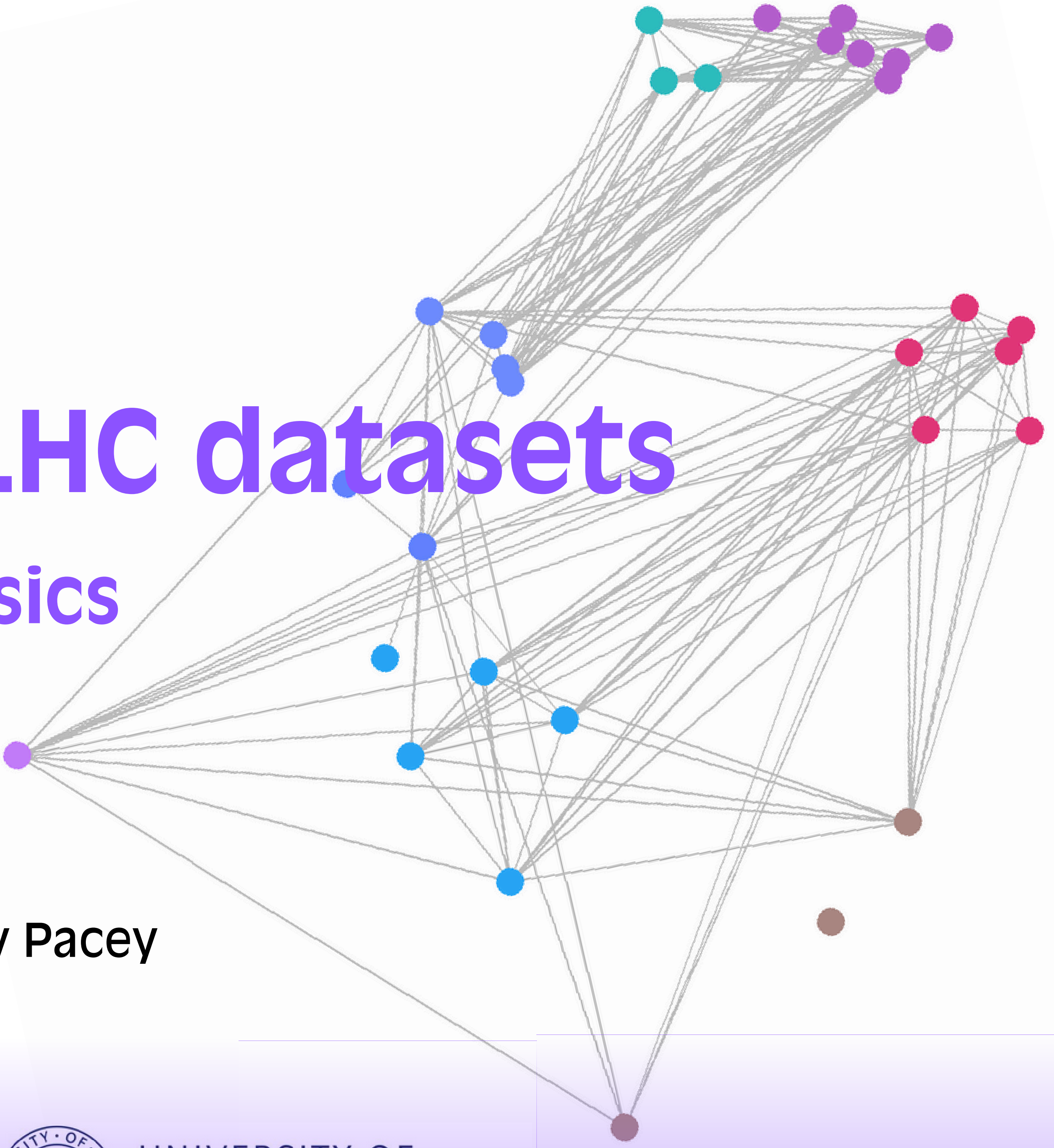
Graph Neural Networks on LHC datasets

in searches for new physics

EPS HEP Marseille 2025

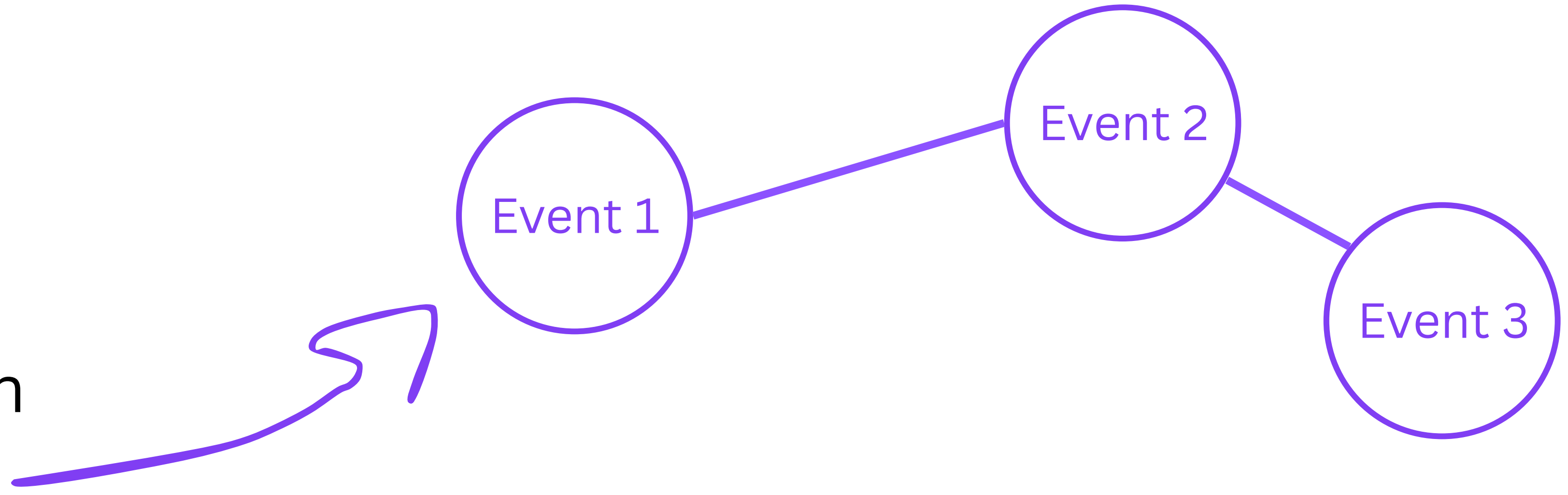


Anna Mullin

Maggie Chen, Sebastian Rutherford, Holly Pacey

UNIVERSITY OF CAMBRIDGE · GATES Cambridge · SCHMIDT FUTURES · UNIVERSITY OF OXFORD

Project scope



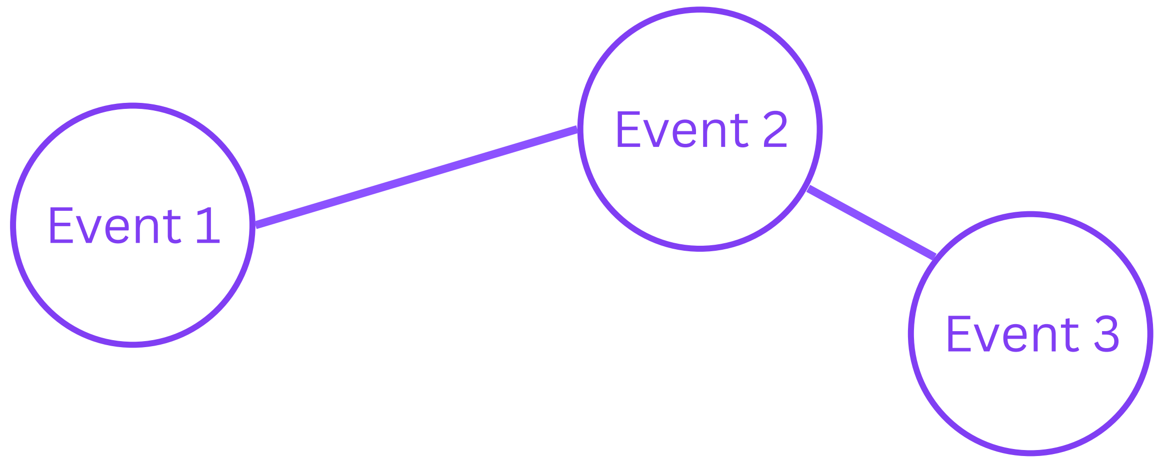
- In most GNNs for HEP, events are graphs

- We propose a unique graph construction
  - with **entire LHC dataset as a graph**
  - where **events (as nodes) are connected (by edges)** if they have similar kinematics

- Classify nodes by learning their connectivity in a neighbourhood of similar nodes

→ How do the graphs impact performance?

Project scope



- In most GNNs for HEP, events are graphs
- We propose a unique graph construction
  - with **entire LHC dataset as a graph**
  - where **events (as nodes) are connected (by edges)** if they have similar kinematics
- Classify nodes by learning their connectivity in a neighbourhood of similar nodes

- We compare graph- and non-graph approaches, in parallel studies:

1 **model-dependent** search

- GNN: graph **convolutional layers** aggregate features of each node's neighbours
- Compare: **convolutional GNN vs DNN**

2 **anomaly detection**

- **Unsupervised** learning with **autoencoder** (AE) and GCN-based graph AE (GAE)
- Compare: **GAE vs AE**

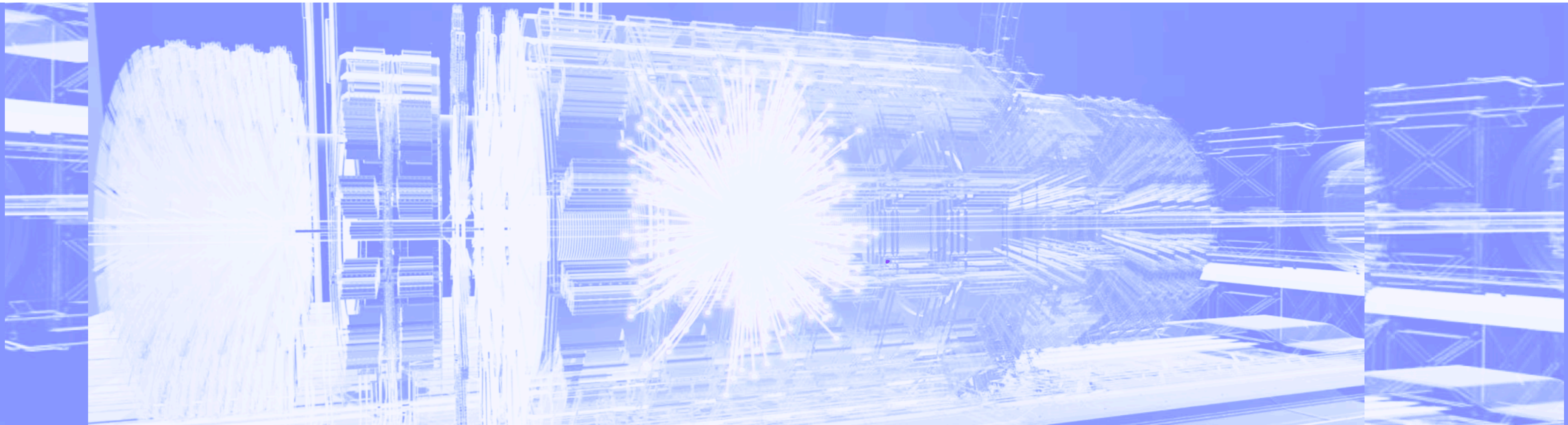
→ How do the graphs impact performance?

Graphs of LHC events

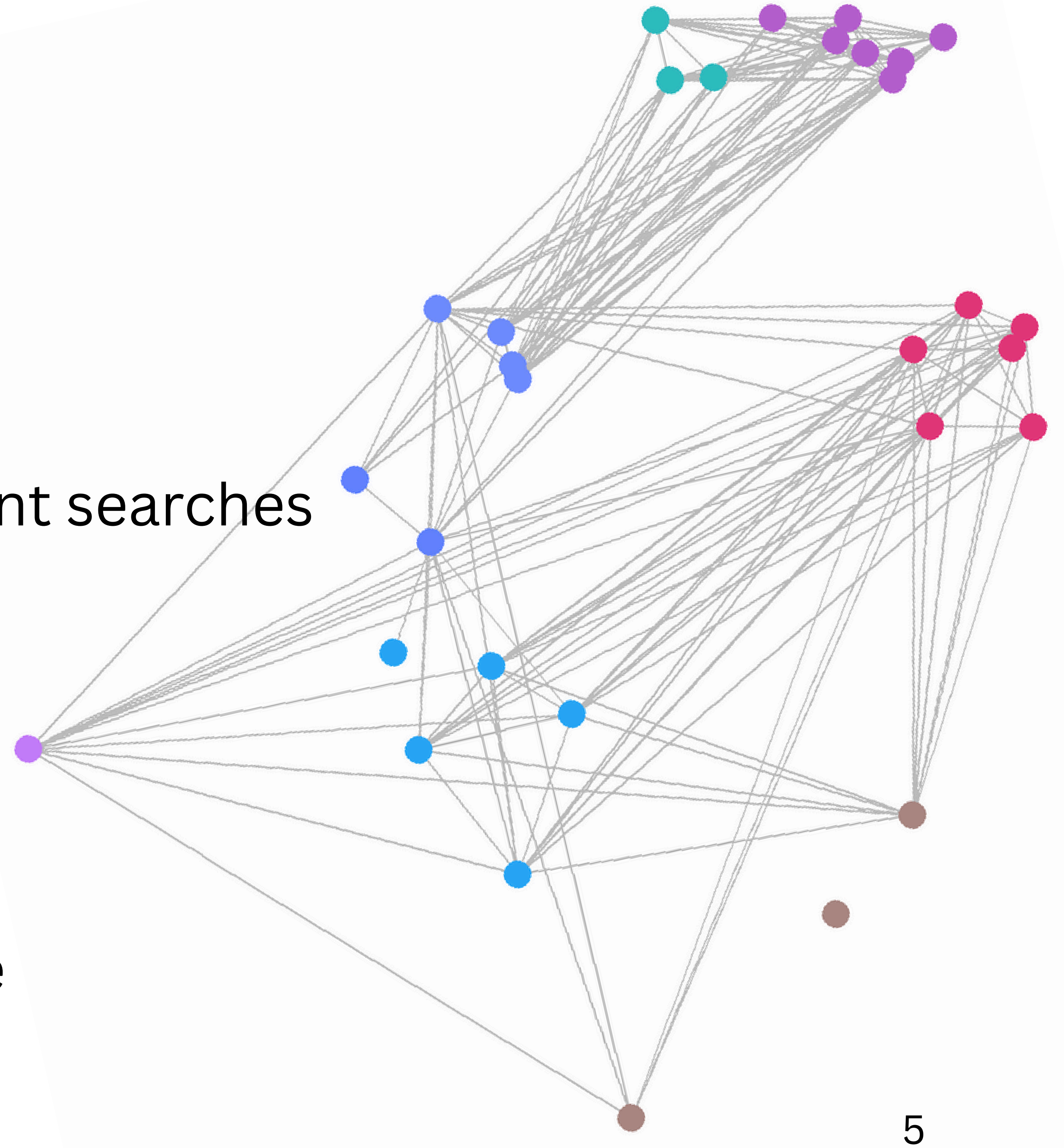
- Collider measurements do not directly reveal underlying physics, so we infer likely processes
- **Graph topology** highlights data subsets that **share characteristics**
  - E.g. in our case: similar decay chains, intermediate states, production modes
- Identify structures of subgraphs: **signal-signal, signal-background, background-background**
- We seek these structures via:
  - network analysis with graph theory
  - more powerful approaches **using GNNs**

Does SUSY have friends? Paper 2020

New work in this talk

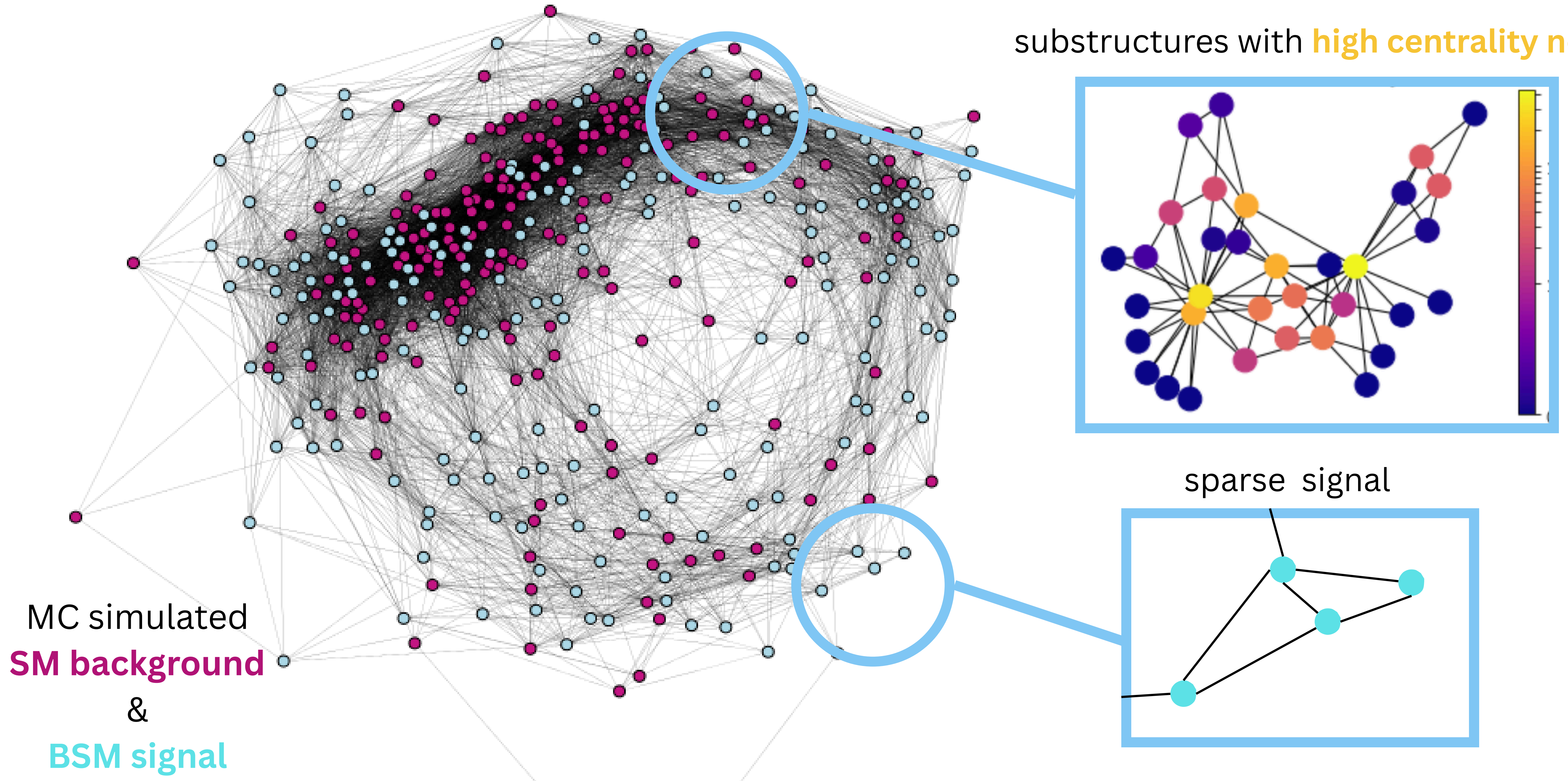


Contents



1. Graph design

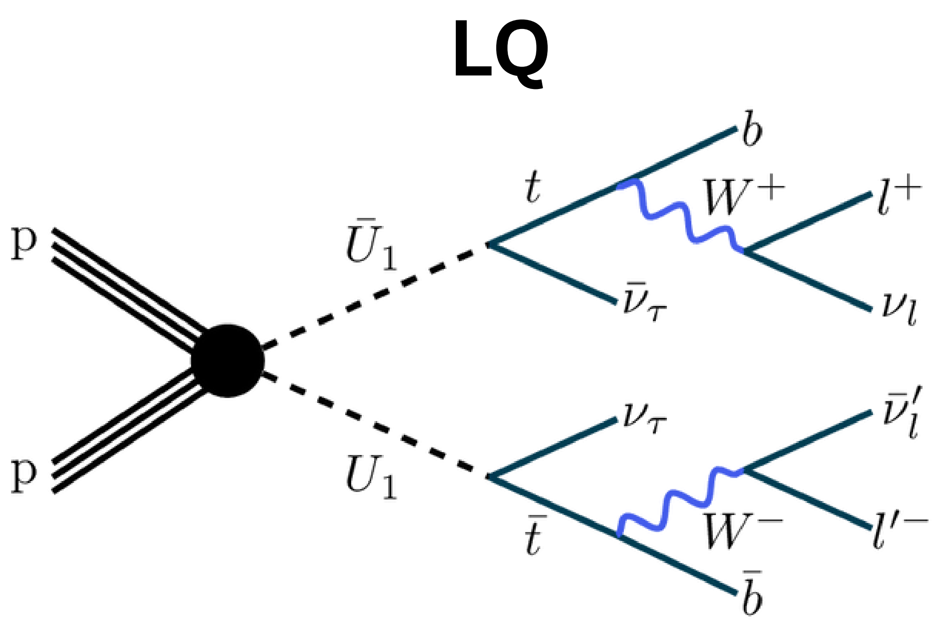
Constructing graphs



Can we achieve **additional discrimination** of signal from background from **graph topology and substructure?**

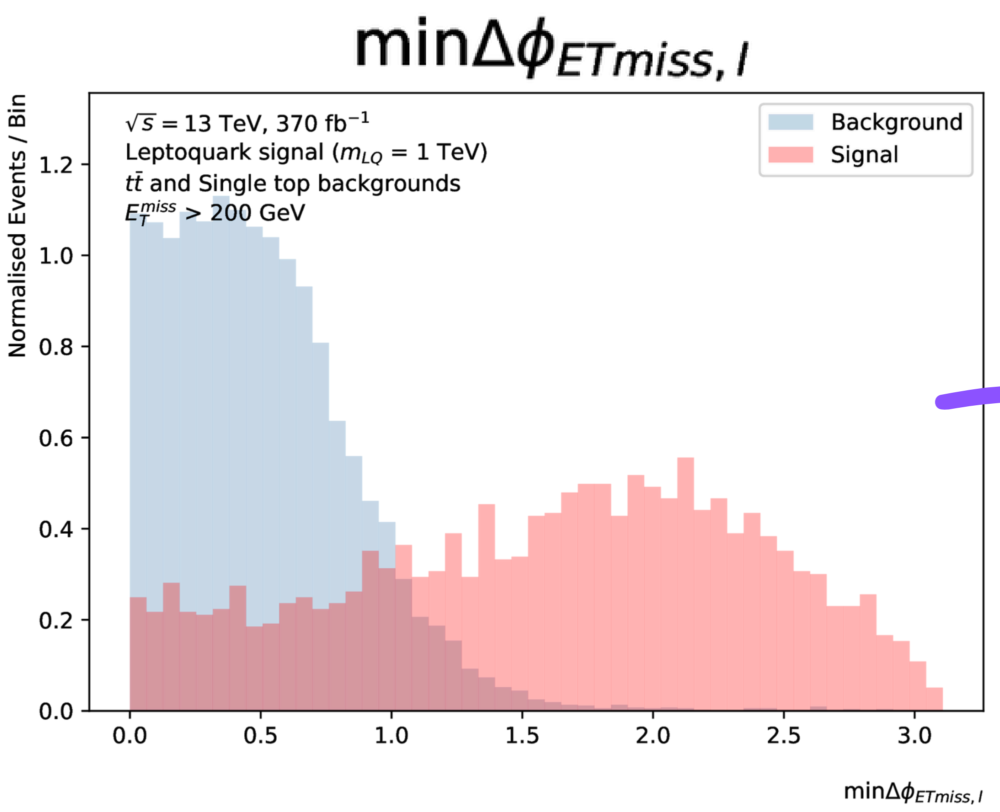
Event selection: signal, background

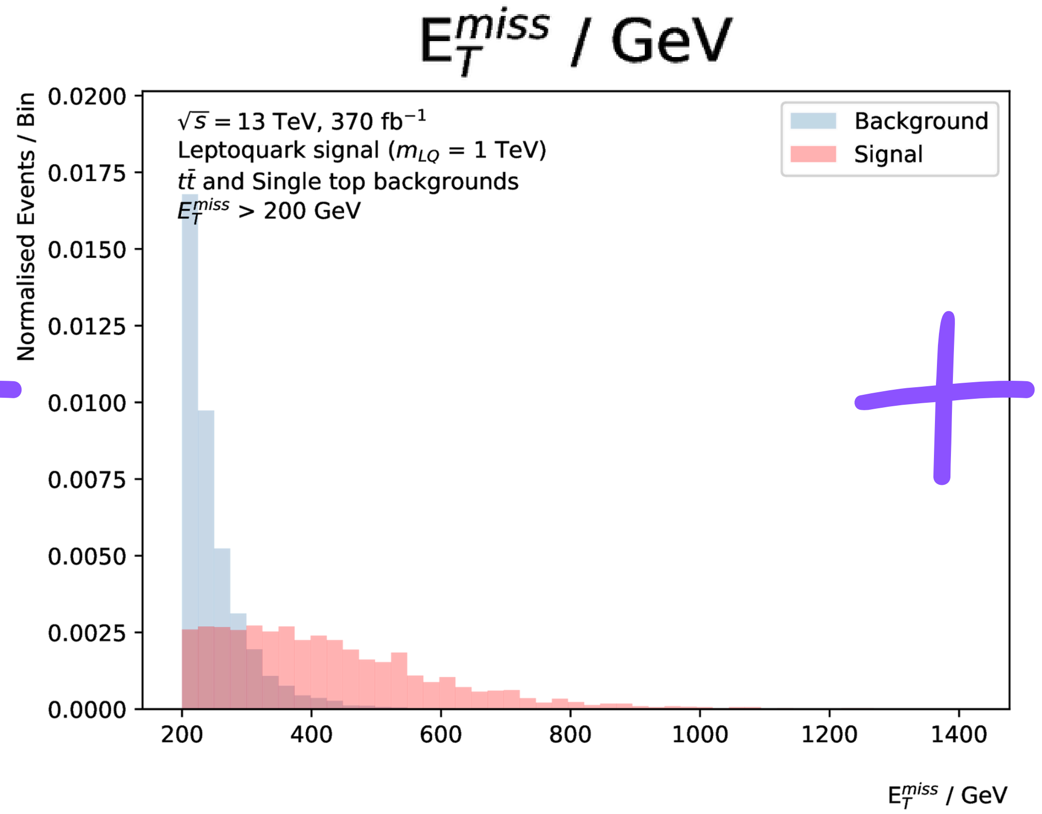
- Simulate **leptoquark** model which has no dedicated search yet:
  - vector leptoquark coupling to top-neutrino (backgrounds: single-top, ttbar)
  - apply preselections (MET > 200 GeV)



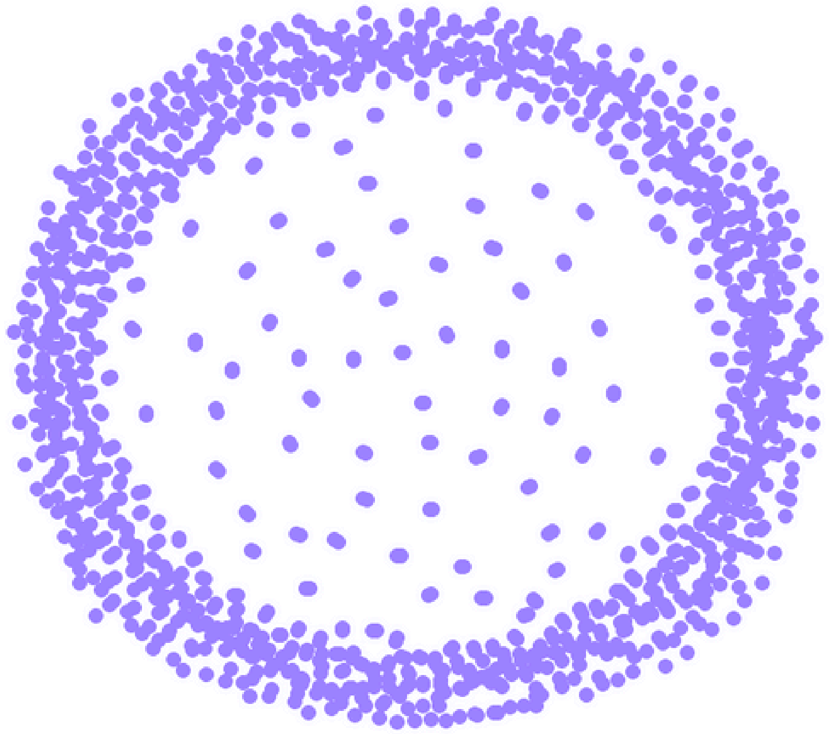
- Choose a **discriminating set of *N* kinematics**, e.g:
  - **High-level kinematics**: composite, often make physics assumptions
  - **Low-level kinematics**: final state particle 4-momenta

- Standardise chosen kinematics → avoid dominance by largest values





others ....



