EggNet: An Evolving Graph-based Graph Attention Network for Particle Track Reconstruction

Paolo Calafiura [1] **Jay Chan** [1] **Loic Delabrouille** [2 1] **Brandon Wang** [3 1]

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 better track performance compared to the methods that require a fixed input graph.

1. Introduction

Track reconstruction is an essential task in particle experiments. For each particle collision event[1], the goal of track reconstruction is to associate a list of 2D or 3D position measurements from a tracking detectors (referred to as "hits") to a list of particle track candidates. A track candidate is formed by a list of hits that are determined by the tracking algorithm. Ideally, the hits in each track candidate should originate from the same particle, and the tracking algorithm should identify all of the particles emerging from the collisions (i.e. 100% efficiency).

The traditional tracking algorithm, the Combinatorial Kalman Filter (Strandlie & Frühwirth, 2010; Aaboud et al., 2017; Chatrchyan et al., 2014), starts with a track seed of two (doublet) or three (triplet) hits, each of which provides an initial track direction. The next possible hits are then added by iteratively looking for the hits that match the extrapolated trajectory. Despite robust track performance, this method is computationally expensive and the computational cost does not scale well with the number of hits (referred to as "event size"). Performing such tracking algorithm thus becomes much more challenging in the future experiments (such as HL-LHC), where the event size is significantly increased.

In order to address the computational challenge of the traditional algorithm, recent developments have focused on applying graph neural networks (GNNs) to the tracking procedure (Ju et al., 2021; Biscarat et al., 2021; Caillou et al., 2022; Lieret et al., 2023) and demonstrated that the GNN-based tracking algorithms can achieve linear scaling with the event size (Ju et al., 2021; Lazar et al., 2023). These GNN-based tracking algorithms can be classified into two approaches: edge classification (EC) (Ju et al., 2021; Biscarat et al., 2021; Caillou et al., 2022) and object condensation (OC) (Lieret et al., 2023). 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 ($N_{\text{edges}} \sim N^2_{\text{nodes}}$), multiple methods, such as metric learning (Ju et al., 2021) and module map (Biscarat et al., 2021) have been developed to construct a graph connecting the hits that are likely to belong to the same particles. In metric learning, embedding

[*]Equal contribution [1]Scientific Data Division, Lawrence Berkeley National Laboratory, Berkeley, CA 94720, USA [2]Department of Computer Science, École normale supérieure de Rennes, 35170 Bruz, France [3]Department of Computer Science, University of California, Berkeley, CA 94720, USA. Correspondence to: Jay Chan <jaychan@lbl.gov>.

Proceedings of the 41st International Conference on Machine Learning, Vienna, Austria. PMLR 235, 2024. Copyright 2024 by the author(s).

[1]A particle collision event is typically a bunch crossing of two particle beams with opposite directions, resulting in numerous particles emerging from the collision point.

for each node is learned by a multi-layer perception (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 (Ju et al., 2021). 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 an Evolving Graph-based Graph Attention Network (EggNet) architecture, which takes point clouds as inputs and iteratively constructs the graphs based on the updated embedding. The message passing in the EggNet 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[2] (Amrouche et al., 2019; 2023) and obtain better track performance compared to the existing GNN-based methods that require a pre-constructed input graph.

2. Method

The proposed end-to-end tracking pipeline consists of an EggNet step and a clustering step. As shown in Figure 1, the EggNet 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 (Qasim et al., 2019), where the graph attention acts on dynamically built KNN graphs. In comparison with (Qasim et al., 2019; Veličković et al., 2018), 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) (Ester et al., 1996) as the clustering algorithm.

[2]TrackML dataset is a dataset commonly used for benchmarking the performance of a tracking algorithm. It contains collision events simulated with a general particle detector. The event size in the TrackML dataset is comparable to HL-LHC.

