, DeVito Z, Raison M, Tejani A, Chilamkurthy S, Steiner B, Fang L, Bai J, and Chintala S. PyTorch: An Imperative Style, High-Performance Deep Learning Library. *Advances in Neural Information Processing Systems 32*. Ed. by Wallach H, Larochelle H, Beygelzimer A, d'Alché-Buc F, Fox E, and Garnett R. Curran Associates, Inc., 2019 :8024–35. Available from: <http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf>
16. Falcon W and The PyTorch Lightning team. PyTorch Lightning. Version 1.4. 2019 Mar. DOI: 10.5281/zenodo.3828935. Available from: <https://github.com/Lightning-AI/lightning>
17. Atkinson MJ, Caillou S, Clafiura P, Collard C, Farrell SA, Huth B, Ju X, Liu R, Minh Pham T, Murnane D(a, Neubauer M, Rougier C, Stark J, Torres H, and Vallier A. gnn4itk. Available from: <https://github.com/GNN4ITkTeam/CommonFramework>
18. Kingma D and Ba J. Adam: A method for stochastic optimization. 2014. arXiv: 1412.6980 [cs]
19. Team RD. RAPIDS: Libraries for End to End GPU Data Science. 2023. Available from: <https://rapids.ai>