An example of signal+background events in N-dimensional kinematic space compressed to 2D

Events are points in an ***N*-dimensional kinematic vector space**

Distances

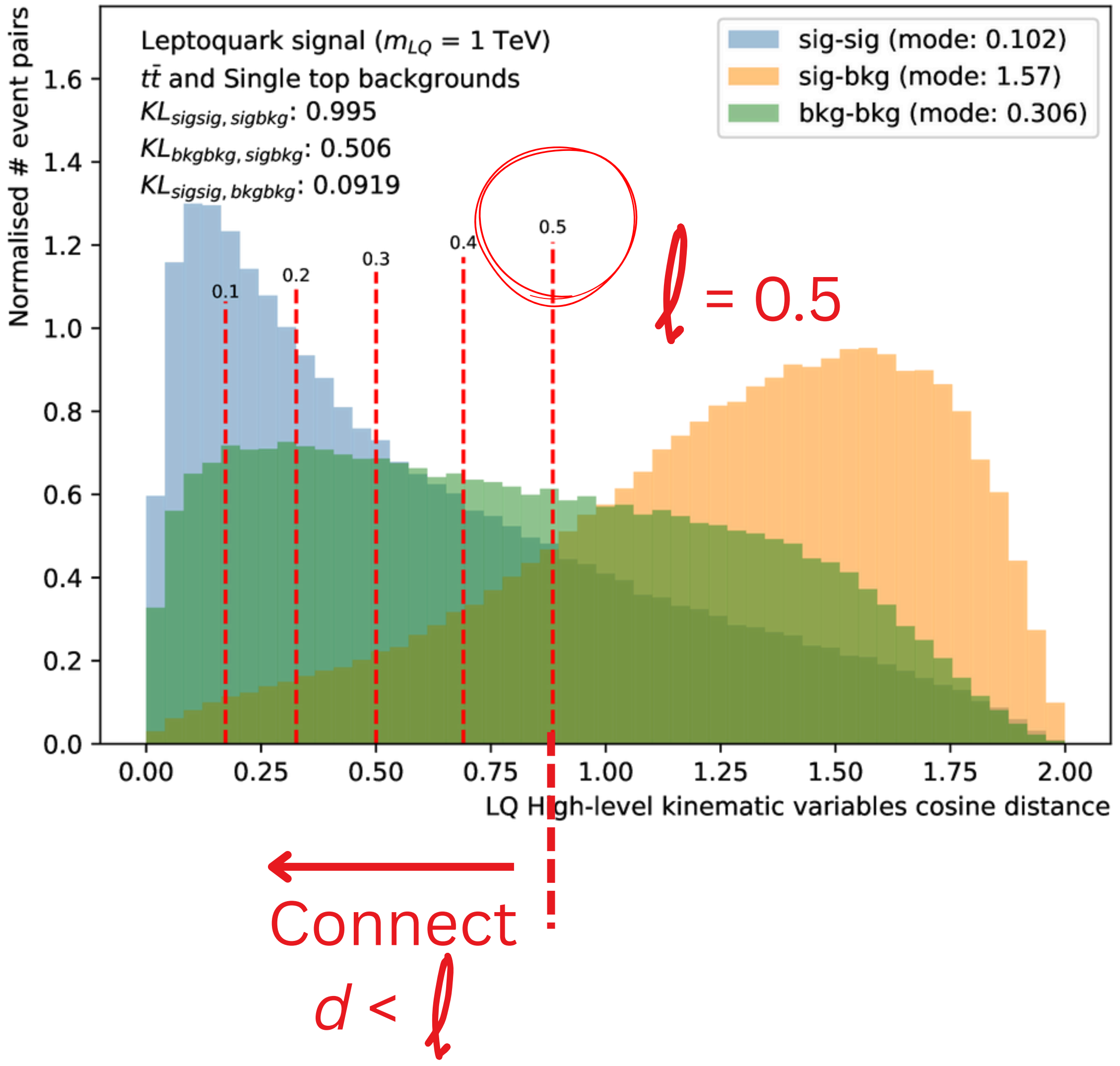
- Calculate distances between events in the **N-dimensional kinematic space**
  - Typical familiar metrics, e.g. between events *u, v:*

1. **Euclidean distance** $d_{\text{euc}} = \sqrt{\sum_{i=1}^{n} (u_i - v_i)^2}.$
2. **Cosine distance** $d_{\cos} = 1 - \frac{u \cdot v}{\sqrt{u \cdot u}\sqrt{v \cdot v}}$
3. **Earth Mover's Distance (EMD)**
   A measure of how different two distributions are in shape and magnitude: see *this source*

Distances

- Calculate distances between events in the **N-dimensional kinematic space**
  - Typical familiar metrics, e.g. between events *u, v:*
    1. **Euclidean distance** $d_{\text{euc}} = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}$.
    2. **Cosine distance** $d_{\cos} = 1 - \frac{u \cdot v}{\sqrt{u \cdot u}\sqrt{v \cdot v}}$
    3. **Earth Mover's Distance (EMD)**
       A measure of how different two distributions are in shape and magnitude: see [this source]

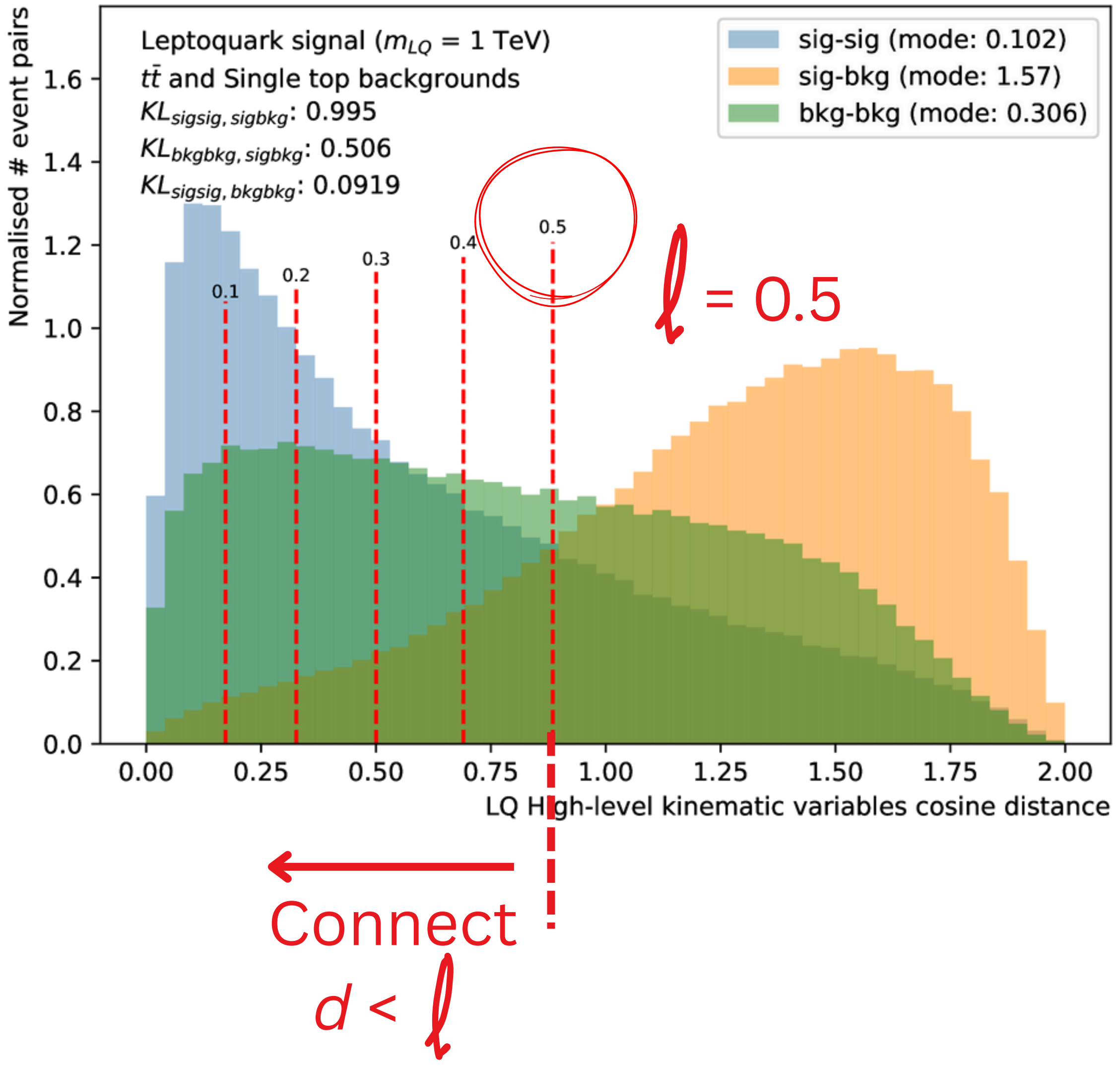
Distance distribution



Distances

- Calculate distances between events in the **N-dimensional kinematic space**
  - Typical familiar metrics, e.g. between events *u, v:*
    1. **Euclidean distance** $d_{\text{euc}} = \sqrt{\sum_{i=1}^{n}(u_i - v_i)^2}$
    2. **Cosine distance** $d_{\cos} = 1 - \frac{u \cdot v}{\sqrt{u \cdot u}\sqrt{v \cdot v}}$
    3. **Earth Mover's Distance (EMD)**
       A measure of how different two distributions are in shape and magnitude: see *this source*

Distance distribution



Adjacency matrix

- Convert events into nodes with edges if their distance *d* in the kinematic space is closer than linking length ℓ

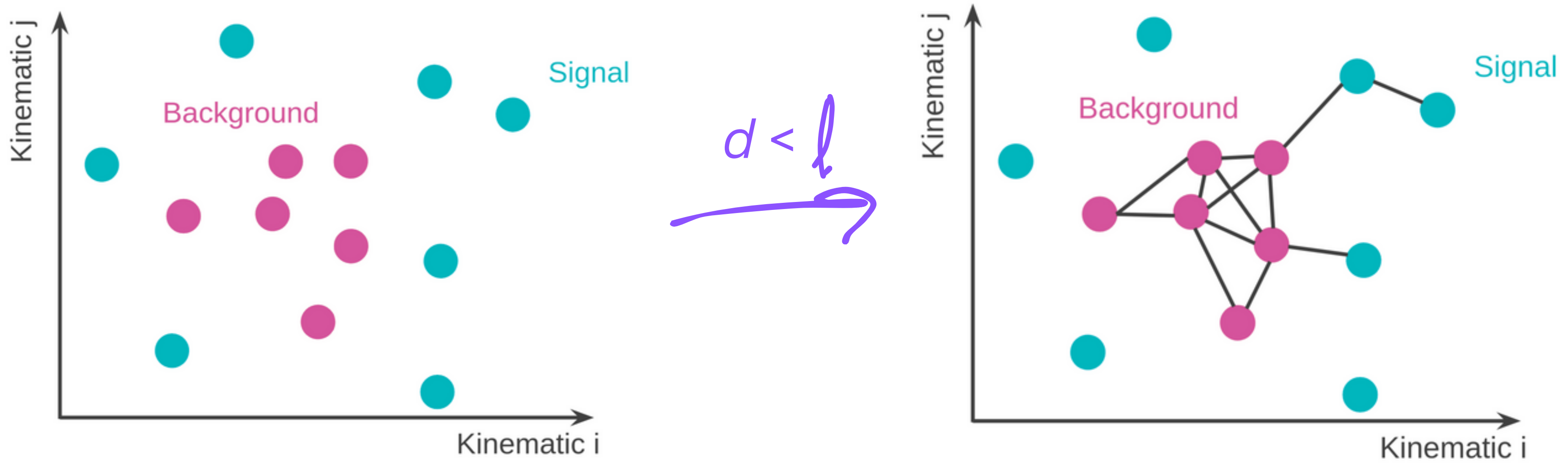
$$a_{ij} = \begin{cases} 1, & \text{if } d_{ij} \leq \ell, \\ 0, & \text{otherwise} \end{cases}$$

$$= \begin{bmatrix} 1 & 0 & 1 & & \\ 0 & 1 & 1 & & \cdots \\ 1 & 1 & 1 & & \\ & \vdots & & \ddots & \\ & & & & 1 \end{bmatrix}$$

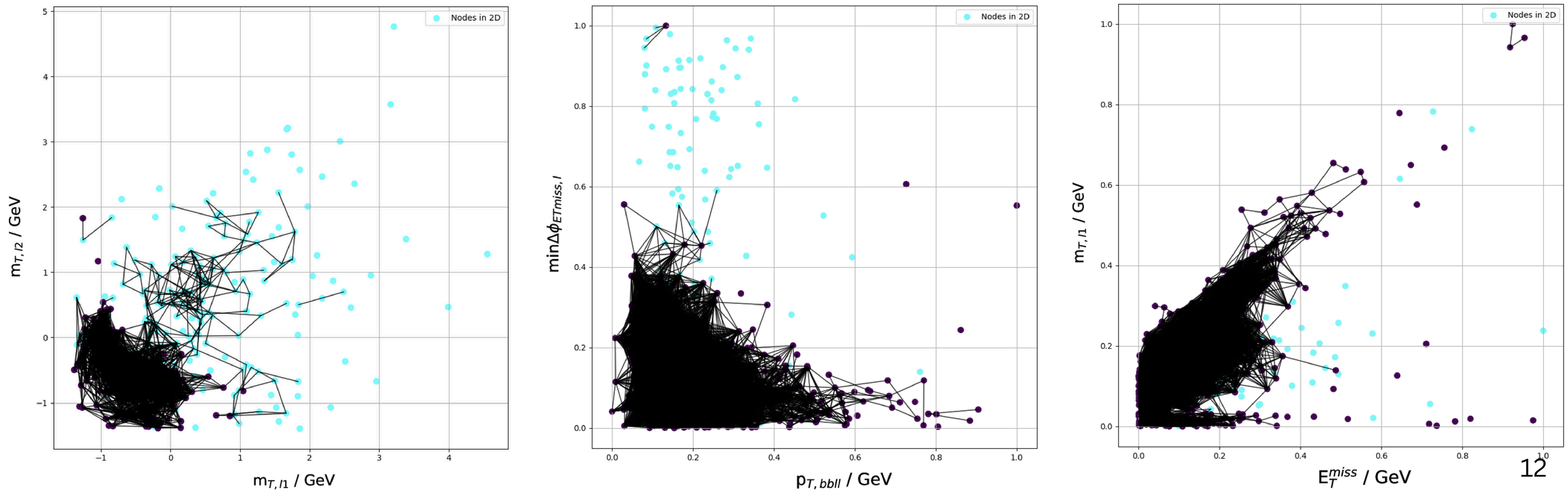
Graph

- Nodes with kinematic features
- Edges encoding structure

Graph

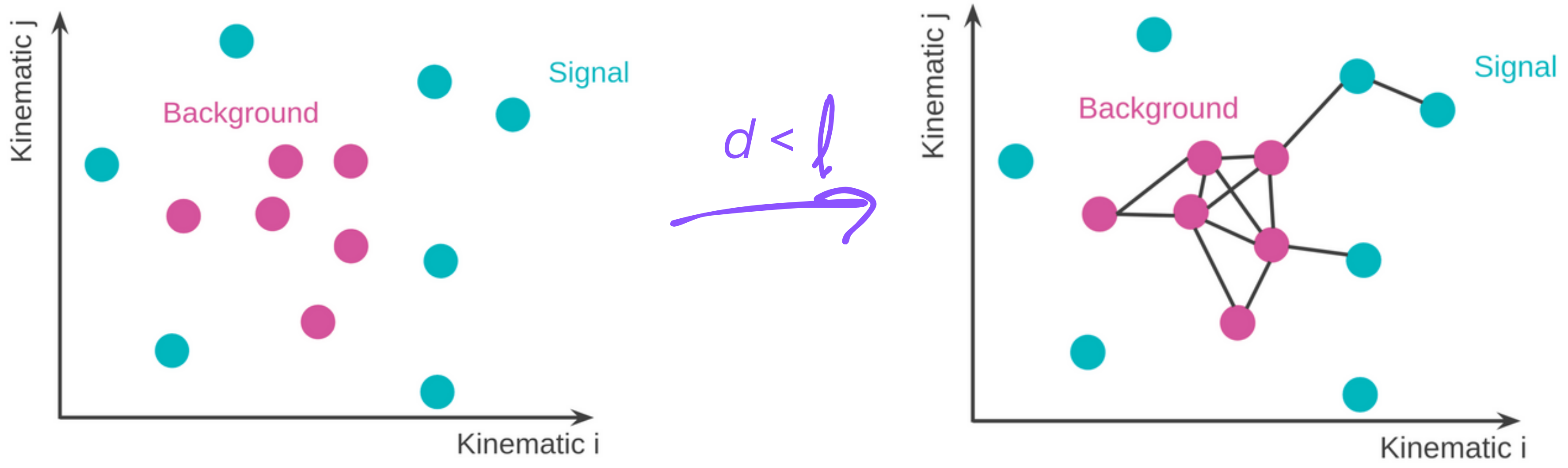


MC simulated
background (ttbar, singletop)
& injection of **signal (leptoquark)**

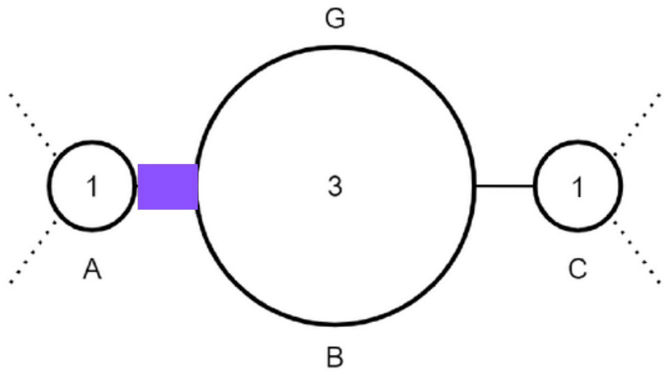


1. Graph design

Graph

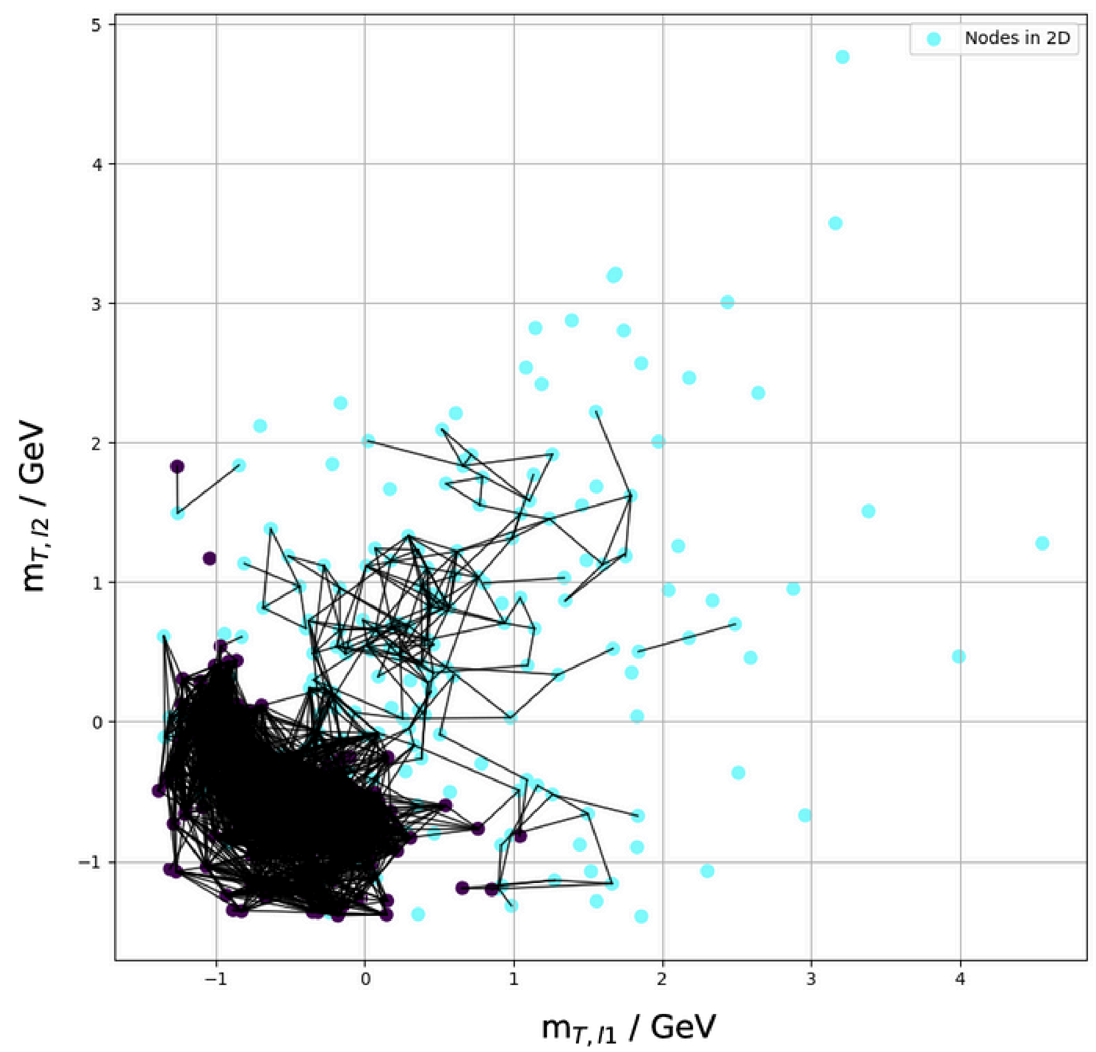


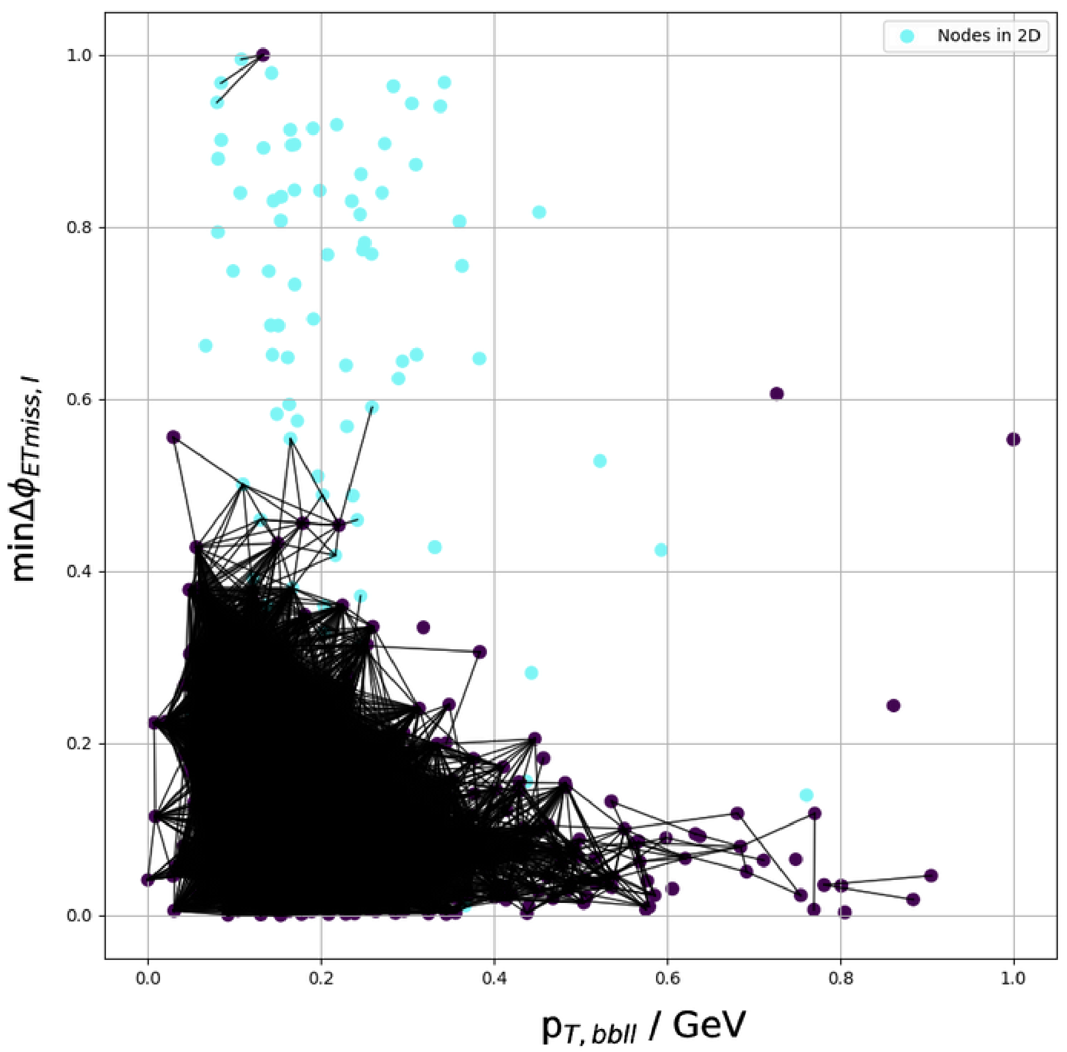
Final construction: add node weights and edge weights