2.1. EggNet

The EggNet architecture is illustrated in Figure 2. The d-dimensional input node features $\nu \in \mathbb{R}^d$ are first encoded:

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

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:

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

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

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

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 (Veličković et al., 2018) 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:

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

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 EggNet 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:

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

and

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

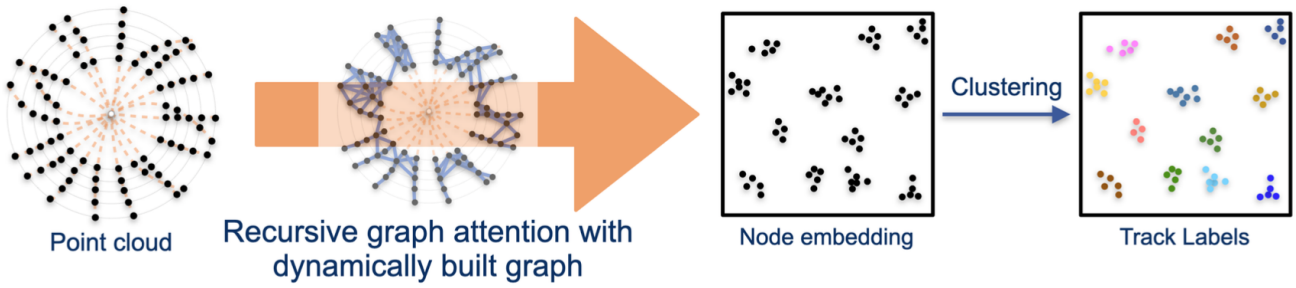

Figure 1. The proposed end-to-end tracking pipeline. Starting with a point cloud, where each point corresponds to a hit, we run an EggNet model that iteratively constructs a graph and performs message passing. EggNet outputs the embedding for each node, and the track candidates are extracted from the DBSCAN clusters that are obtained in the node embedding space.

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_ϵ^j and f_w^j also take the previous edge representation $e_{i(j-1)}$ as inputs.

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

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

$$h_{ij} = \begin{cases} f_\nu^j\left(x,\ a_{i0}\right), & i \geq 1,\ j = 0 \\ f_\nu^j\left(h_{i(j-1)},\ a_{ij}\right), & i \geq 1,\ j \geq 1 \end{cases}, \tag{8}$$

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 Equation (4).

2.2. Loss Function

The EggNet model is optimized to minimize the loss L:

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

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

$$l_\epsilon = y_\epsilon d_\epsilon^2 + \left(1 - y_\epsilon\right) \max^2\left(0, m - d_\epsilon\right), \tag{10}$$

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 EggNet 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 Equation (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:

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

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.

2.4. Models and Implementation

All MLPs in the EggNet model consist of 3 hidden layers, each with a width of 128 neurons. Each intermediate layer in these networks uses a SiLU (Elfwing et al., 2018) activation function and layer normalization (Ba et al., 2016). 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 EggNet model consists of up to 5 EggNet iterations ($i \leq 4$), and each graph attention 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 (Paszke et al., 2019) and PyTorch Lightning (Falcon & The PyTorch