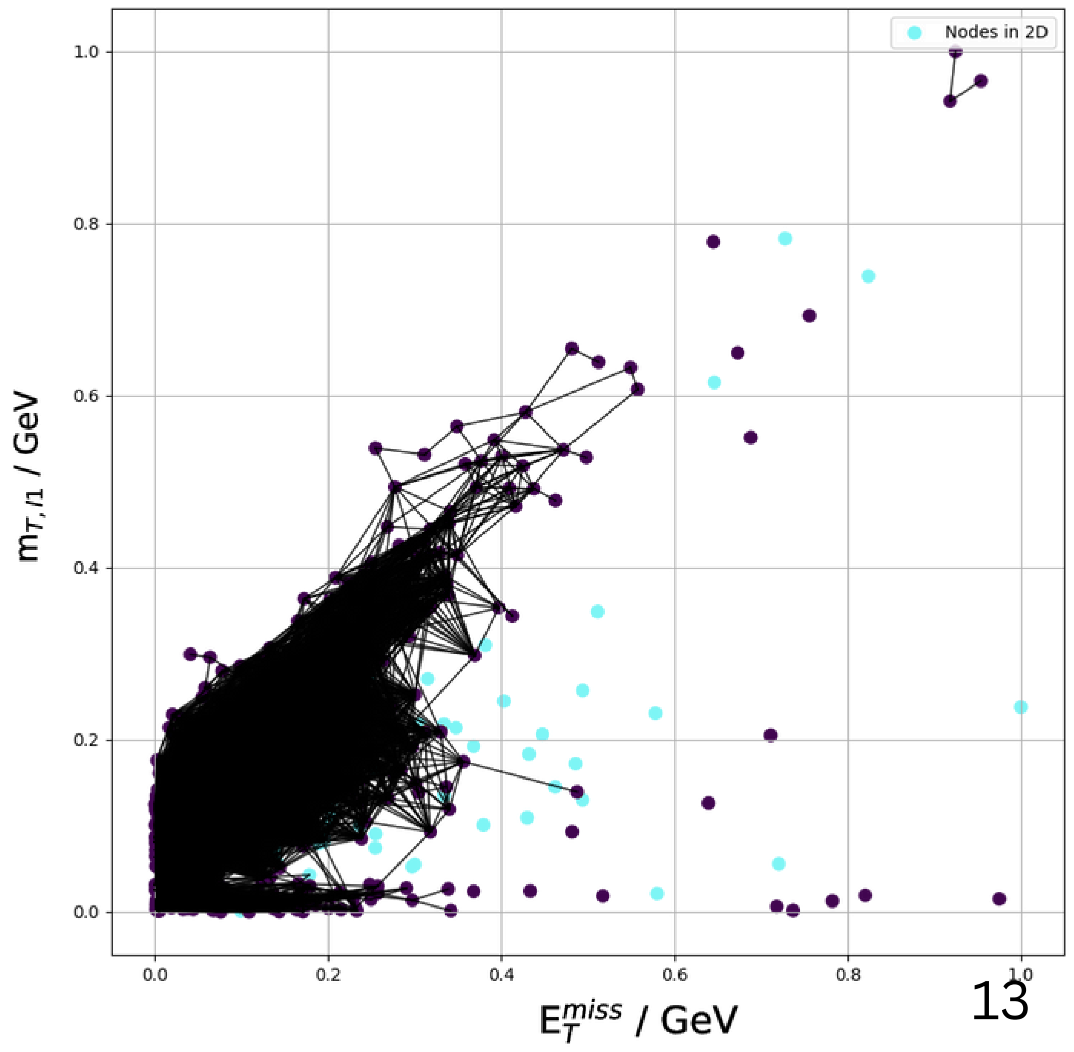


- Nodes: MC event weights
- Edges: MC x inverse distance

MC simulated
background (ttbar, singletop)
& injection of **signal (leptoquark)**





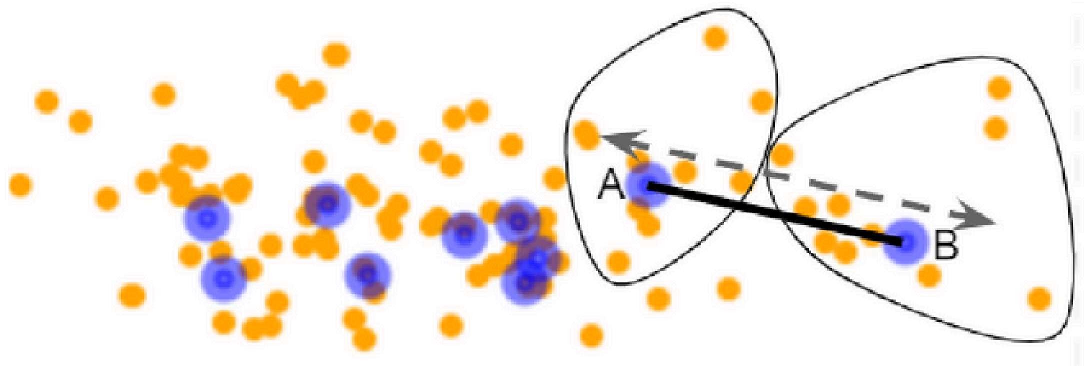


Validation

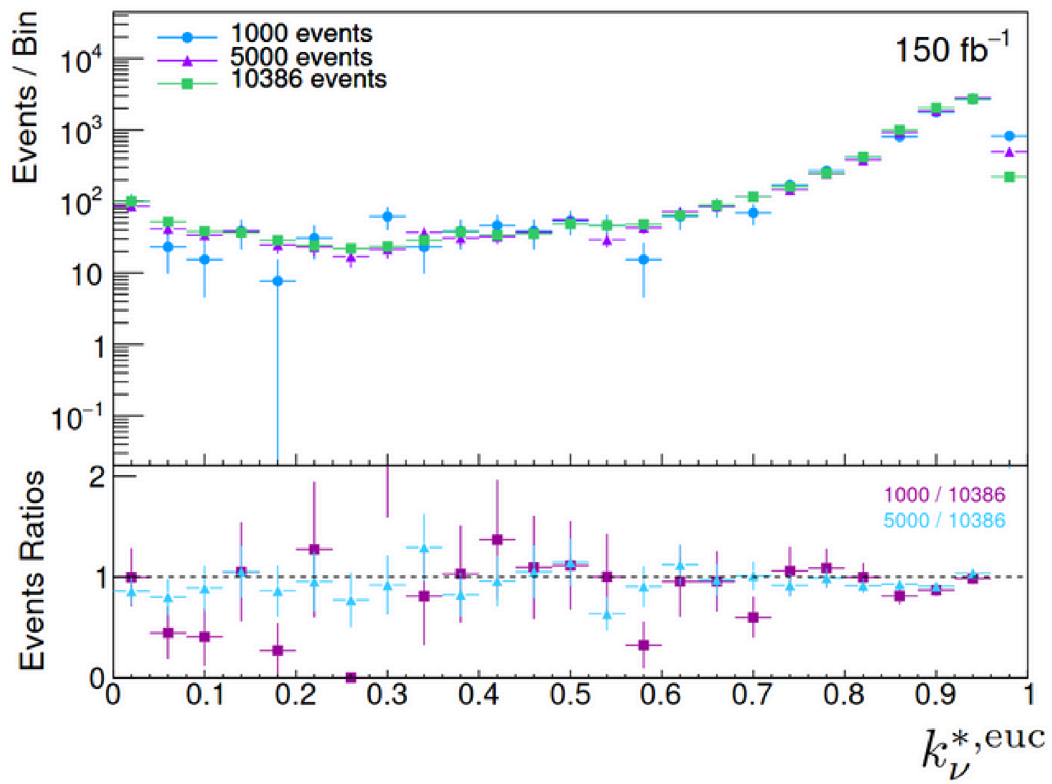
Does MC graph behaviour represent real data graphs?

Possible biases

- MC graph represents **true proportions** of events using **node weights** (preserve kinematic shapes)
  - when oversampling to improve modelling
  - or subsampling over-represented processes



Checks



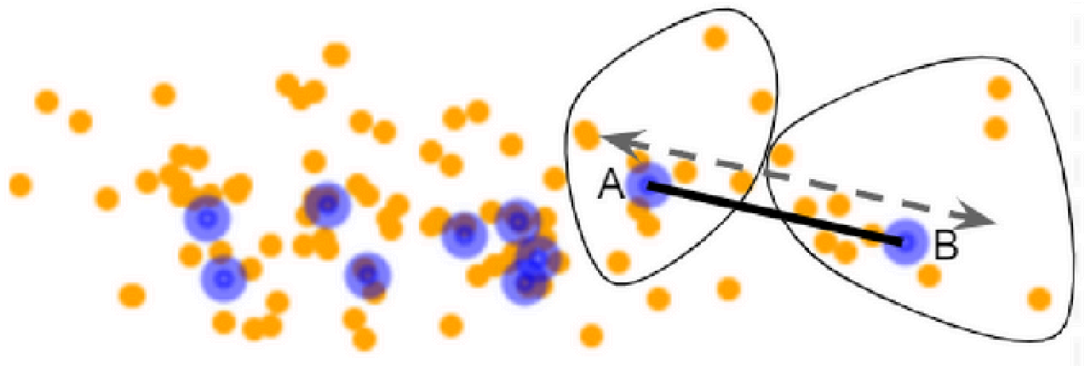
Validation

Does MC graph behaviour represent real data graphs?

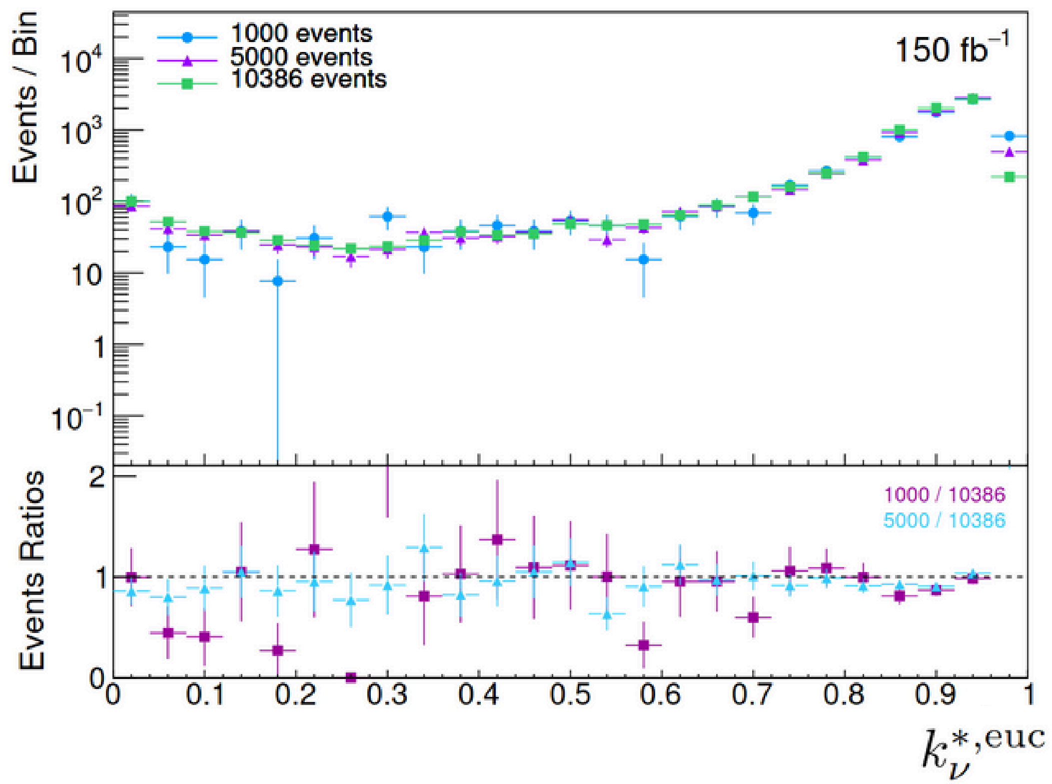
Possible biases	Checks

Possible biases

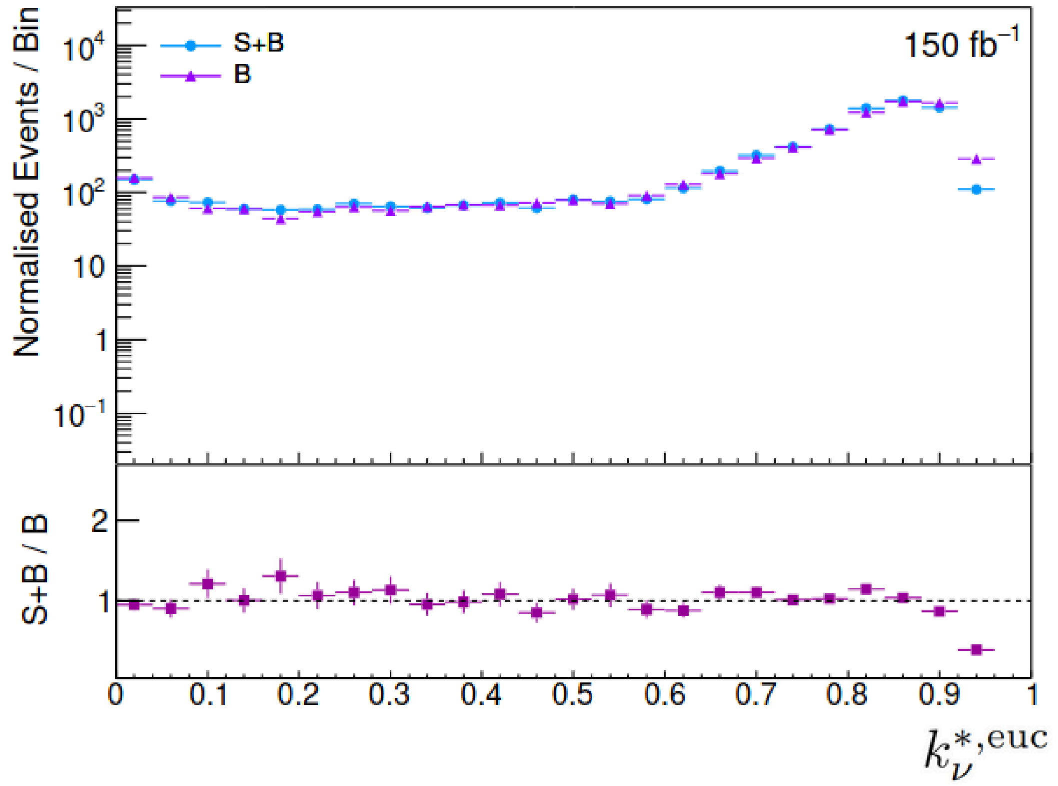
- MC graph represents **true proportions** of events using **node weights** (preserve kinematic shapes)
  - when oversampling to improve modelling
  - or subsampling over-represented processes



Checks



- MC graphs connect signal & background to characterise signal hypotheses, yet also characterise **background-only** null hypothesis:
  - ensure that **SM-only graph** is consistent with **SM in SM+signal graph**



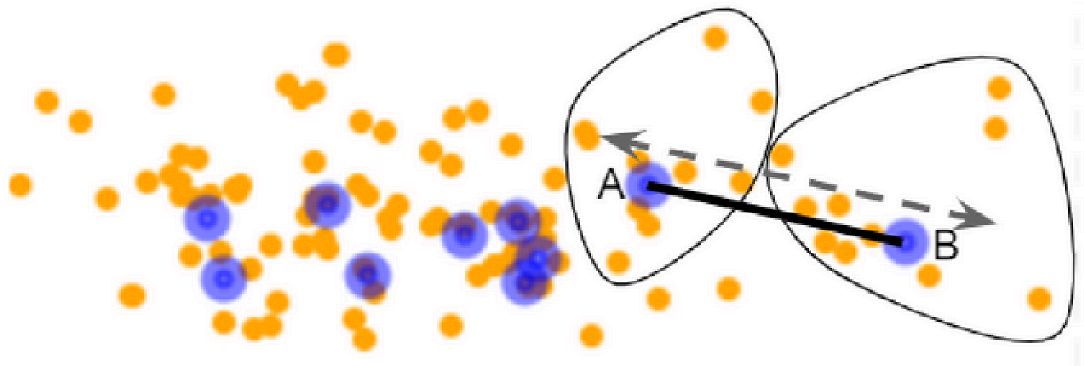
Validation

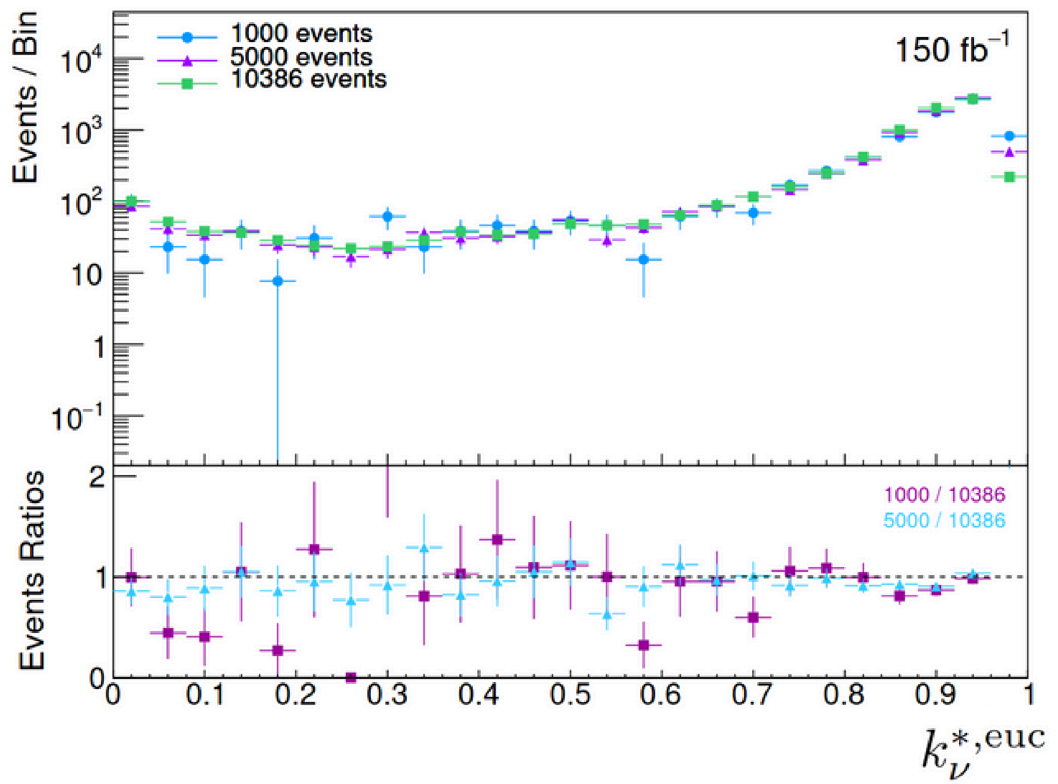
Does MC graph behaviour represent real data graphs?

Possible biases

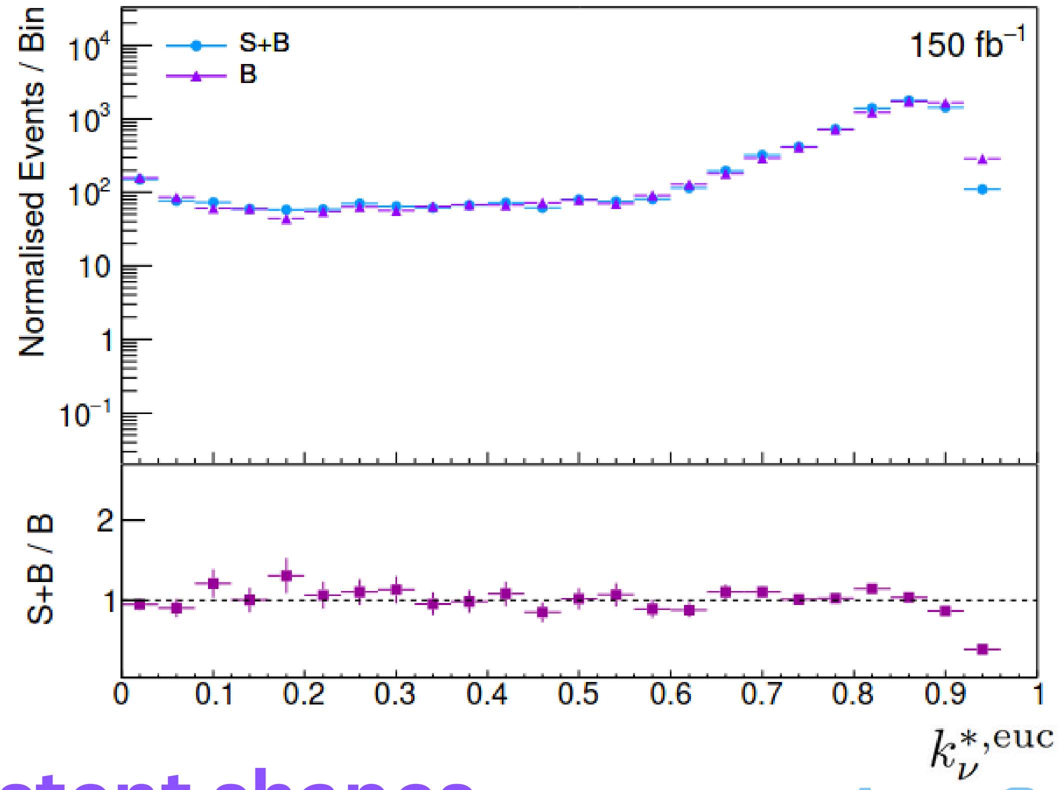
Checks

- MC graph represents **true proportions** of events using **node weights** (preserve kinematic shapes)
  - when oversampling to improve modelling
  - or subsampling over-represented processes



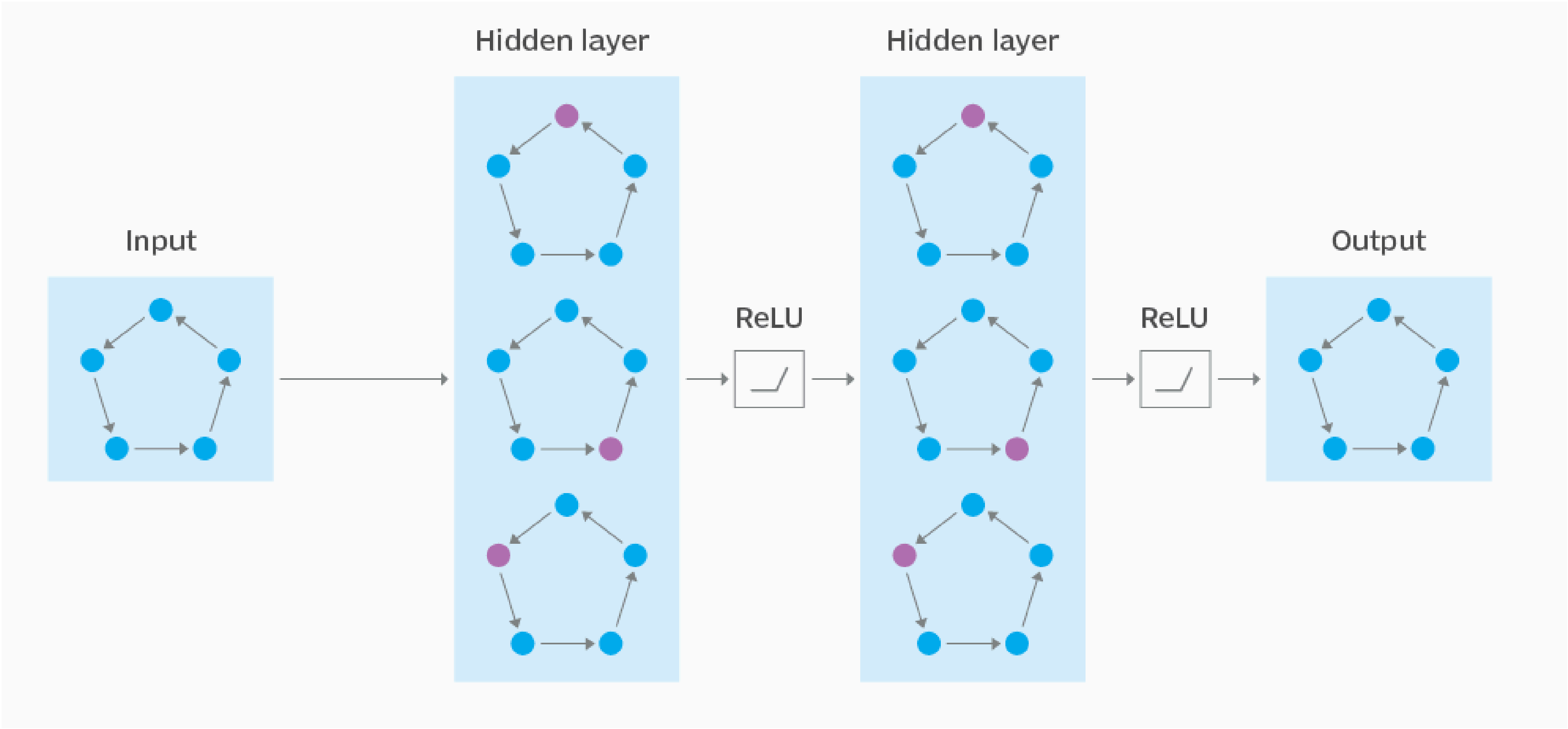


- MC graphs connect signal & background to characterise signal hypotheses, yet also characterise **background-only** null hypothesis:
  - ensure that **SM-only graph** is consistent with **SM in SM+signal graph**



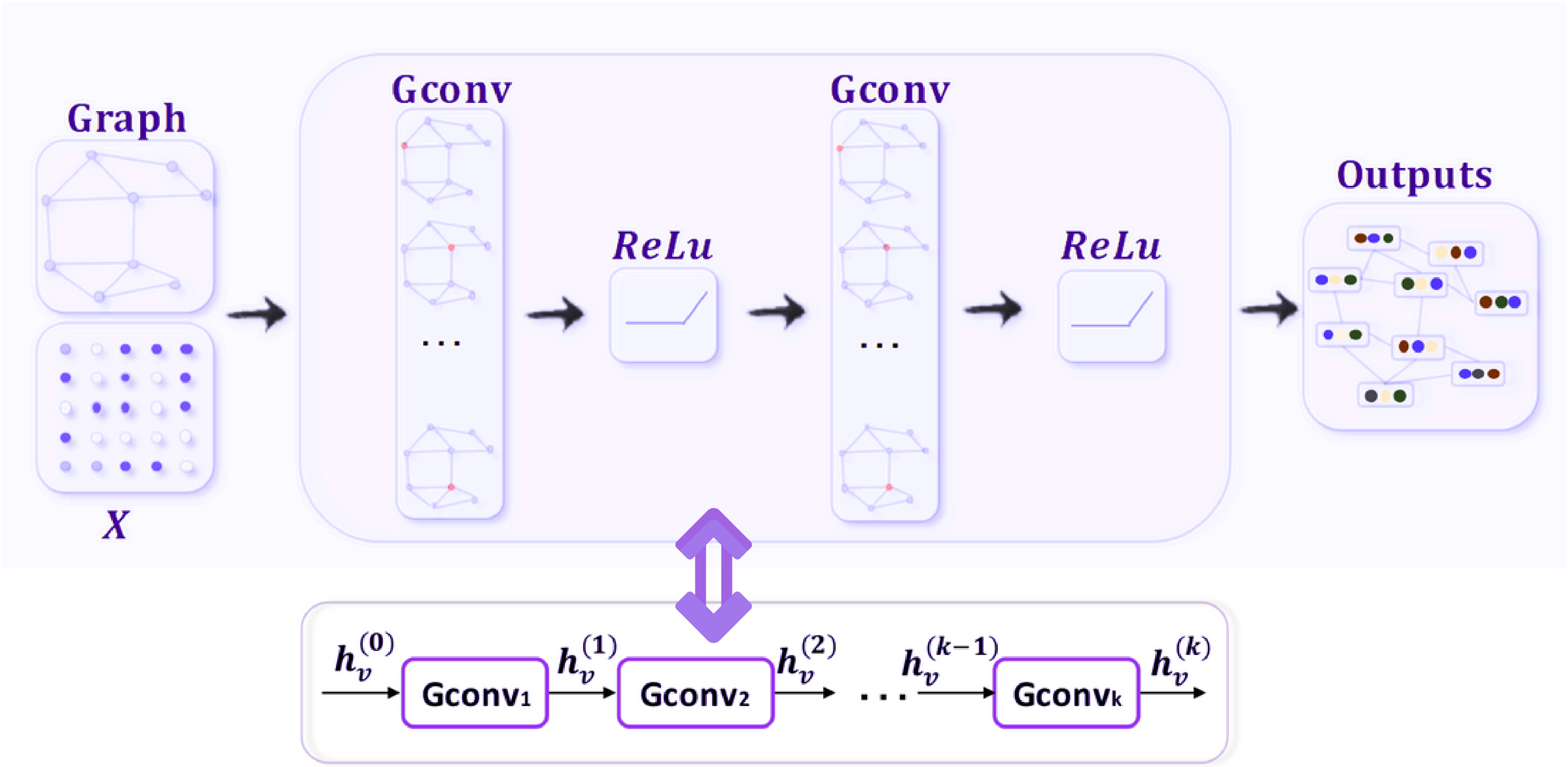
→ bulk distributions have **consistent shapes**

2. ML construction



GNN: model-dependent search

Graph convolutional networks

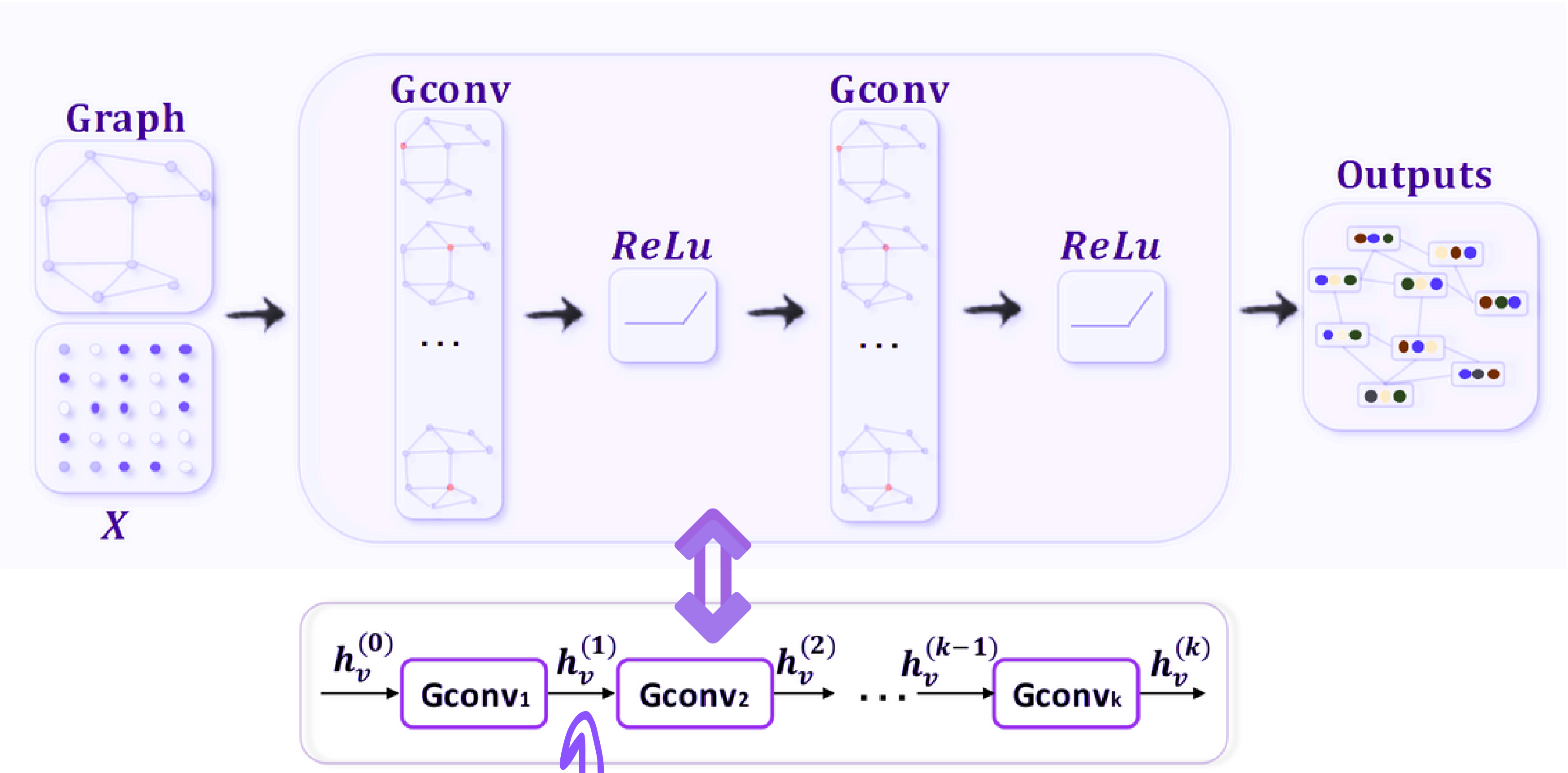


- Every layer develops a node's hidden representation by **aggregating information from its neighbours,** which **updates the kinematic features using connected events**
- Our models: PyTorch's **GCNConv** and **GraphConv**

[Original GCN concept: arxiv:1609.02907](https://arxiv.org/abs/1609.02907)
[GNN survey: arxiv:1901.00596](https://arxiv.org/abs/1901.00596)

GNN: model-dependent search

Graph convolutional networks



$$H^{(l)} = \left\{ \underbrace{\begin{bmatrix} 1 & 0 & 1 & & \\ 0 & 1 & 1 & \cdots & \\ 1 & 1 & 1 & & \\ & \vdots & & \ddots & \\ & & & & 1 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_{11} & \cdots & x_{1h'} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nh'} \end{bmatrix}}_{H^{(l-1)}} \right\}$$

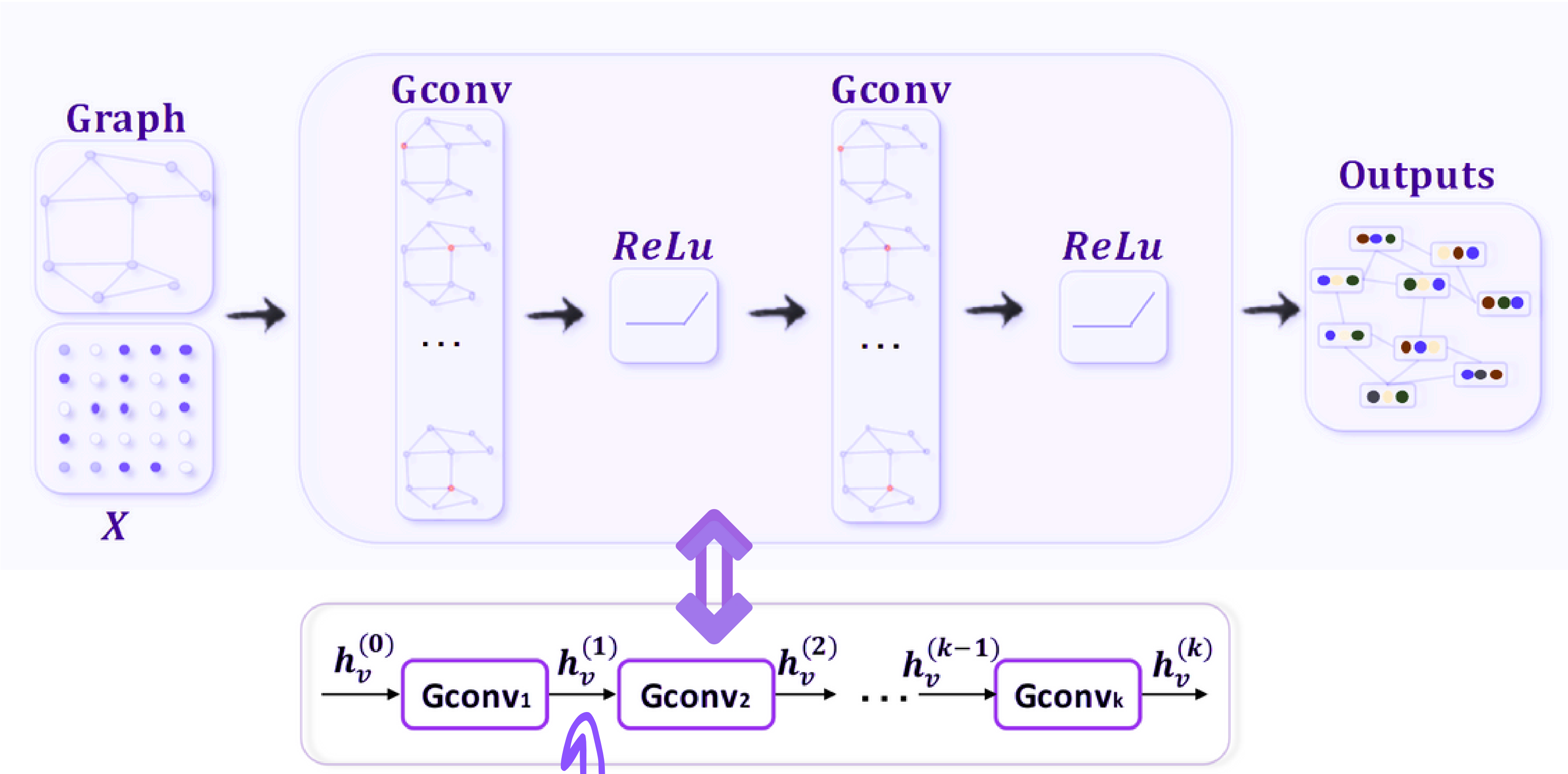
- Every layer develops a node's hidden representation by **aggregating information from its neighbours,** which **updates the kinematic features using connected events**
- Our models: PyTorch's **GCNConv** and **GraphConv**

[Original GCN concept: arxiv:1609.02907](https://arxiv.org/abs/1609.02907)

[GNN survey: arxiv:1901.00596](https://arxiv.org/abs/1901.00596)

GNN: model-dependent search

Graph convolutional networks



- Every layer develops a node's hidden representation by **aggregating information from its neighbours,** which **updates the kinematic features using connected events**
- Our models: PyTorch's **GCNConv** and **GraphConv**

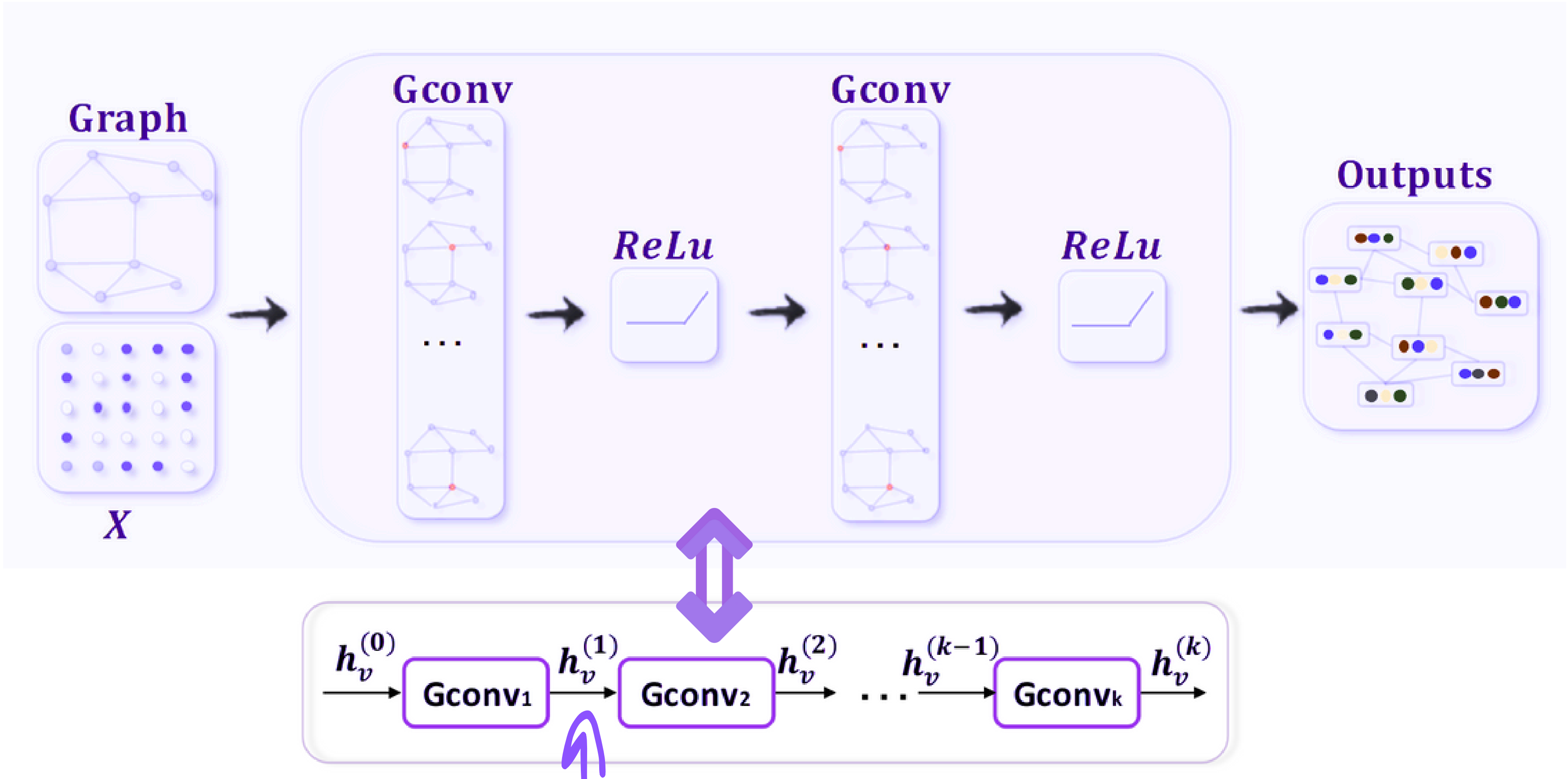
$$H^{(l)} = \left\{ \underbrace{\begin{bmatrix} 1 & 0 & 1 & & \\ 0 & 1 & 1 & \cdots & \\ 1 & 1 & 1 & & \\ & \vdots & & \ddots & \\ & & & & 1 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_{11} & \cdots & x_{1h'} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nh'} \end{bmatrix}}_{H^{(l-1)}} \right\} \underbrace{\begin{bmatrix} w_{11} & \cdots & w_{1h} \\ \vdots & \ddots & \cdots \\ w_{h'1} & \cdots & w_{h'h} \end{bmatrix}}_{W^{(l)}}$$

- **Weights on nodes** represent **effective yields** using MC event weights
- **Weights on edges** emphasise **local relationships:** multiply by inverse distance to value short-distance edges

[Original GCN concept: arxiv:1609.02907]
[GNN survey: arxiv:1901.00596]

GNN: model-dependent search

Graph convolutional networks



- Every layer develops a node's hidden representation by **aggregating information from its neighbours,** which **updates the kinematic features using connected events**
- Our models: PyTorch's **GCNConv** and **GraphConv**

$$H^{(l)} = \left\{ \underbrace{\begin{bmatrix} 1 & 0 & 1 & & \\ 0 & 1 & 1 & \cdots & \\ 1 & 1 & 1 & & \\ & \vdots & & \ddots & \\ & & & & 1 \end{bmatrix}}_{\tilde{A}} \underbrace{\begin{bmatrix} x_{11} & \cdots & x_{1h'} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nh'} \end{bmatrix}}_{H^{(l-1)}} \right\} \underbrace{\begin{bmatrix} w_{11} & \cdots & w_{1h} \\ \vdots & \ddots & \cdots \\ w_{h'1} & \cdots & w_{h'h} \end{bmatrix}}_{W^{(l)}}$$

- **Weights on nodes** represent **effective yields** using MC event weights
- **Weights on edges** emphasise **local relationships:** multiply by inverse distance to value short-distance edges

Each convolution: $H^{(l+1)} = \sigma\left(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}H^{(l)}W^{(l)}\right)$

for degree matrix normalisation with $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$

[Original GCN concept: arxiv:1609.02907]

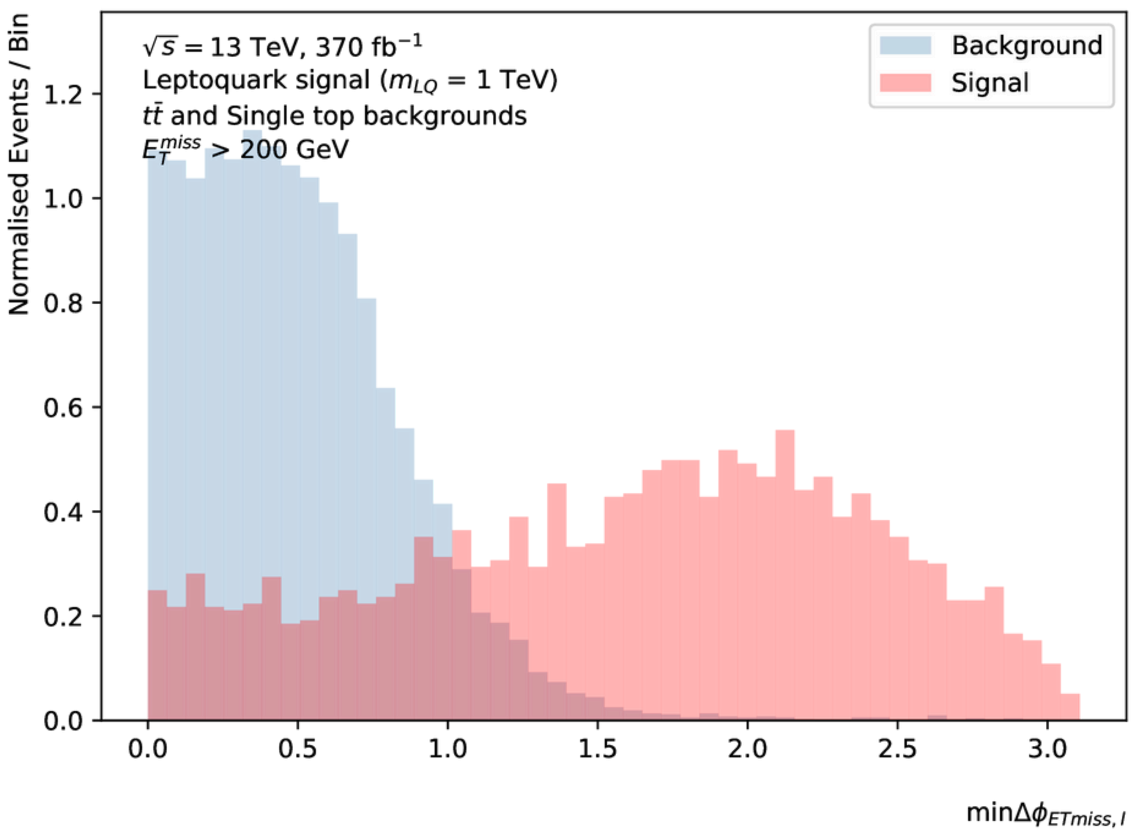
[GNN survey: arxiv:1901.00596]