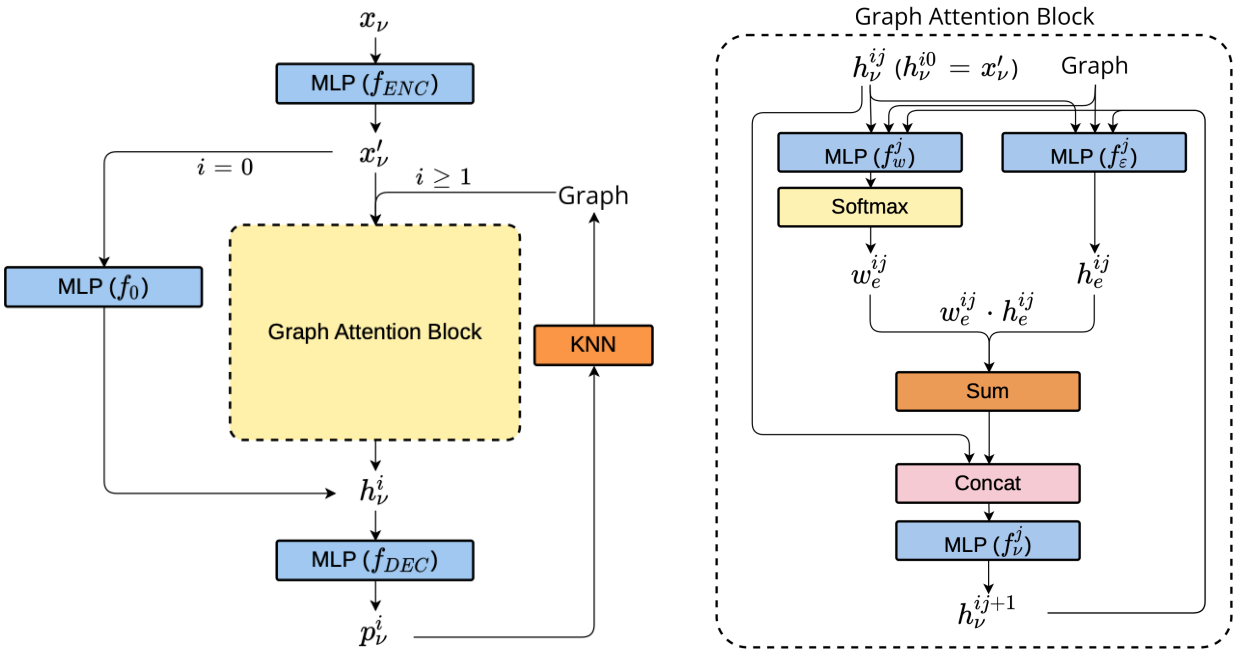

Figure 2. The EggNet architecture. i corresponds to each EggNet iteration, and j corresponds to each message passing step. An EggNet iteration generally consists of a graph attention block and a KNN. The first iteration does not perform graph attention and the last iteration does not perform KNN.

Lightning team, 2019) in the ACORN framework (Atkinson et al.) and simultaneously optimized with Adam (Kingma & Ba, 2014) 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 (Team, 2023).

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 $p_\mathrm{T} > 1$ GeV. This means hits coming from particles with $p_\mathrm{T} < 1$ GeV and noise hits are removed from our simulated datasets, reducing the number of hits by a factor of ~ 10. Note that applying such a selection on the transverse momentum significantly reduces the event size, making the task of track reconstruction much easier. It is therefore important to study the performance with the full events (no hits are removed). We leave the study with full events for future work.

We train the EggNet model with 12 input node features $\nu \in \mathbb{R}^{12}$ (described in (Ju et al., 2021)). We first evaluate the EggNet 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" $\mathrm{Eff}_\mathrm{KNN}$ and "edge-wise purity" [3] $\mathrm{Pur}_\mathrm{KNN}$:

$$\mathrm{Eff}_\mathrm{KNN} = \frac{N^{y=1}_\mathrm{KNN}}{N^{y=1}}, \tag{12}$$

$$\mathrm{Pur}_\mathrm{KNN} = \frac{N^{y=1}_\mathrm{KNN}}{N_\mathrm{KNN}} = \frac{N^{y=1}_\mathrm{KNN}}{N_\mathrm{hits} \cdot k}, \tag{13}$$

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 $N_\mathrm{hits} \cdot k$, and $N^{y=1}_\mathrm{KNN}$ is the number of true KNN edges. The EggNet performance is the best when $\mathrm{Eff}_\mathrm{KNN}$ and $\mathrm{Pur}_\mathrm{KNN}$ are closest to 1 (they cannot be larger than 1). Note that $N^{y=1}_\mathrm{KNN}$ cannot exceed k per node or the number of true edges for each source node:

$$N^{y=1}_\mathrm{KNN} \leq \sum_{\nu^S} \min\left(k,\ N^{y=1}_{\nu^S}\right). \tag{14}$$

This thus determines the upper bounds of $\mathrm{Eff}_\mathrm{KNN}$ and $\mathrm{Pur}_\mathrm{KNN}$. As shown in Figure 3, we plot $\mathrm{Eff}_\mathrm{KNN}$ and $\mathrm{Pur}_\mathrm{KNN}$ as a function of k. $\mathrm{Eff}_\mathrm{KNN}$ increases and approaches 1 as k increases, while $\mathrm{Pur}_\mathrm{KNN}$ decreases as k