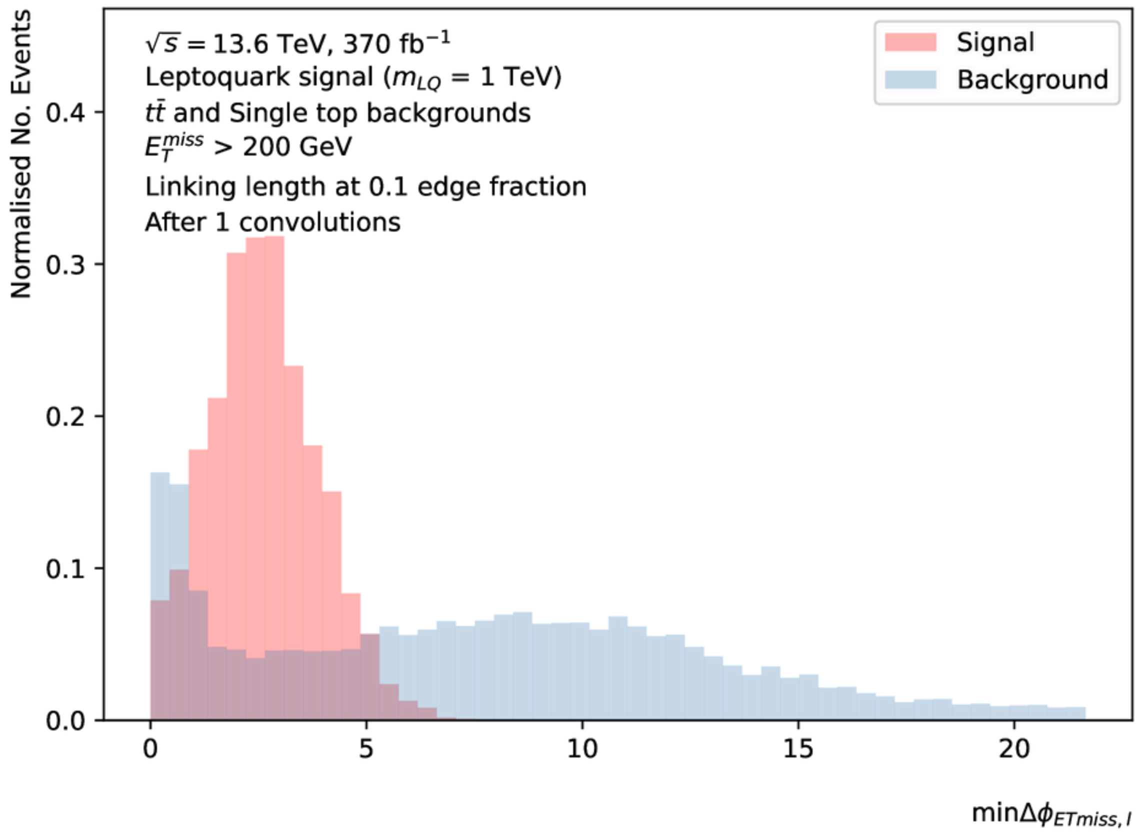
GNN: model-dependent searches

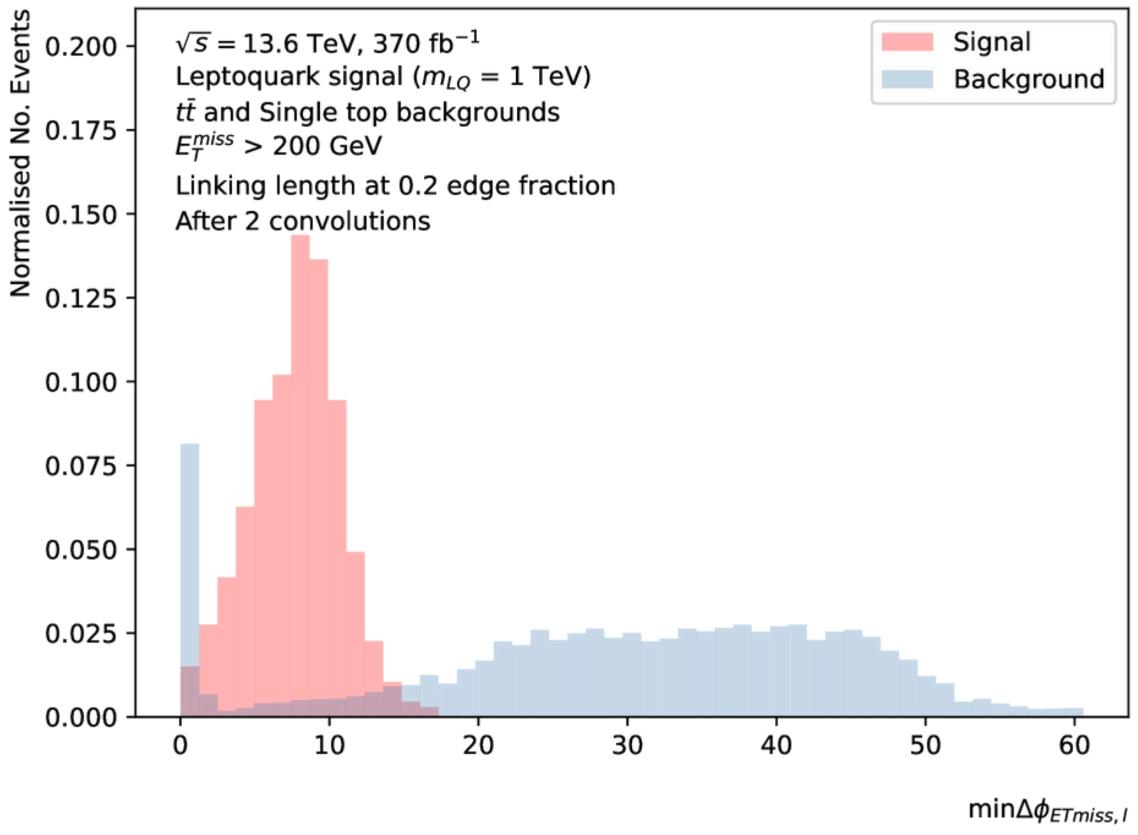
Convolutions

- More GCN hidden layers **receive messages from deeper into graph**
  → the final node representation is informed by messages from a further neighbourhood
  → in theory, more discriminating

More convolutions





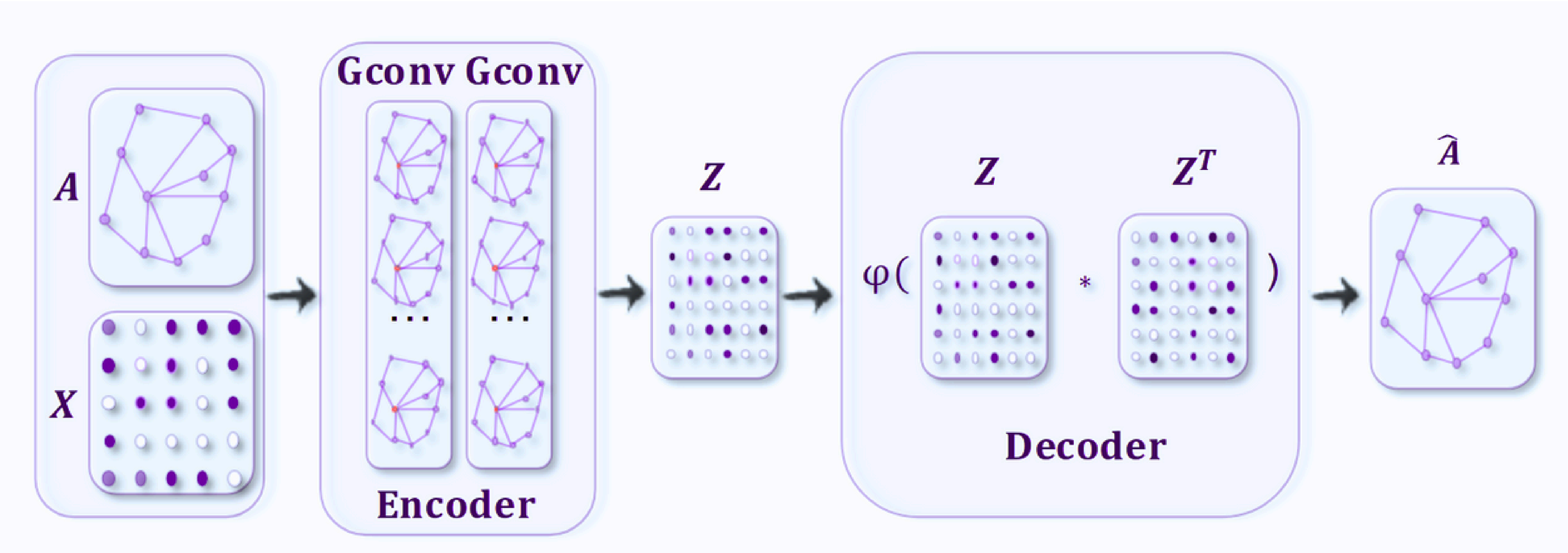


GNN: anomaly detection

→ How can a similar graph construction contribute to AD strategies?

Seek deviations from normal patterns/topologies in high-dimensional data, e.g. rare outlier events, unusual clusters

GAE



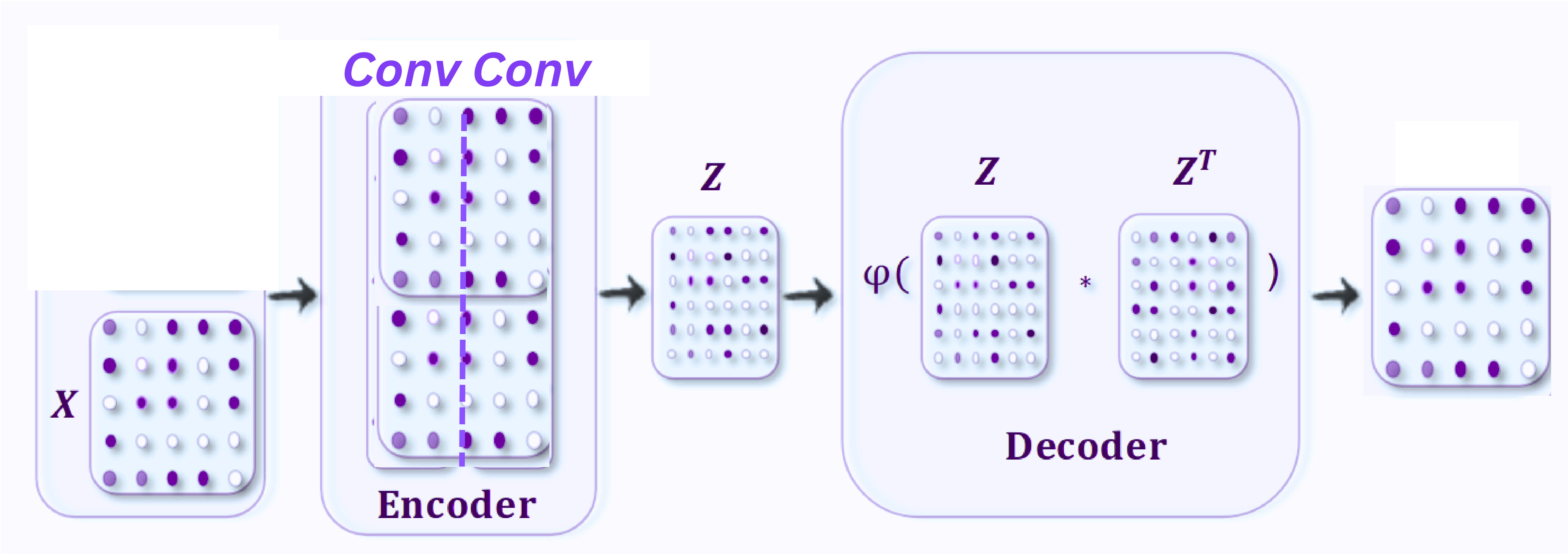
- **Encoder**: graph convolutional layers obtain network embedding for each node
- **Decoder**: computes pair-wise distances given network embeddings and reconstructs the graph structure (adjacency matrix)
- **Training**: learns latent node representations that minimise differences between real vs reconstructed adjacency matrices

GNN: anomaly detection

How can a similar graph construction contribute to AD strategies?

Seek deviations from normal patterns/topologies in high-dimensional data, e.g. rare outlier events, unusual clusters

AE



3. Results

Model-dependent search results

Convolution model	Distance metric	Graph domain	AUC (validation)
High-level kinematic input variables			
DNN			0.956
GCN	Euclidean	High-level kinematic space	0.951
GCN	Cosine	High-level kinematic space	0.954
GraphConv	Euclidean	High-level kinematic space	0.943
GraphConv	Cosine	High-level kinematic space	0.952
GCN	EMD	Low-level kinematic space	0.954
GraphConv	EMD	Low-level kinematic space	**0.972**
Low-level kinematic input variables			
DNN			0.919
GCN	Euclidean	Low-level kinematic space	0.826
GCN	Cosine	Low-level kinematic space	0.852
GCN	EMD	Low-level kinematic space	0.901
GraphConv	Euclidean	Low-level kinematic space	0.812
GraphConv	Cosine	Low-level kinematic space	0.804
GraphConv	EMD	Low-level kinematic space	**0.951**
GCN	Euclidean	Latent space	0.892
GCN	Cosine	Latent space	0.901
GraphConv	Euclidean	Latent space	0.892
GraphConv	Cosine	Latent space	0.873

Best area under curve from **GraphConv** layers with distance=**EMD** where the graph is built in a **space of low-level** kinematics

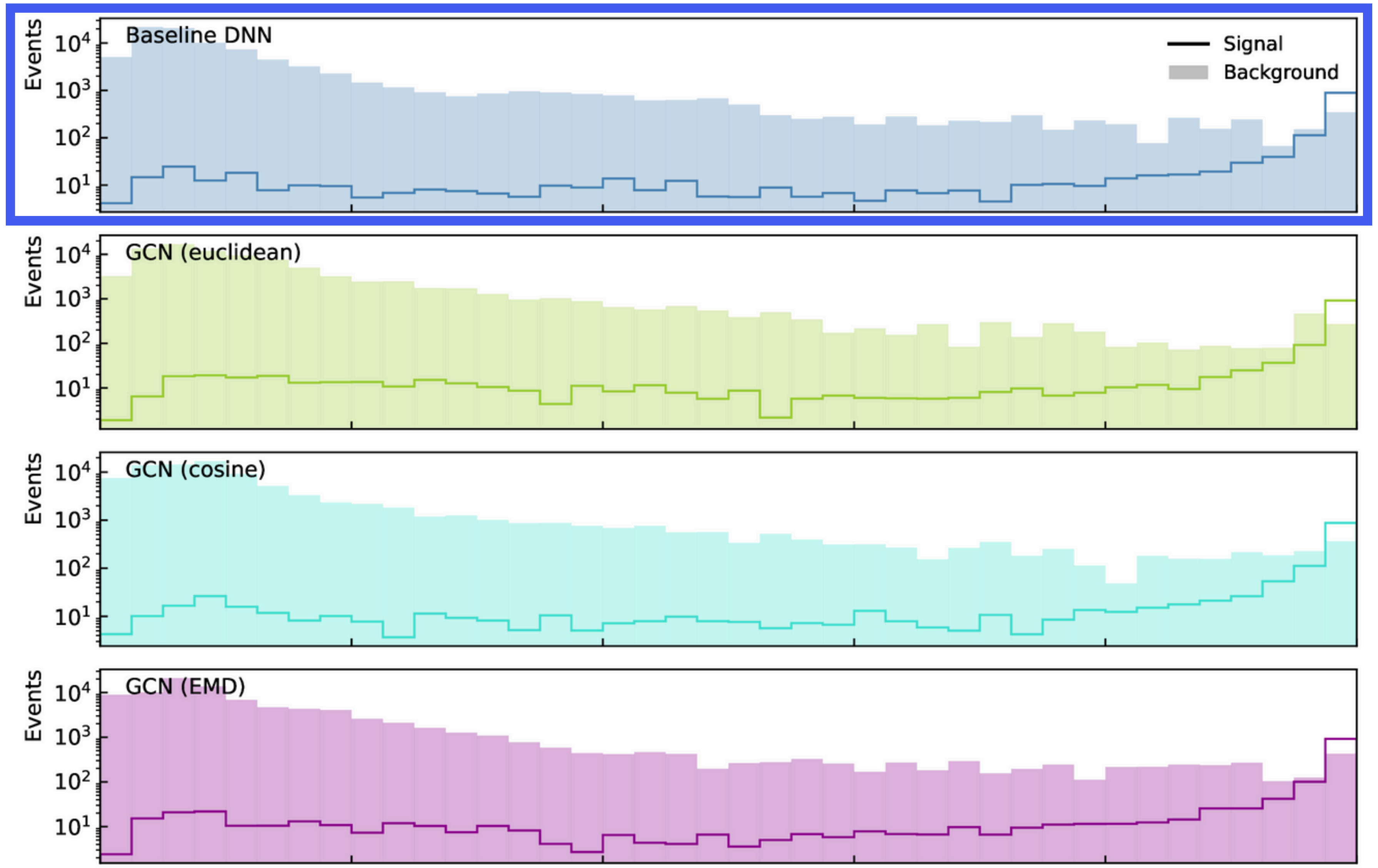
Winning hyperparameters:

Convolution model	Distance metric	Graph domain	GNN layers	Edge fraction	Neighbours sampled [nodes, layers]	Dropout
DNN						0.05
GraphConv	EMD	Low-level kinematic space	[12, 12]	0.2	[60, 6]	0.0
DNN						0.1
GraphConv	EMD	Low-level kinematic space	[12, 12]	0.1	[60, 6]	0.0

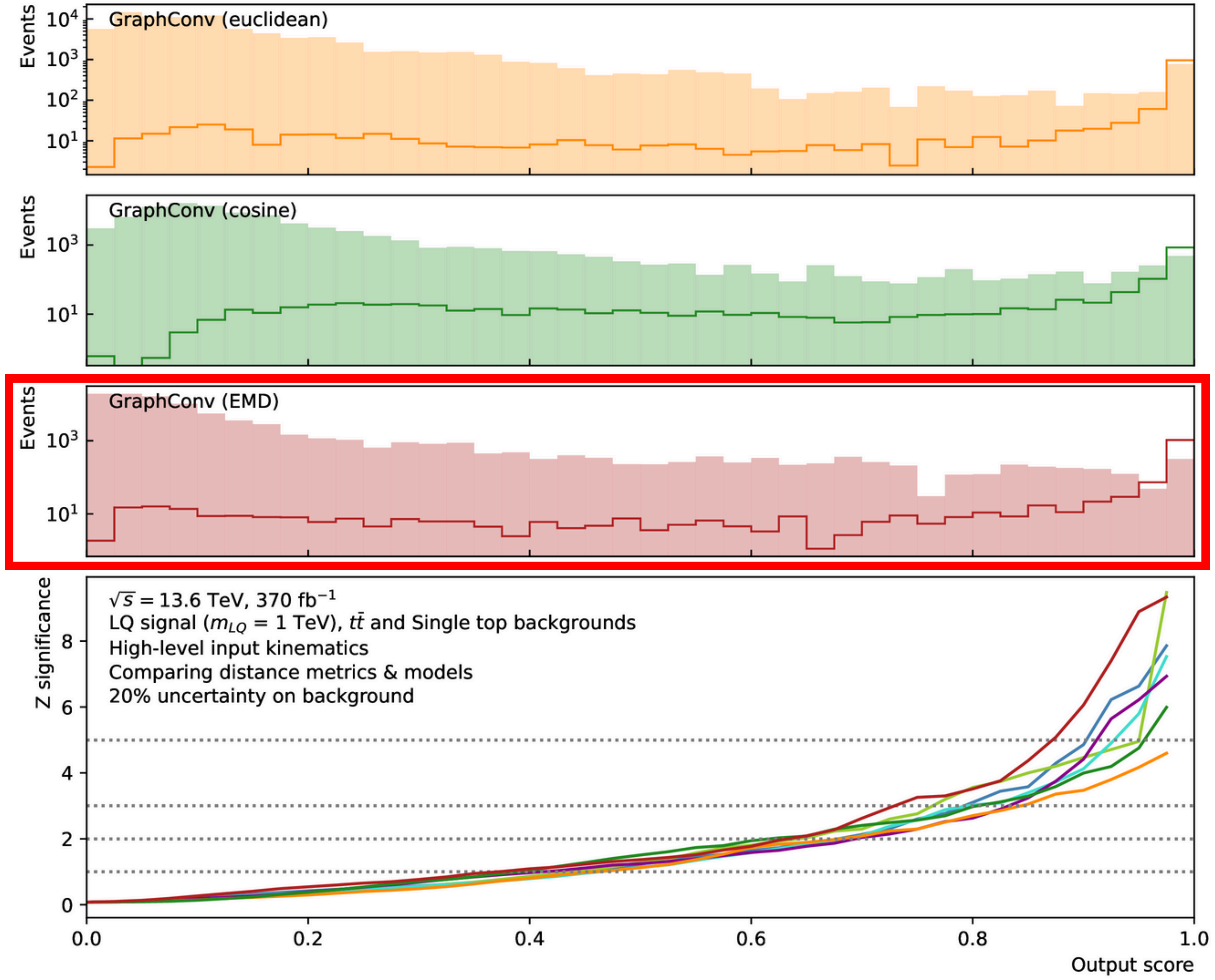
- Winning graph technique is consistent across swap of input features in training from low-level <--> high-level kinematics

Model-dependent search results

Z-score for DNN vs GNNs with low-level kinematic inputs



Convolution model	Distance metric	Graph domain	AUC (validation)
DNN			0.956
GraphConv	EMD	Low-level kinematic space	**0.972**



Z-score: lower-bound cut

Anomaly detection results

Preliminary

Parameter choices:

1. Calculate **Euclidean distance** between events in 5-dim kinematic space
2. Connect **closest 5%** of possible neighbours
3. Choose GAE model **5 layers deep** sampling **5 neighbours** each time
4. **Train AE and GAE** unsupervised: background-only samples (10000 events)
5. Inject **10% or 20% ‘anomalous’ leptoquark signal** into test samples (10000 events)
6. Evaluate with trained models

→ Can we identify the leptoquark signal as ‘anomalous’?

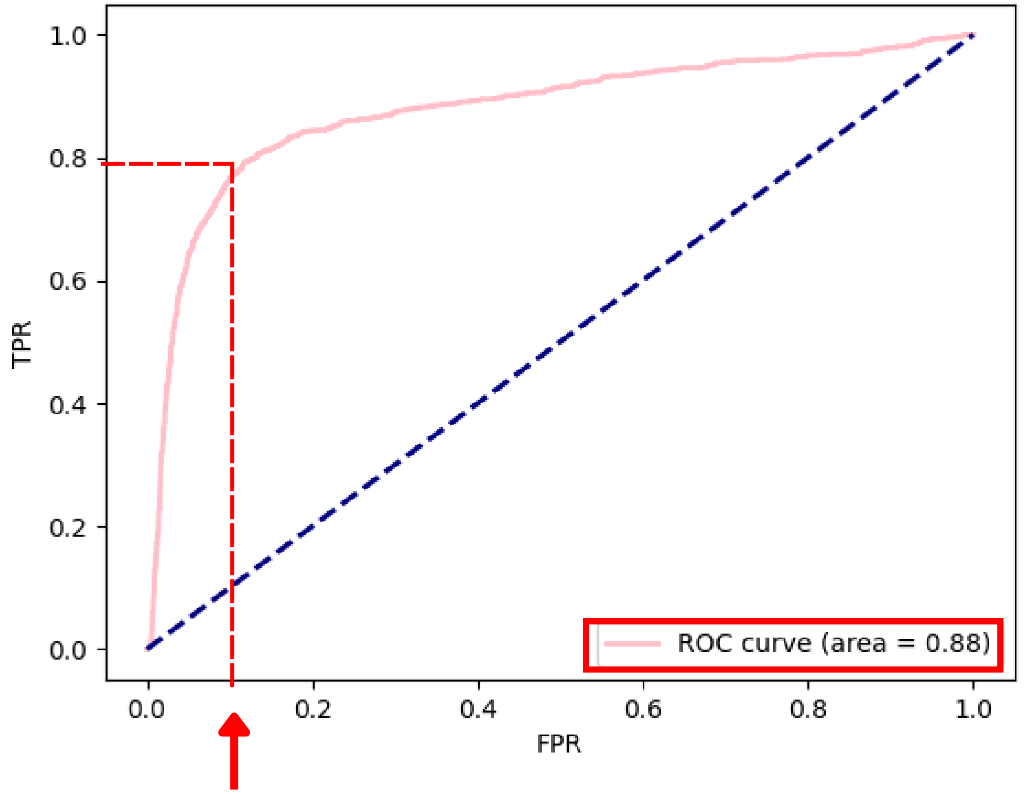
Anomaly detection results

Preliminary

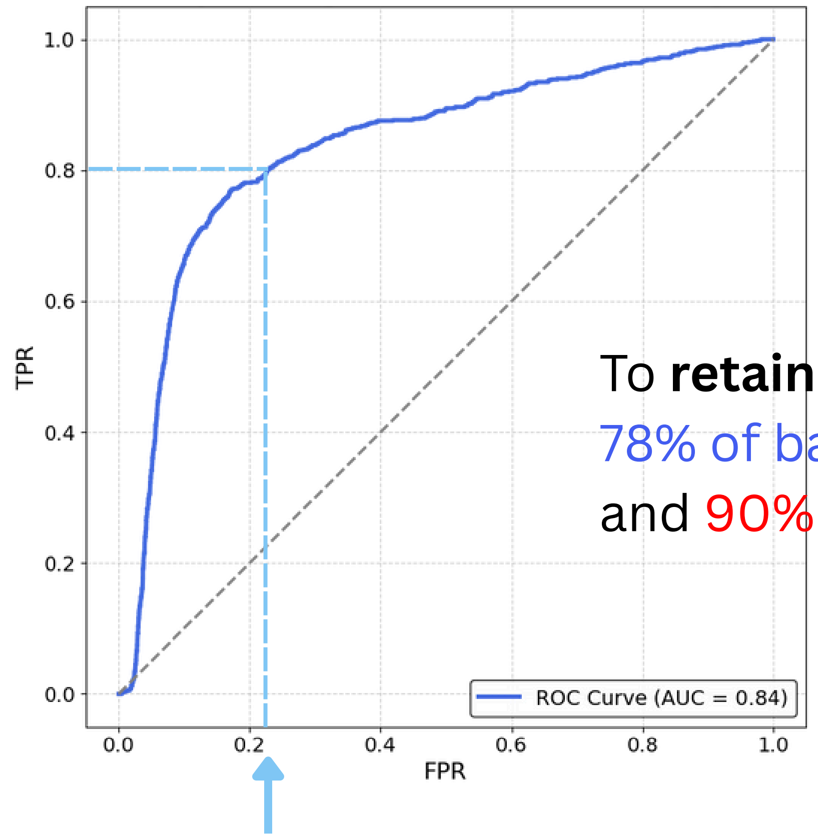
ROC

Test sample: 10% signal

GAE

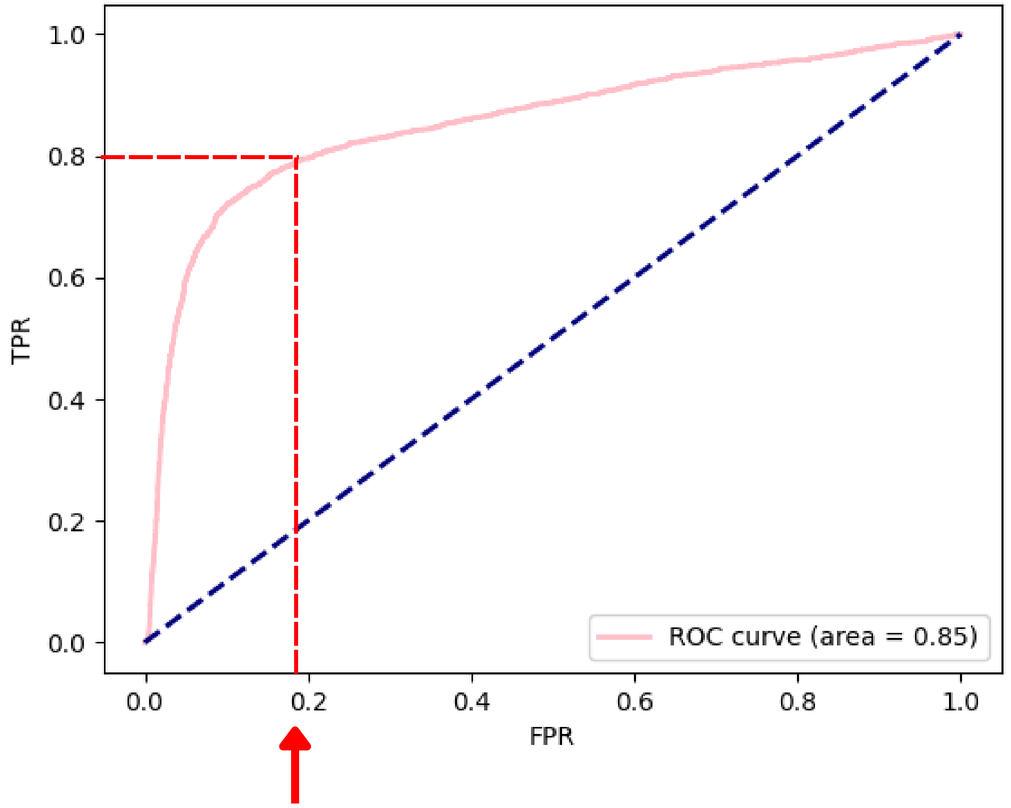


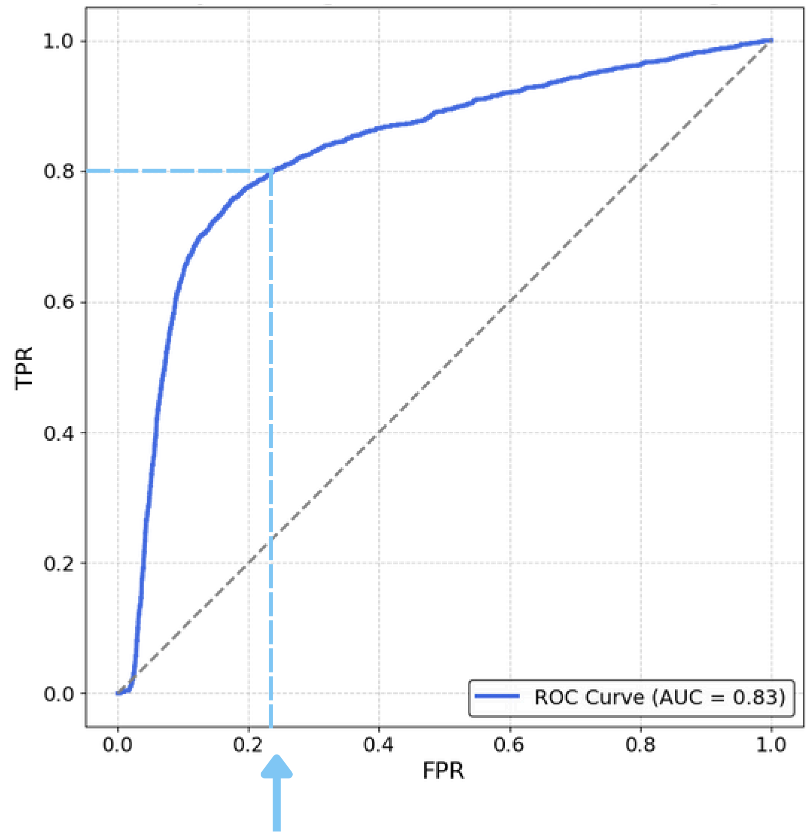
AE



To **retain 80% signal**, we reject 78% of background with the AE, and 90% with the GAE