[3] "Efficiency" and "purity" are also commonly known as "recall" and "precision".

increases. Upper bounds of $\mathrm{Eff}_{\mathrm{KNN}}$ and $\mathrm{Pur}_{\mathrm{KNN}}$ are also shown in the figure. We observe that both $\mathrm{Eff}_{\mathrm{KNN}}$ and $\mathrm{Pur}_{\mathrm{KNN}}$ are very close to the upper bounds for any k, which indicates that the EggNet 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 track performance. However, it allows us to evaluate the EggNet model performance with effects decoupled from the later stage of DBSCAN.

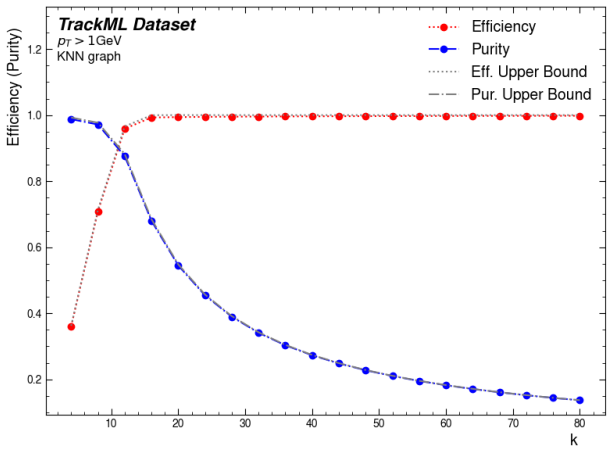

Figure 3. Edge-wise efficiency and purity in the KNN graphs as a function of k. The EggNet here consists of 5 iterations ($i \leq 4$). The upper bound for each metric is also shown in the plot.

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.

Figure 4 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 0.9956, duplication rate of 0.0129 and fake rate of 0.0006. Note that the choice of $\epsilon = 0.1$ here is made to maximize the track efficiency with the trade off of slightly higher duplication and fake rates.

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 (Ju et al., 2021) for graph construction. We simply take the same architecture as in the first EggNet iteratioin ($i = 0$), 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 EggNet model for both EC and OC approaches. For each scenario, we estimate the uncertainty by training 5 independent models and taking the standard deviation. The median is taken as the nominal result. A comparison with different number of EggNet iterations, and with methods that require a pre-constructed input graph, is shown in Table 1. We observe that EggNet starts outperforming the pre-constructed graph-based methods when the number of EggNet iterations is greater than 3 ($i \leq 2$).

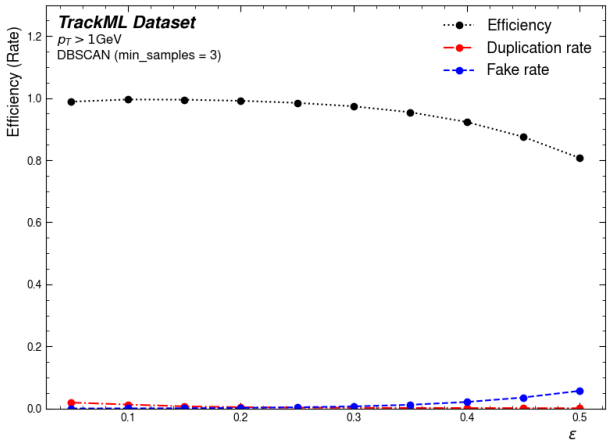

Figure 4. Track efficiency, duplication rate and fake rate with track candidates obtained from the DBSCAN clusters as a function of ϵ. The EggNet here consists of 5 iterations ($i \leq 4$).

Finally, Figure 5 shows the 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 0.26 seconds. The total computing time is also broken down into different components, including Graph Attention, KNN 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, termed EggNet.

Table 1. Track performance with EggNet with various number of EggNet iterations (i). $\epsilon = 0.1$ is used for the DBSCAN in all scenarios. The performance is compared with methods that require pre-constructed input graphs, including the edge classification approach (EC) and the object condensation approach (OC). The uncertainty in each scenario is estimated with the standard deviation of 5 independent models, and the median is taken as the nominal result.

METHOD	EFFICIENCY	DUP. RATE	FAKE RATE
EC	0.9898 ± 0.0009	0.0421 ± 0.0011	0.0012 ± 0.0000
OC	0.9902 ± 0.0007	0.0328 ± 0.0007	0.0015 ± 0.0001
EGGNET ($i \leq 1$)	0.7454 ± 0.0027	0.2202 ± 0.0024	0.0099 ± 0.0004
EGGNET ($i \leq 2$)	0.9905 ± 0.0005	0.0179 ± 0.0006	0.0011 ± 0.0000
EGGNET ($i \leq 3$)	0.9940 ± 0.0002	$\mathbf{0.0117 \pm 0.0003}$	$\mathbf{0.0005 \pm 0.0000}$
EGGNET ($\mathbf{i \leq 4}$)	$\mathbf{0.9956 \pm 0.0003}$	0.0129 ± 0.0002	0.0006 ± 0.0000

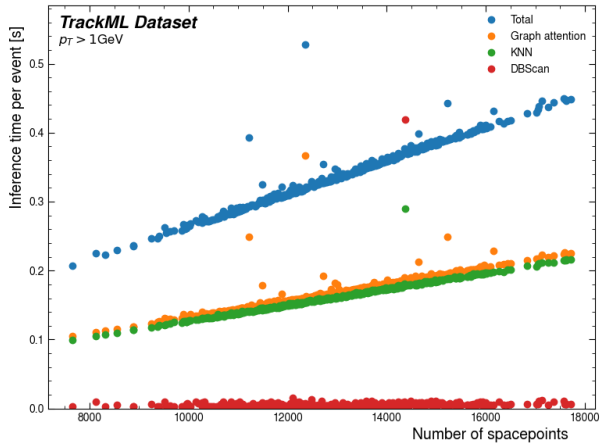

Figure 5. Inference time for each event versus the number of hits. The inference time is also broken down different components, including Graph Attention, KNN and DBSCAN. The EggNet here consists of 5 iterations ($i \leq 4$).

EggNet takes point clouds as inputs and builds graphs on the fly for each EggNet iteration. The constructed graphs are iteratively updated, improving message-passing efficacy. We demonstrated this approach with the TrackML dataset and obtained excellent track performance in both KNN graphs and reconstructed track candidates. EggNet also outperforms the pre-constructed graph-based tracking methods when sufficient EggNet iterations are performed. The result is encouraging and motivates future work to improve the computational performance. In particular, KNN has a significant contribution to the computing time. It will be beneficial to reduce the time consumption with an alternative method, such as approximate nearest neighbors and radius nearest neighbors. Furthermore, the performance with the full events where no hits are removed needs to be studied, and, if necessary, improved.

Impact Statement

The development of machine learning-based and in particular GNN-based particle track reconstruction algorithms provides significant potential for positive broader impacts in both the field of particle physics and society at large. By utilizing GNN as part of the track reconstruction pipeline, it is possible for the computational cost to scale linearly with the event size, which accelerates the discovery potential of future particle experiments such as HL-LHC. The method's potential benefits include enhancing our understanding of fundamental particles and their interactions, thereby advancing scientific knowledge and contributing to the development of more precise experimental techniques in particle physics research. Furthermore, EggNet has the potential to extend beyond particle physics, with applications in various domains that involve pattern recognition and representation learning. For instance, it could find applications in medical imaging, where accurate reconstruction of complex image structures from noisy data is crucial.

References

Aaboud, M. et al. Performance of the ATLAS Track Reconstruction Algorithms in Dense Environments in LHC Run 2. *Eur. Phys. J. C*, 77(10):673, 2017. doi: 10.1140/epjc/s10052-017-5225-7.

Amrouche, S. et al. *The Tracking Machine Learning challenge : Accuracy phase*. 4 2019. doi: 10.1007/978-3-030-29135-8_9.