20% signal





Anomaly detection results

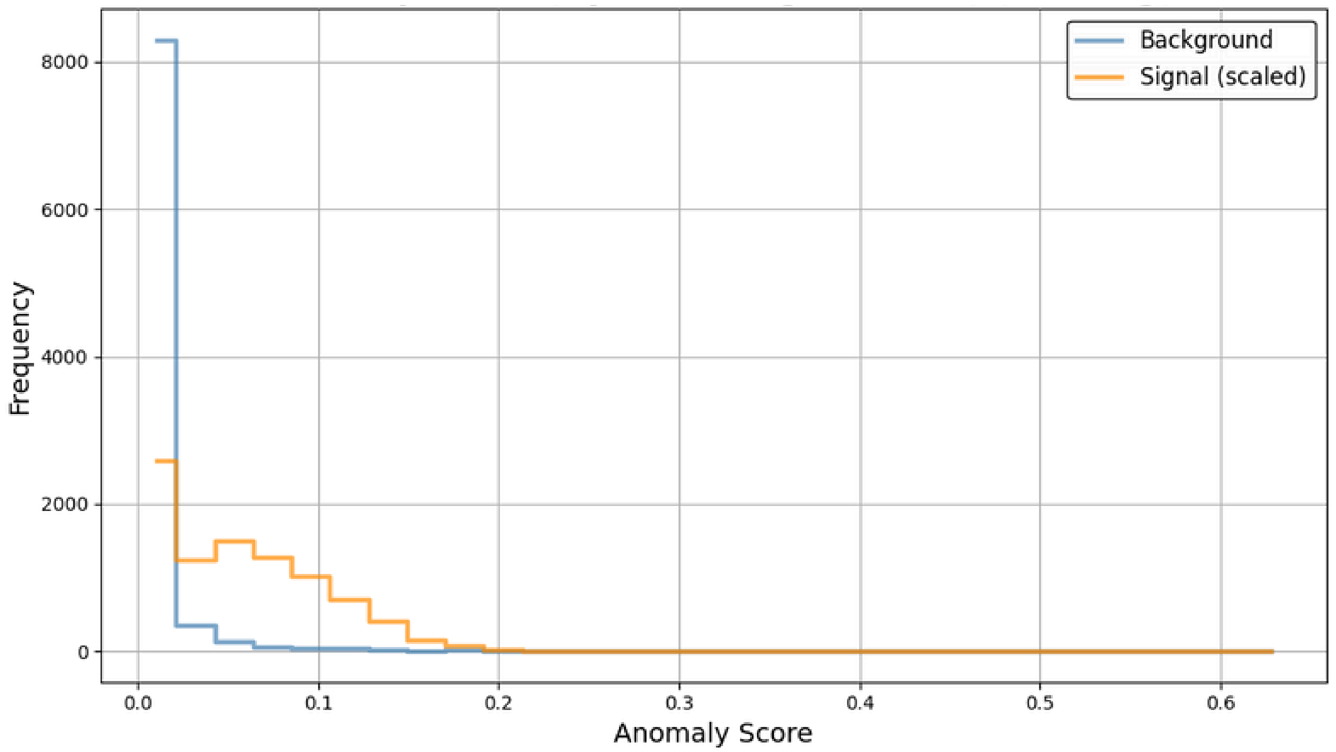
Preliminary

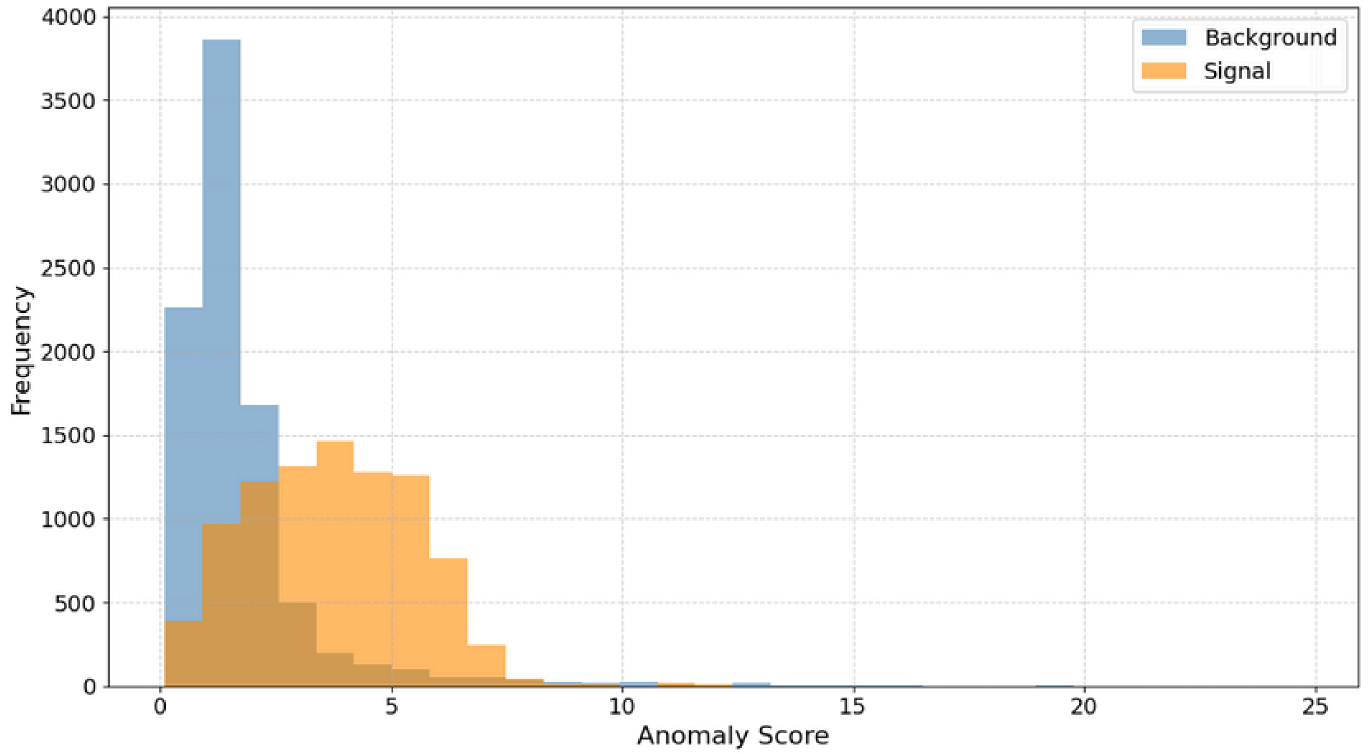
GAE

AE

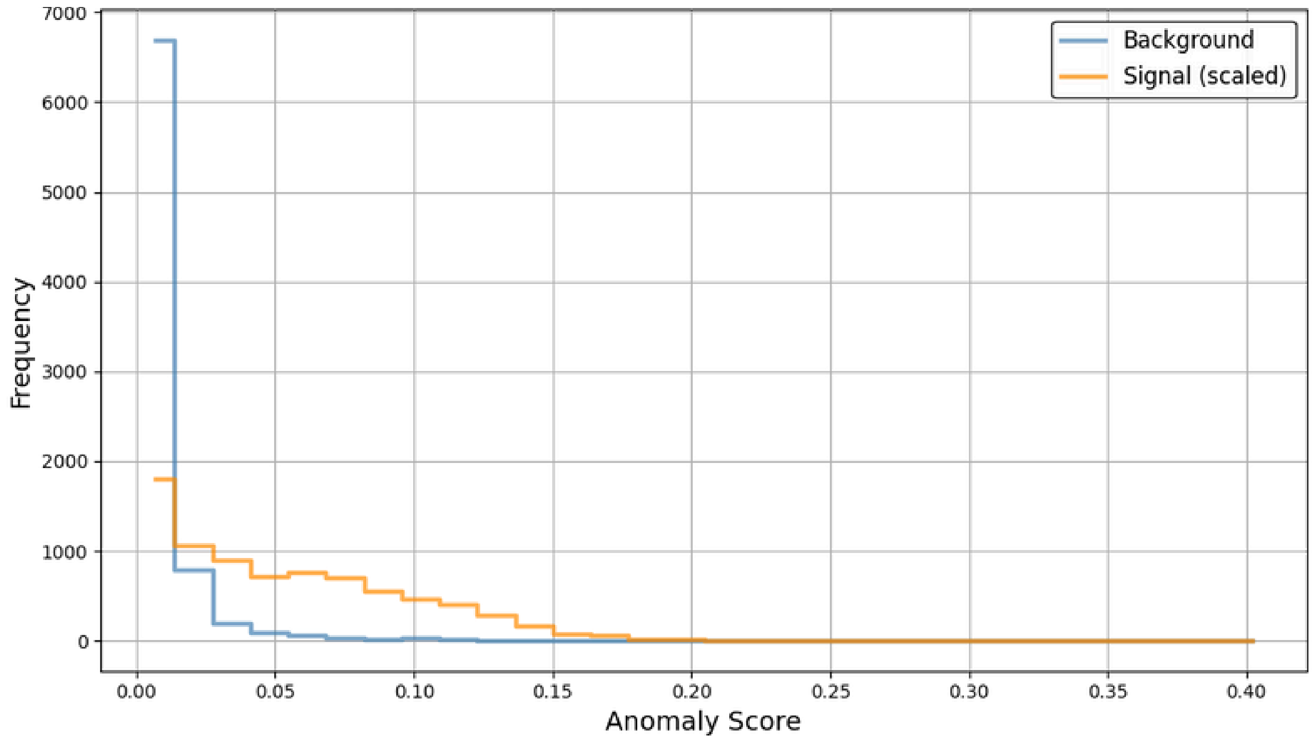
Anomaly scores

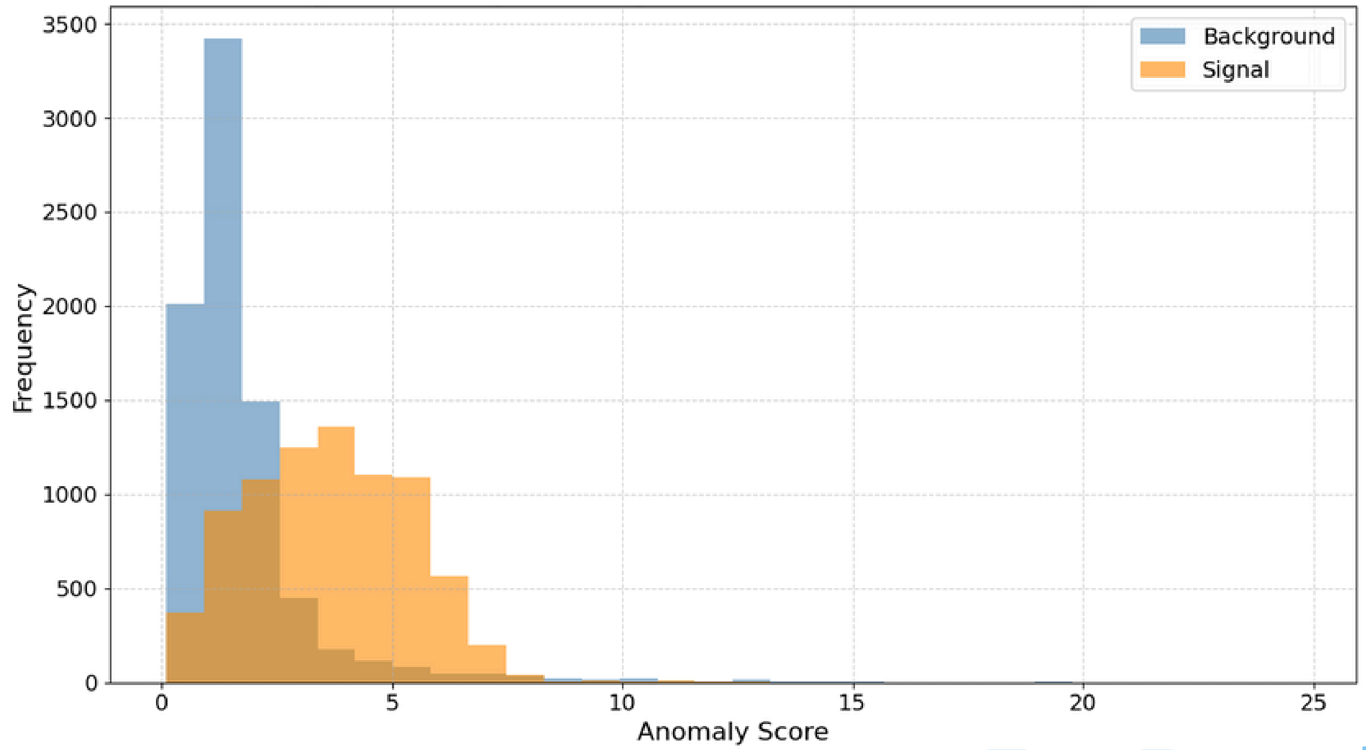
Test sample: 10% signal





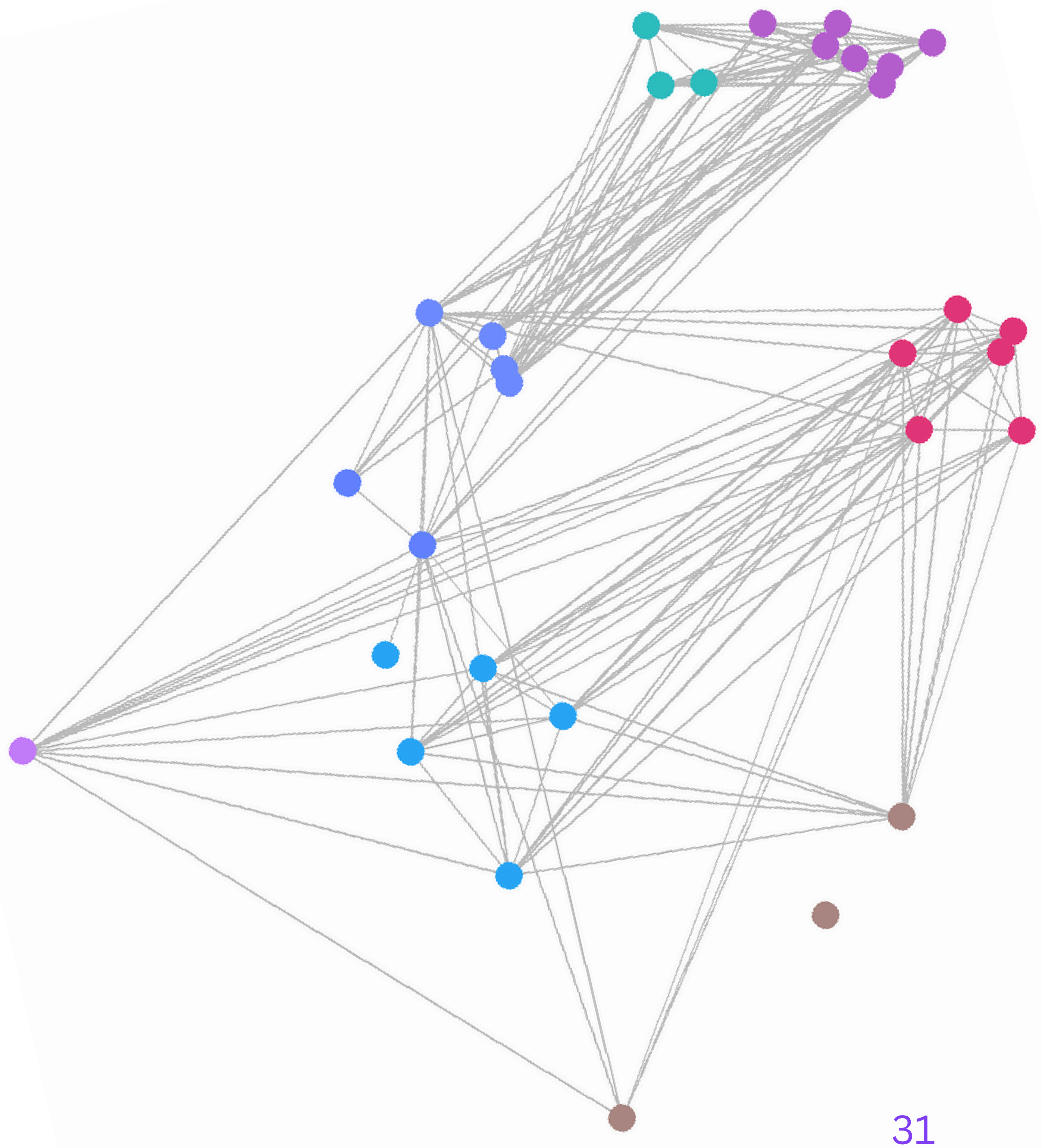
20% signal





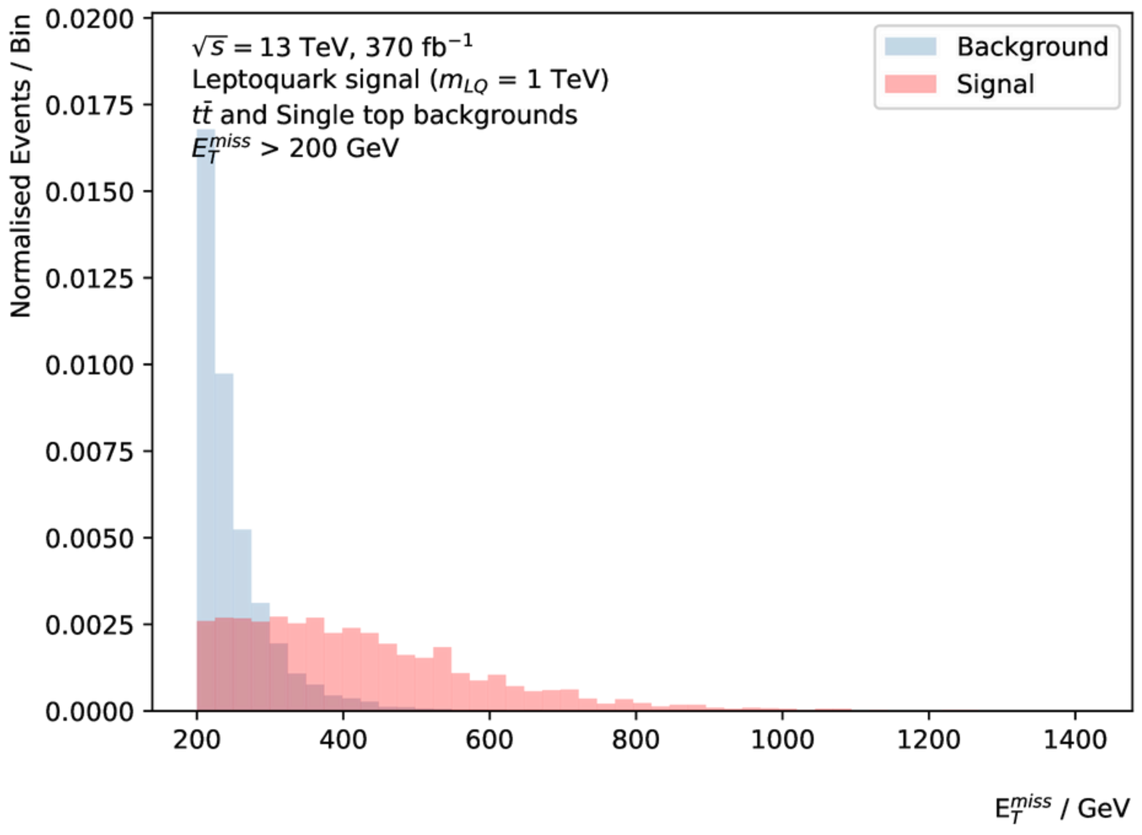
Conclusions

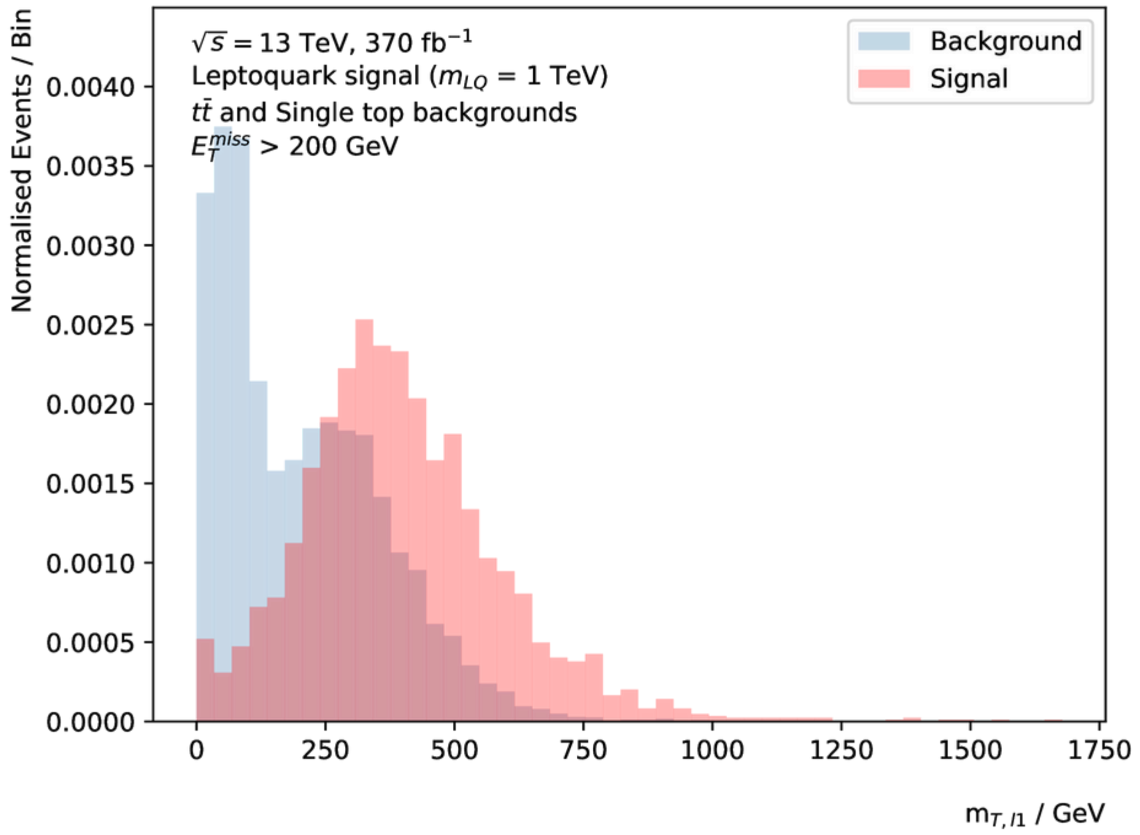
- Model dependent strategy: best performance from **GraphConv** (compared with other GNNs and the DNN)
- Anomaly detection: best performance so far from **GAE** (compared with AE)
- Ongoing work towards robust results with **larger graphs**