Amrouche, S. et al. The Tracking Machine Learning Challenge: Throughput Phase. *Comput. Softw. Big Sci.*, 7(1): 1, 2023. doi: 10.1007/s41781-023-00094-w.

Atkinson, M. J., Caillou, S., Clafiura, P., Collard, C., Farrell, S. A., Huth, B., Ju, X., Liu, R., Minh Pham, T., Murnane, D. c. a., Neubauer, M., Rougier, C., Stark, J., Torres, H., and Vallier, A. gnn4itk. URL https://github.com/GNN4ITkTeam/CommonFramework.

Ba, J. L., Kiros, J. R., and Hinton, G. E. Layer normalization, 2016.

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.*, 251:03047, 2021. doi: 10.1051/epjconf/202125103047.

Caillou, S., Calafiura, P., Farrell, S. A., Ju, X., Murnane, D. T., Rougier, C., Stark, J., and Vallier, A. ATLAS ITk Track Reconstruction with a GNN-based pipeline. Technical report, CERN, Geneva, 2022. URL https://cds.cern.ch/record/2815578.

Chatrchyan, S. et al. Description and performance of track and primary-vertex reconstruction with the CMS tracker. *JINST*, 9(10):P10009, 2014. doi: 10.1088/1748-0221/9/10/P10009.

Elfwing, S., Uchibe, E., and Doya, K. Sigmoid-weighted linear units for neural network function approximation in reinforcement learning. *Neural Networks*, 107:3–11, 2018. ISSN 0893-6080. doi: https://doi.org/10.1016/j.neunet.2017.12.012. URL https://www.sciencedirect.com/science/article/pii/S0893608017302976. Special issue on deep reinforcement learning.

Ester, M., Kriegel, H.-P., Sander, J., and Xu, X. A density-based algorithm for discovering clusters in large spatial databases with noise. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pp. 226–231. AAAI Press, 1996.

Falcon, W. and The PyTorch Lightning team. PyTorch Lightning, March 2019. URL https://github.com/Lightning-AI/lightning.

Ju, X. et al. Performance of a geometric deep learning pipeline for HL-LHC particle tracking. *Eur. Phys. J. C*, 81 (10):876, 2021. doi: 10.1140/epjc/s10052-021-09675-8.

Kingma, D. and Ba, J. Adam: A method for stochastic optimization. 2014.

Lazar, A. et al. Accelerating the Inference of the Exa.TrkX Pipeline. *J. Phys. Conf. Ser.*, 2438(1):012008, 2023. doi: 10.1088/1742-6596/2438/1/012008.

Lieret, K., DeZoort, G., Chatterjee, D., Park, J., Miao, S., and Li, P. High Pileup Particle Tracking with Object Condensation. 12 2023.

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. In Wallach, H., Larochelle, H., Beygelzimer, A., d'Alché-Buc, F., Fox, E., and Garnett, R. (eds.), *Advances in Neural Information Processing Systems 32*, pp. 8024–8035. Curran Associates, Inc., 2019. URL http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance pdf.

Qasim, S. R., Kieseler, J., Iiyama, Y., and Pierini, M. Learning representations of irregular particle-detector geometry with distance-weighted graph networks. *Eur. Phys. J. C*, 79(7):608, 2019. doi: 10.1140/epjc/s10052-019-7113-9.

Strandlie, A. and Frühwirth, R. Track and vertex reconstruction: From classical to adaptive methods. *Rev. Mod. Phys.*, 82:1419–1458, May 2010. doi: 10.1103/RevModPhys.82.1419. URL https://link.aps.org/doi/10.1103/RevModPhys.82.1419.

Team, R. D. *RAPIDS: Libraries for End to End GPU Data Science*, 2023. URL https://rapids.ai.

Veličković, P., Cucurull, G., Casanova, A., Romero, A., Liò, P., and Bengio, Y. Graph attention networks. In *International Conference on Learning Representations*, 2018. URL https://openreview.net/forum?id=rJXMpikCZ.