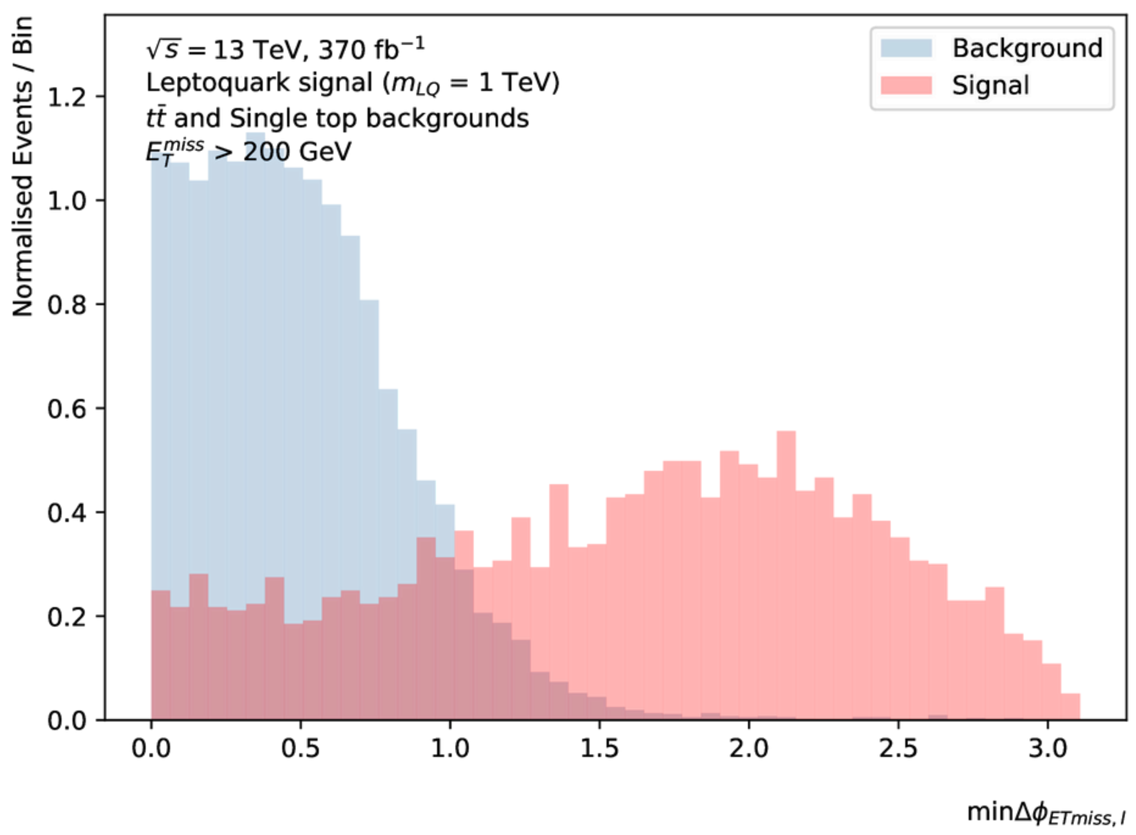


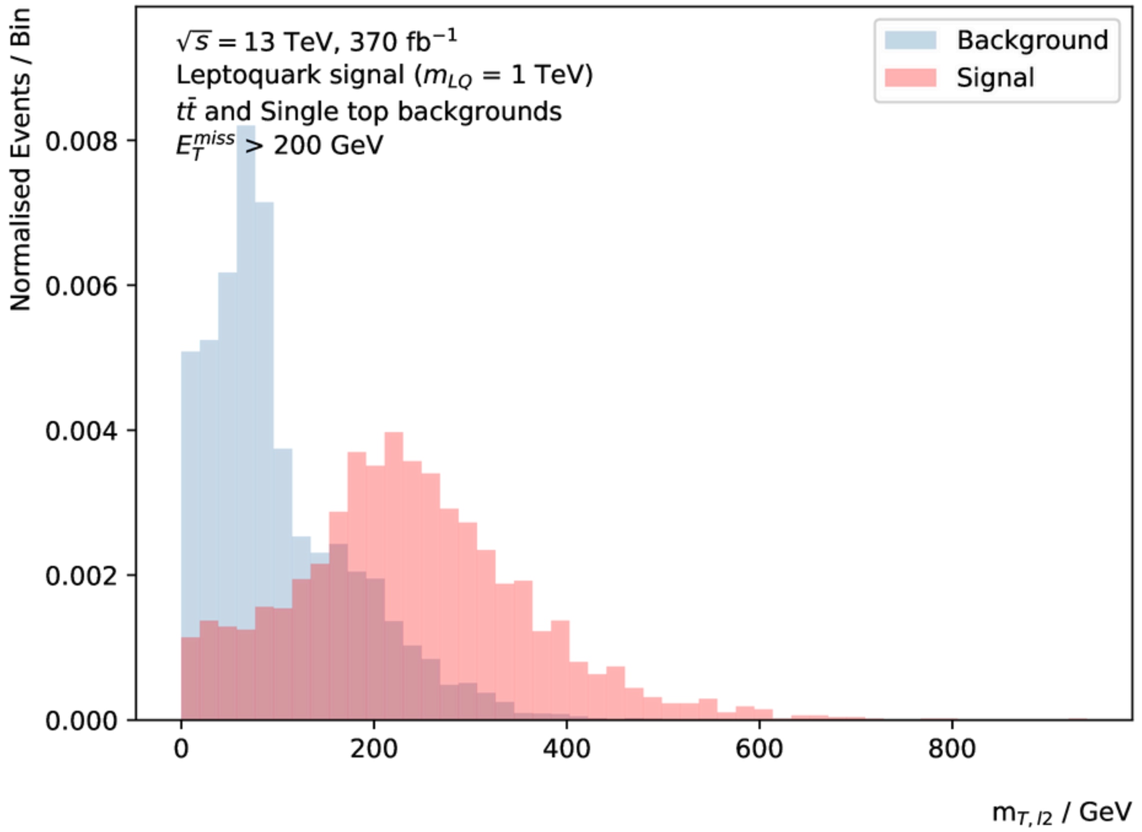
Backup

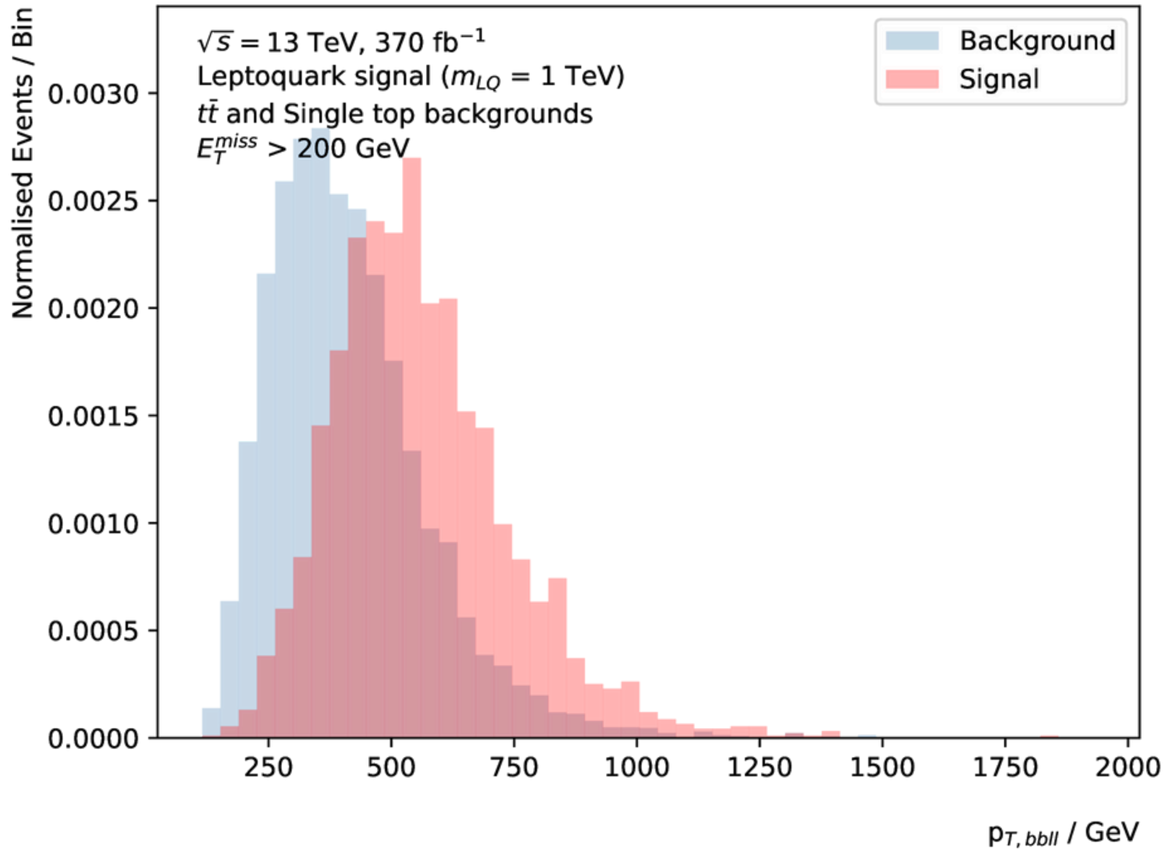
Kinematic distributions



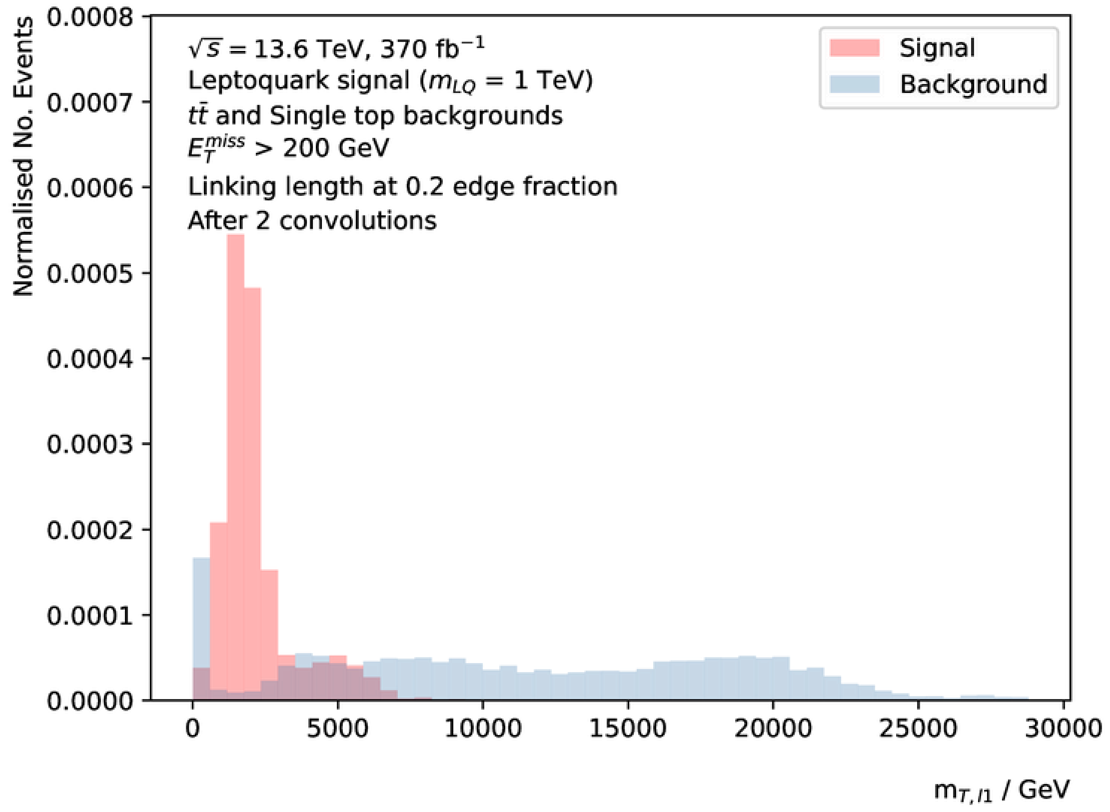


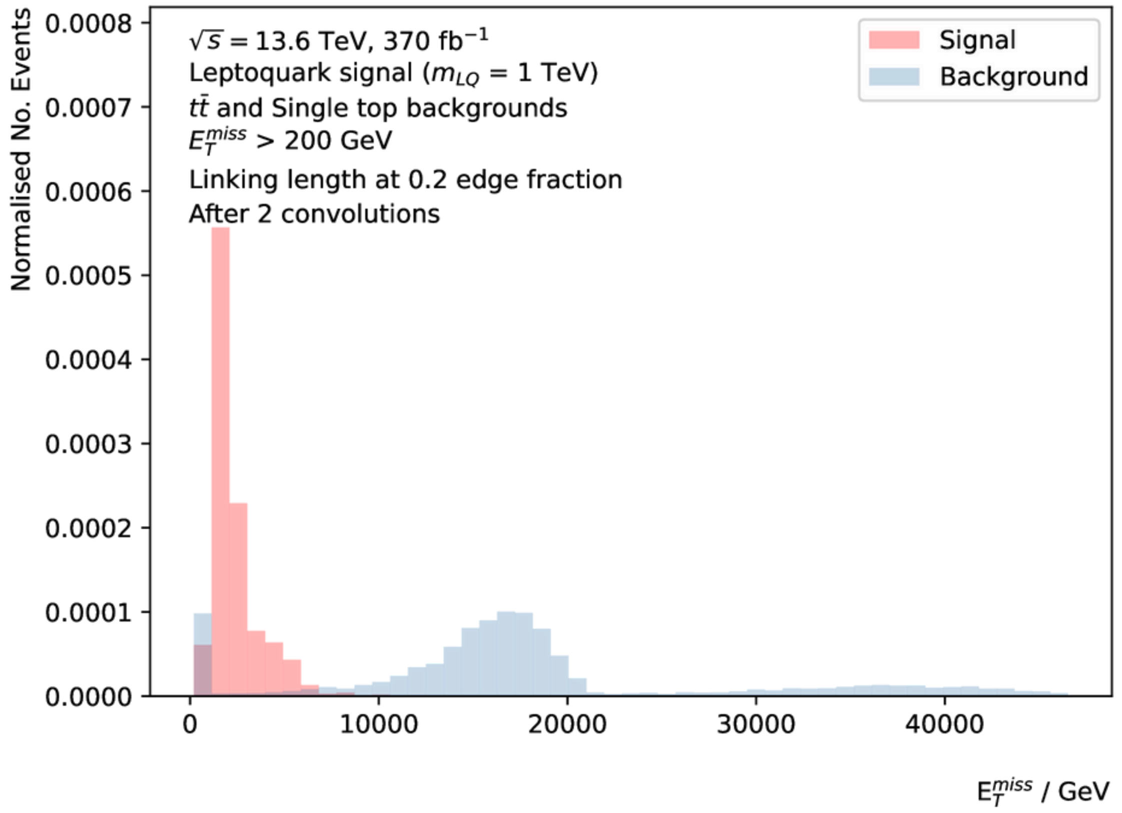


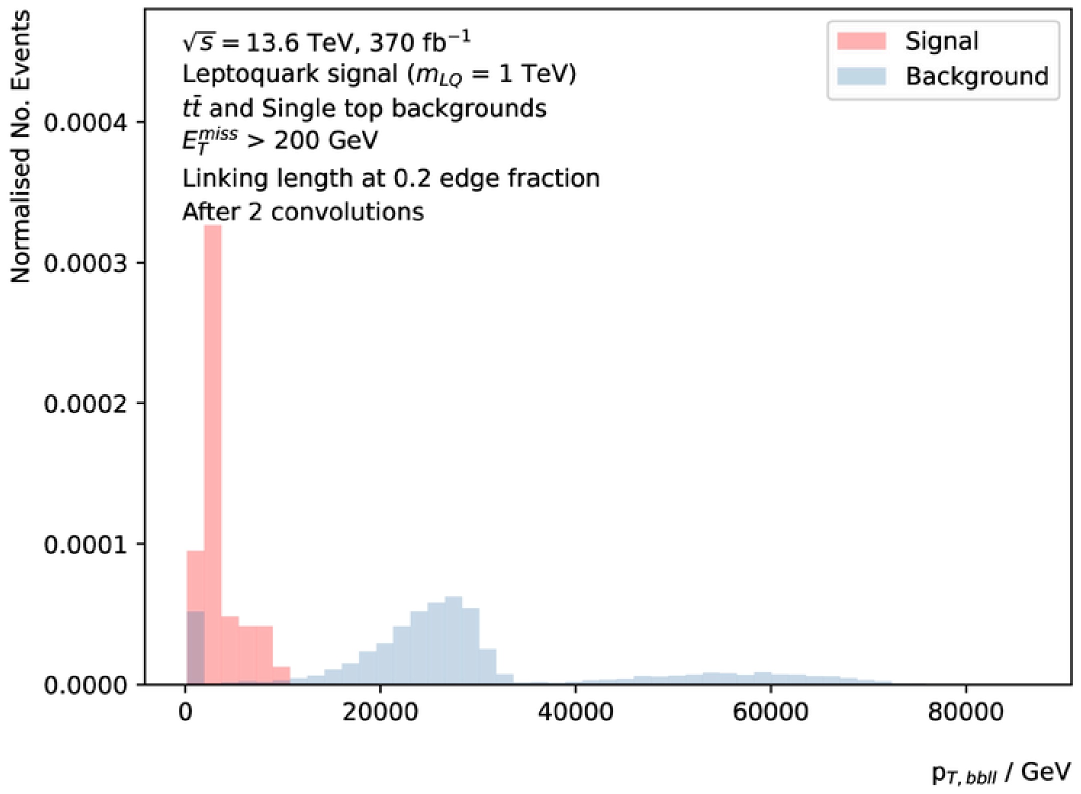


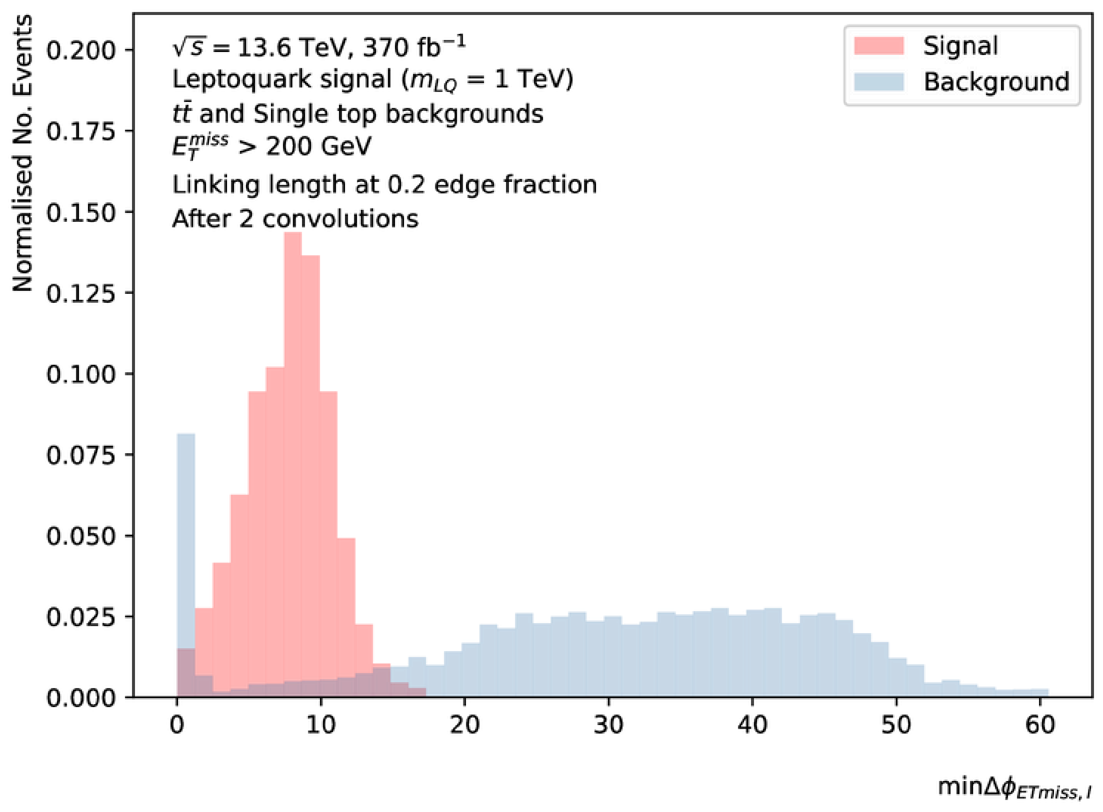


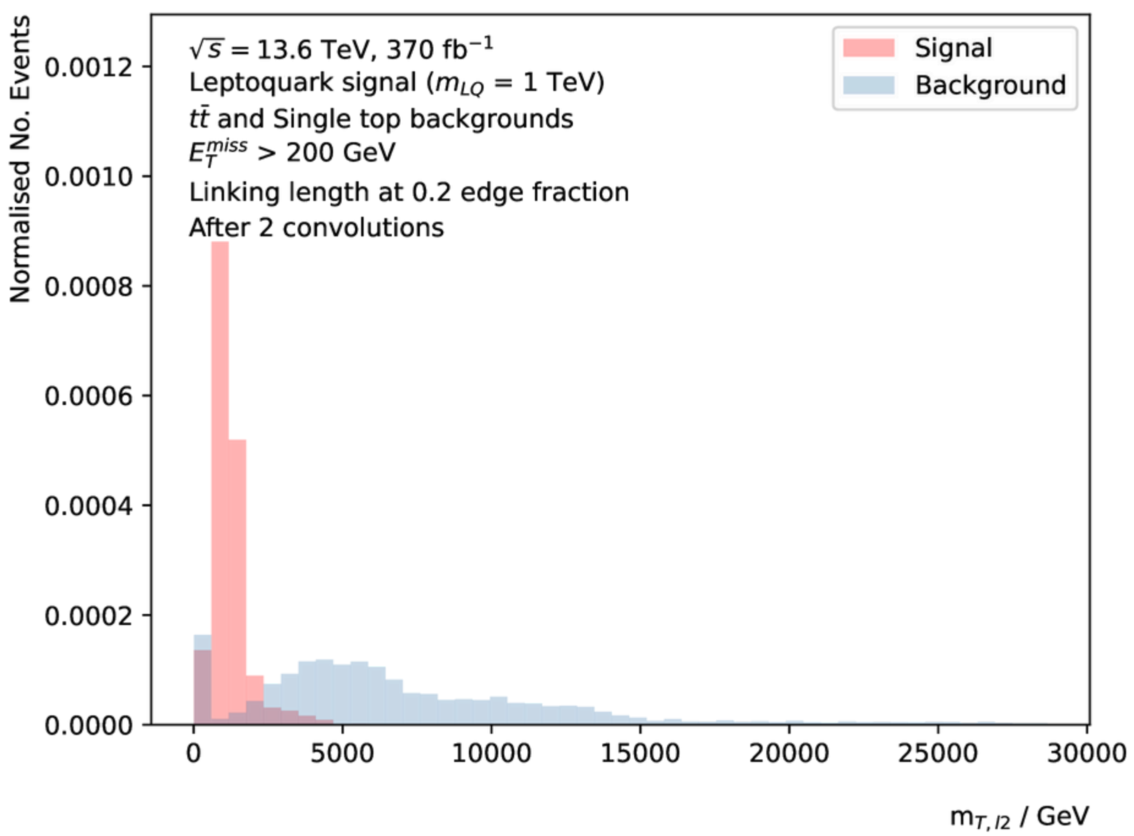
Twice-convoluted kinematic distributions











Node weighting

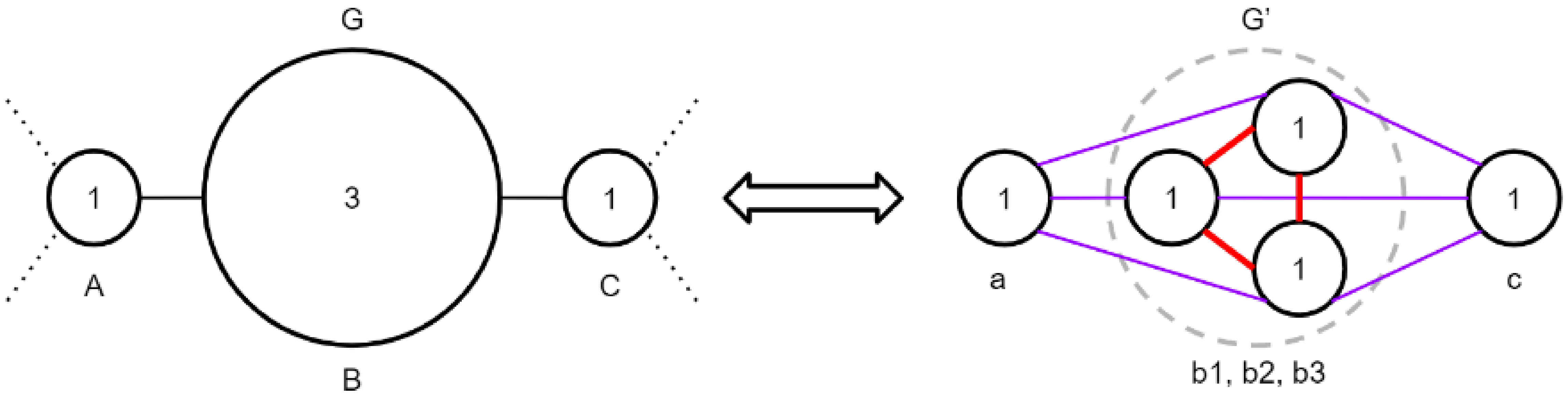


Figure 15: The principle of node splitting invariance means that a node-weighted graph G is equivalent to a refined graph G' where all nodes have been split into a number of unweighted nodes proportional to the weight. Here node B is split into b1, b2 and b3. The nodes b1, b2 and b3 are assumed to have i) full internal connectivity (red links) and ii) identical external connectivity (purple links).

Matrix dimensions

$$H^{(l)} = \left\{ \overset{\tilde{A}}{\begin{bmatrix} 1 & 0 & 1 & & \\ 0 & 1 & 1 & \cdots & \\ 1 & 1 & 1 & & \\ & \vdots & & \ddots & \\ & & & & 1 \end{bmatrix}} \overset{H^{(l-1)}}{\begin{bmatrix} x_{11} & \cdots & x_{1h'} \\ \vdots & \ddots & \vdots \\ x_{n1} & \cdots & x_{nh'} \end{bmatrix}} \right\} \overset{W^{(l)}}{\begin{bmatrix} w_{11} & \cdots & w_{1h} \\ \vdots & \ddots & \cdots \\ w_{h'1} & \cdots & w_{h'h} \end{bmatrix}} + \overset{b^{(l)}}{\begin{bmatrix} b_{11} & \cdots & b_{1h} \\ \vdots & \ddots & \vdots \\ b_{k1} & \cdots & b_{kh} \end{bmatrix}}$$

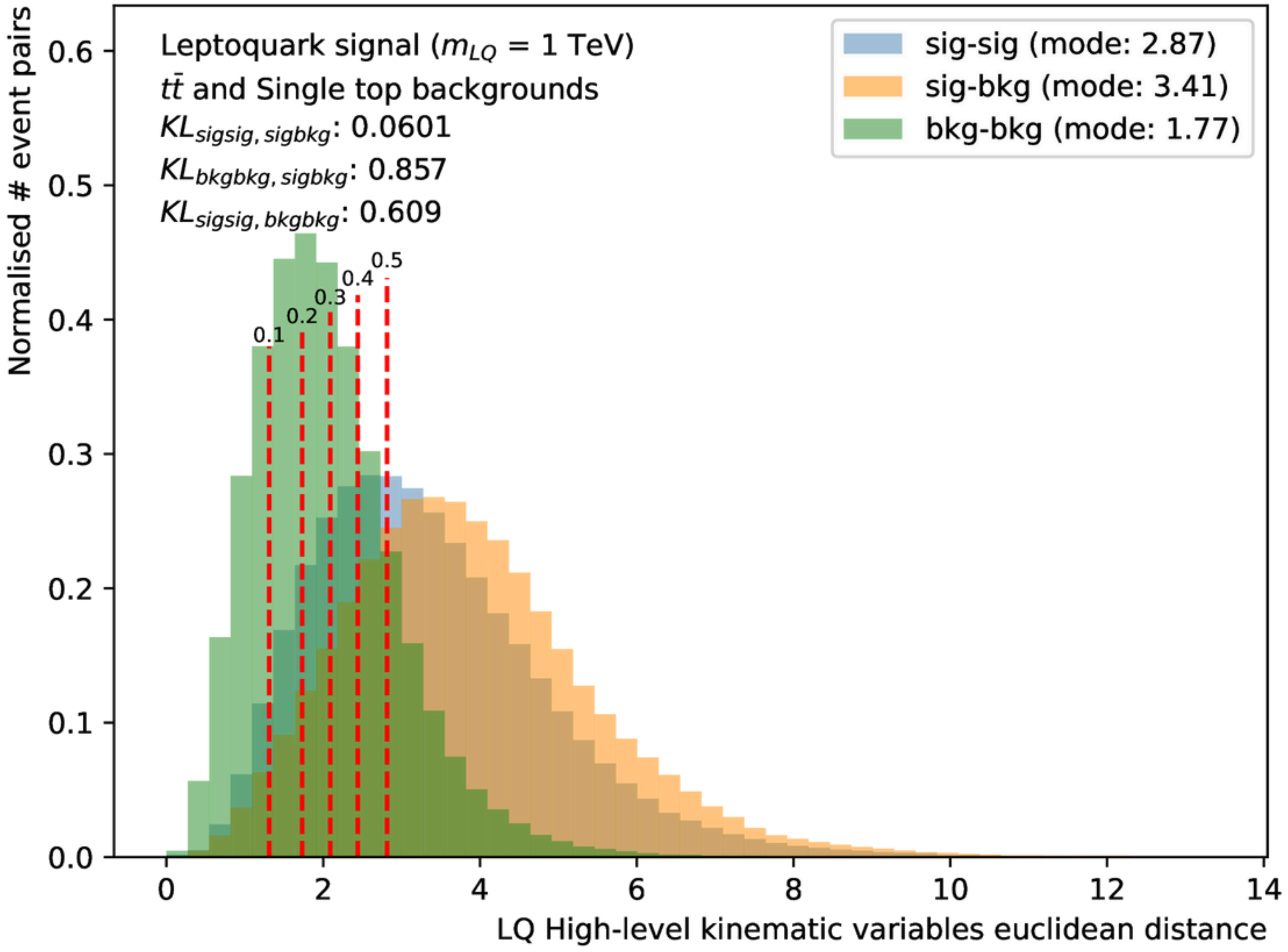
$$\underbrace{\{[n \times n] \quad \times \quad [n \times h']\}}_{[n \times h']} \quad \times \quad [h' \times h] \quad + \quad [n \times h]$$

$$\underbrace{[n \times h'] \quad \times \quad [h' \times h]}_{[n \times h]} \quad + \quad [n \times h]$$

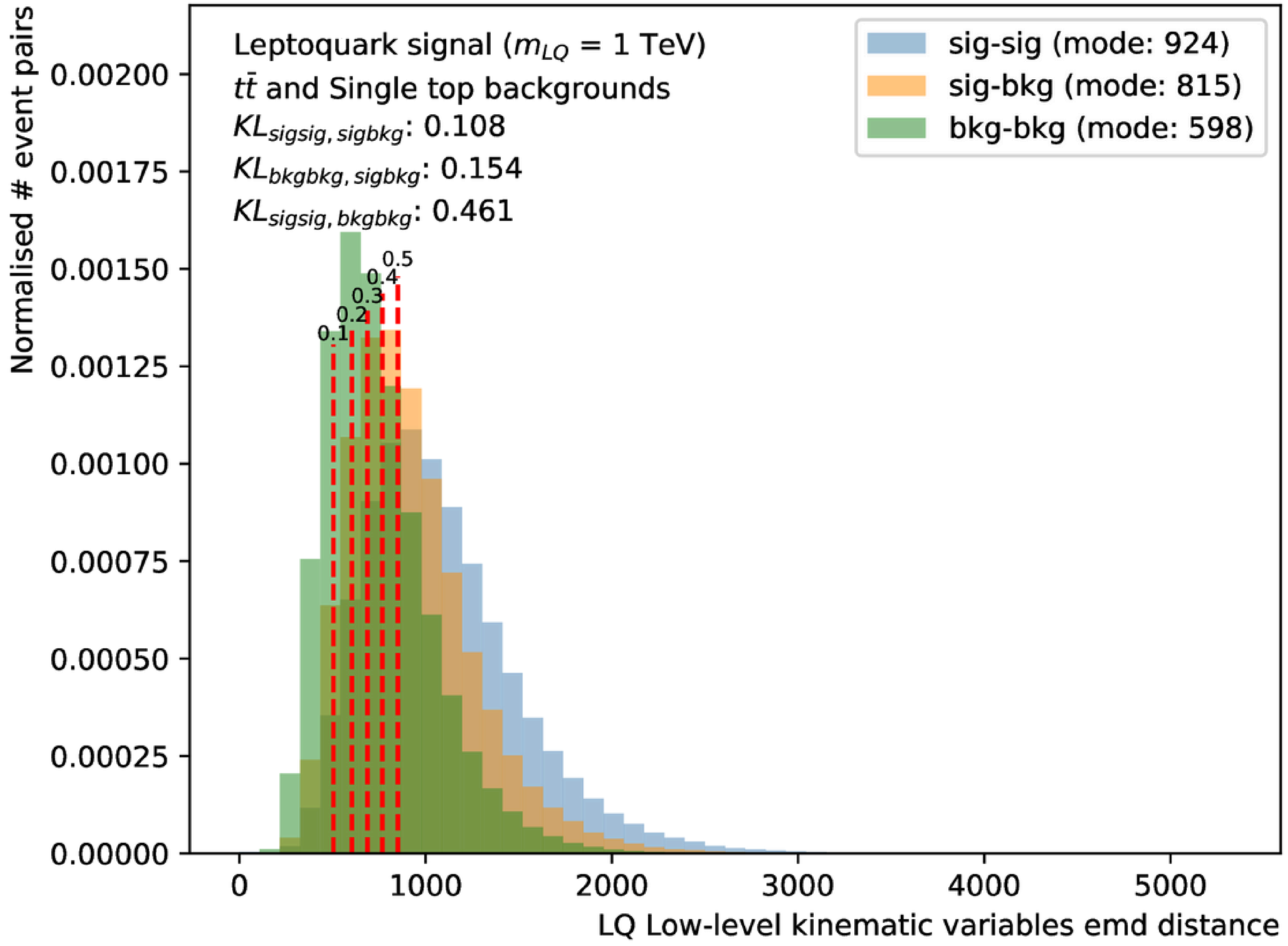
$$\underbrace{[n \times h] \quad + \quad [n \times h]}_{[n \times h]}$$

Distance distributions

Euclidean Distance



Earth Mover's Distance



Earth Mover's Distance

Computing the EMD is based on a solution to the well-known *transportation problem* [1]. Suppose that several *suppliers*, each with a given amount of goods, are required to supply several *consumers*, each with a given limited capacity. For each supplier-consumer pair, the cost of transporting a single unit of goods is given. The transportation problem is then to find a least-expensive flow of goods from the suppliers to the consumers that satisfies the consumers' demand. Matching signatures can be naturally cast as a transportation problem by defining one signature as the supplier and the other as the consumer, and by setting the cost for a supplier-consumer pair to equal the ground distance between an element in the first signature and an element in the second. Intuitively, the solution is then the minimum amount of ``work'' required to transform one signature into the other.

This can be formalized as the following linear programming problem: Let $P = \{(p_1, w_{p_1}), \ldots, (p_m, w_{p_m})\}$ be the first signature with m clusters, where p_i is the cluster representative and w_{p_i} is the weight of the cluster; $Q = \{(q_1, w_{q_1}), \ldots, (q_n, w_{q_n})\}$ the second signature with n clusters; and $\mathbf{D} = [d_{ij}]$ the ground distance matrix where d_{ij} is the ground distance between clusters p_i and q_j.

We want to find a flow $\mathbf{F} = [f_{ij}]$, with f_{ij} the flow between p_i and q_j, that minimizes the overall cost

$$\mathrm{WORK}(P, Q, \mathbf{F}) = \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij} ,$$

subject to the following constraints:

$$\begin{aligned} f_{ij} &\geq 0 & & 1 \leq i \leq m,\ 1 \leq j \leq n \\ \sum_{j=1}^{n} f_{ij} &\leq w_{p_i} & & 1 \leq i \leq m \\ \sum_{i=1}^{m} f_{ij} &\leq w_{q_j} & & 1 \leq j \leq n \\ \sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} &= \min\left(\sum_{i=1}^{m} w_{p_i}, \sum_{j=1}^{n} w_{q_j}\right) , \end{aligned}$$

The first constraint allows moving ``supplies'' from P to Q and not vice versa. The next two constraints limits the amount of supplies that can be sent by the clusters in P to their weights, and the clusters in Q to receive no more supplies than their weights; and the last constraint forces to move the maximum amount of supplies possible. We call this amount the *total flow*. Once the transportation problem is solved, and we have found the optimal flow $\mathbf{F}$, the earth mover's distance is defined as the work normalized by the total flow:

$$\mathrm{EMD}(P, Q) = \frac{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij} d_{ij}}{\sum_{i=1}^{m} \sum_{j=1}^{n} f_{ij}} .$$

https://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/RUBNER/emd.htm

GNN: model-dependent searches

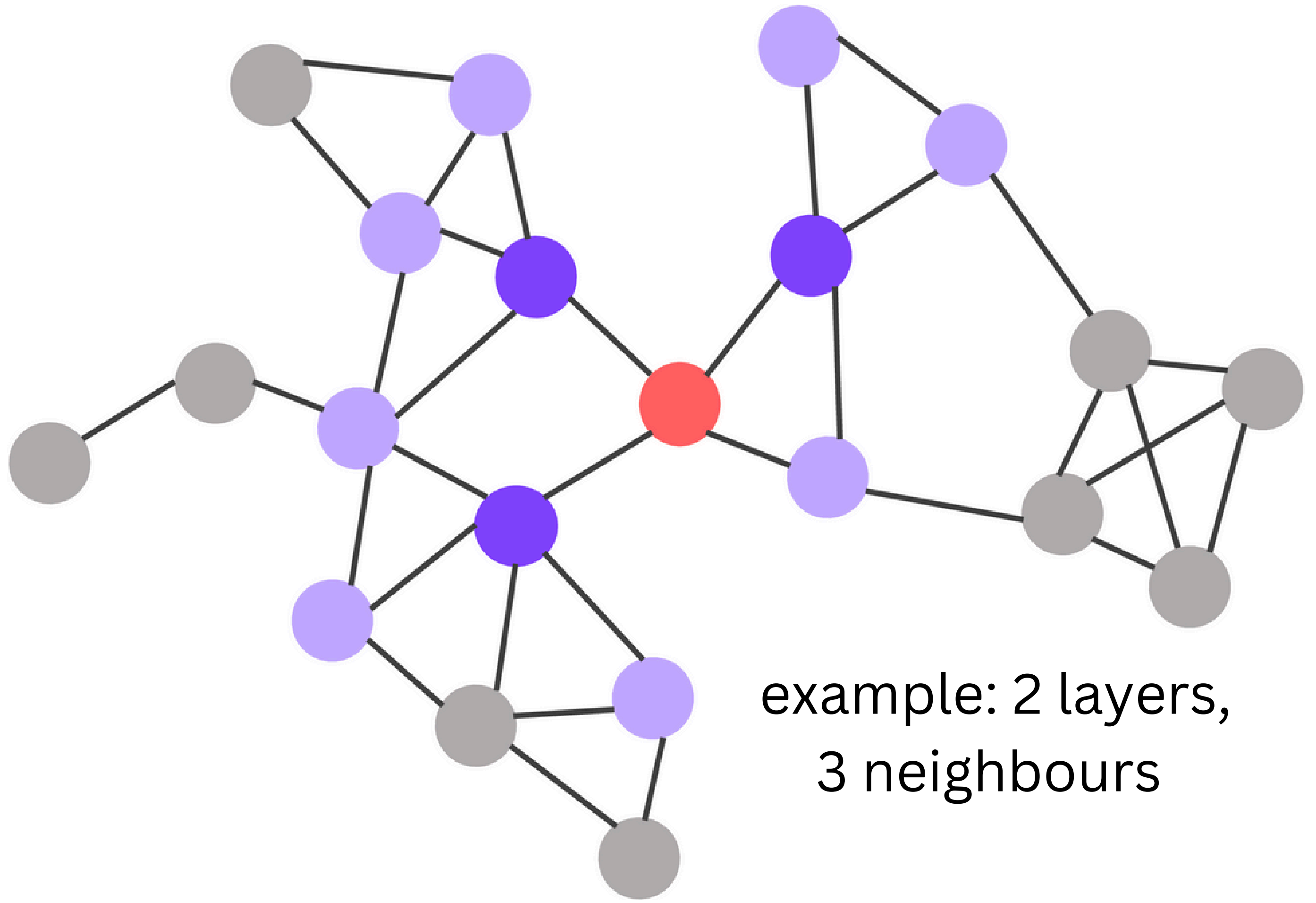
Scalability

Large LHC datasets → large scale graph constraints

Solutions:

- restrict the **depth of neighbour sampling**
  - limit number of layers (which also avoids vanishing gradient problem)
  - recursively sample a fixed max number of neighbours for each node
- torch geometric **sparse tensors**
- **subsampling** nodes
- **mini-batching**
- careful choice of **edge fraction** (by tuning linking length) to decrease density of adjacency matrix
- parallelising across **multiple GPU**

Neighbour sampling



example: 2 layers,
3 neighbours

Anomaly detection models

Python libraries for anomaly detection in multivariate data:

PyGOD

- Python Graph Outlier Detection
- Scalable for processes large graphs with mini-batch and sampling
- Our focus: GAE based on Kipf+Welling VGAEs arXiv:1611.07308, 2016
- Implements options for backbone: clustering, GNN+AE, MF, MLP+AE, GAN, GNN+SSL+AE

PyOD

- Python Outlier Detection
- Our focus: autoencoder neural network
- Implements other algorithms, of types: probabilistic, linear model, proximity-based, outlier ensembles, neural networks, R-graph

- Parameters in our implementations:
  - N hidden dimensions = 5
  - N epochs = 20-30
  - N layers = 4-6
  - Loss = MSE

$$\mathrm{MSE} = \frac{1}{n}\sum_{i=1}^{n}(Y_i - \hat{Y}_i)^2$$

MSE = mean squared error
n = number of data points
Y_i = observed values
$\hat{Y}_i$ = predicted values

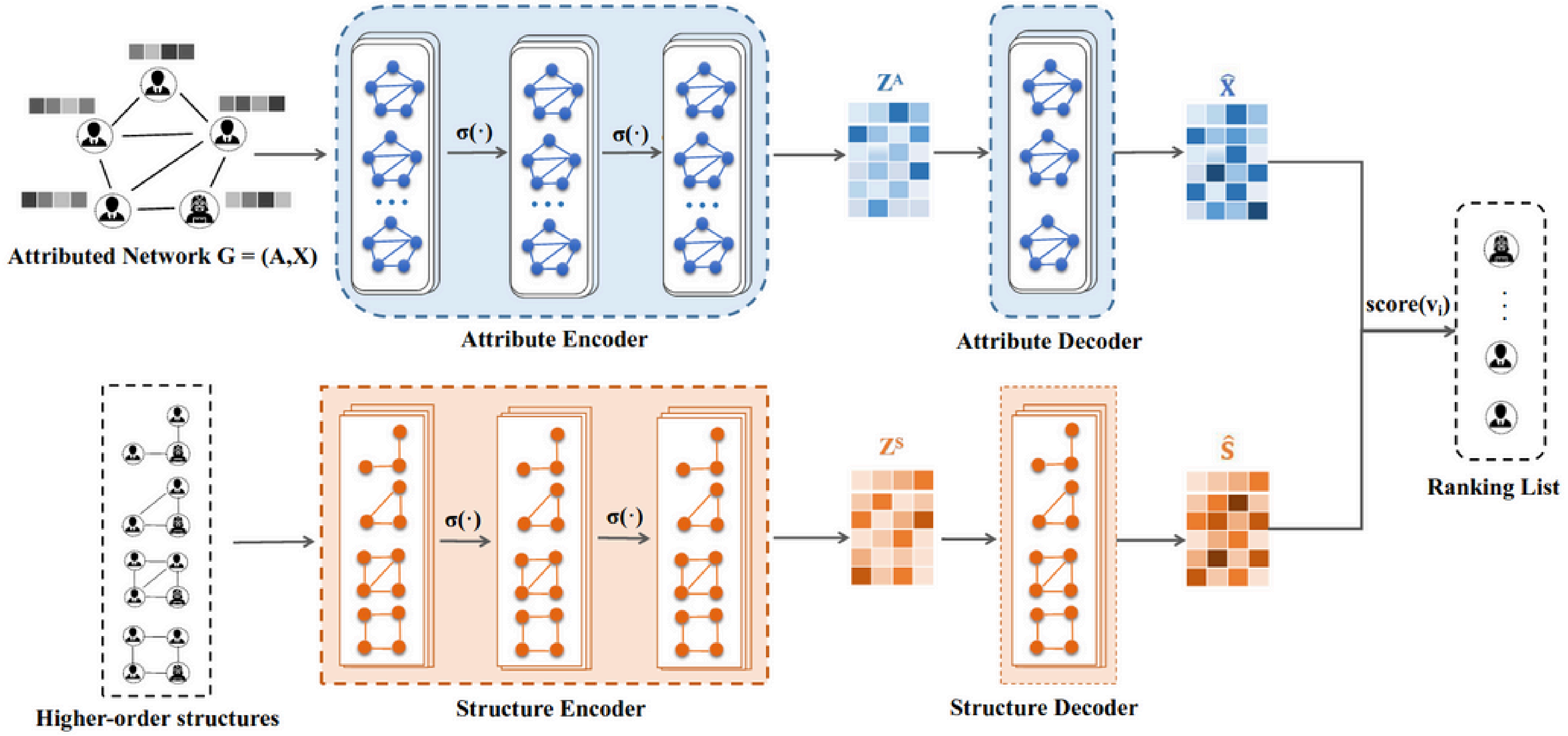
Anomaly detection models

Other options for AD GNN models in PyGOD:

GUIDE

- Structure encoder/decoder separate from node attribute encoder/decoder
- ‘Structure’ here refers to common small-scale motif structures

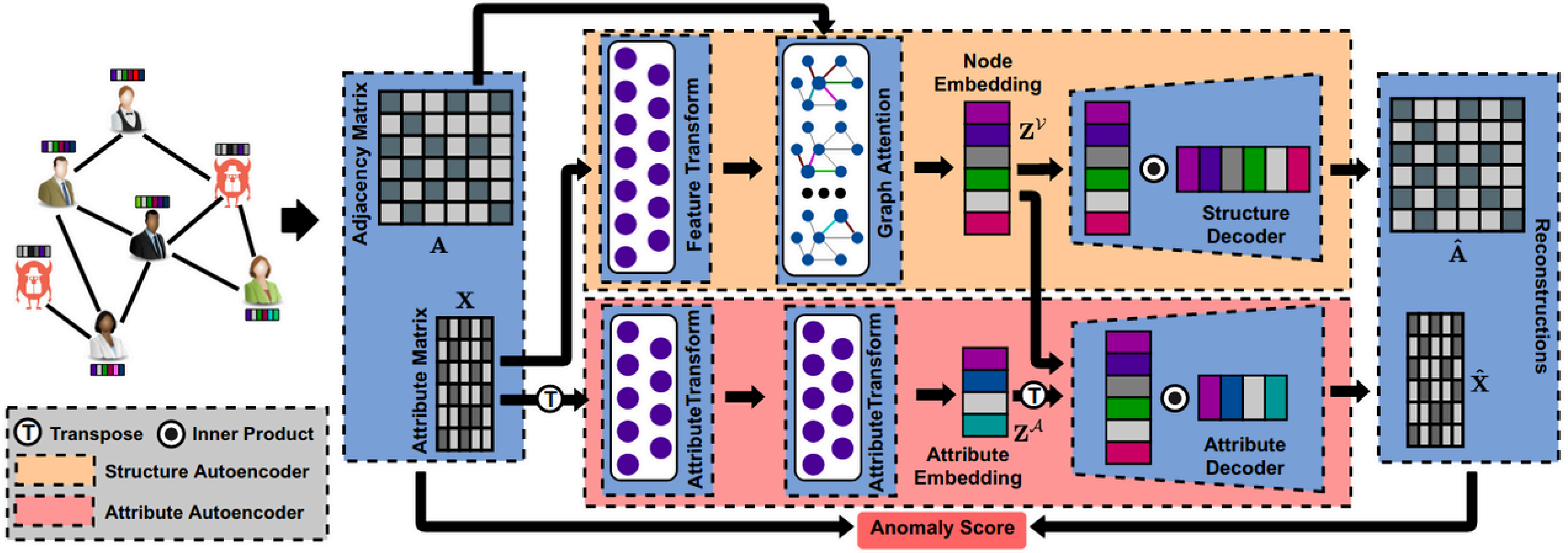
[arxiv:2406.04690](https://arxiv.org/abs/2406.04690)



AnomalyDAE

- Dual autoencoders:
  a. Adjancency (structure) and attribute matrices as input
  b. Attribute-only embeddings

[arxiv:2002.03665](https://arxiv.org/abs/2002.03665)



Hyperparameters

Convolution model	Distance metric	Graph domain	GNN layers	MLP layers	Edge fraction	Neighbours sampled [nodes, layers]	Dropout
High-level kinematic input variables							
DNN				[12, 12, 12, 12]			0.05
GCN	Euclidean	High-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.2	[60, 6]	0.0
GCN	Cosine	High-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.2	[60, 6]	0.0
GraphConv	Euclidean	High-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.2	[60, 6]	0.0
GraphConv	Cosine	High-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.2	[20, 2]	0.0
GCN	EMD	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.2	[60, 6]	0.0
GraphConv	EMD	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.2	[60, 6]	0.0
Low-level kinematic input variables							
DNN				[12, 12, 12, 12]			0.1
GCN	Euclidean	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GCN	Cosine	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GCN	EMD	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GraphConv	Euclidean	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GraphConv	Cosine	Low-level kinematic space	[12, 12]		0.1	[20, 2]	0.0
GraphConv	EMD	Low-level kinematic space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GCN	Euclidean	Latent space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GCN	Cosine	Latent space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GraphConv	Euclidean	Latent space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0
GraphConv	Cosine	Latent space	[32, 32, 32, 32]	[12, 12]	0.1	[60, 6]	0.0

Table 6.2: Table summarising the neural network architecture used in the training for each variation of the baseline DNN and GNN models.

4 hidden layers with 12 neurons each (multilayer perceptrons, classifier after the GNN, ignores the graph)

4 message-passing layers each mapping 32-dim node features, each using graph structure to update node embeddings

Results summary: model-dependent search

Convolution model	Distance metric	Graph domain	AUC (validation)
High-level kinematic input variables			
DNN			0.956
GCN	Euclidean	High-level kinematic space	0.951
GCN	Cosine	High-level kinematic space	0.954
GraphConv	Euclidean	High-level kinematic space	0.943
GraphConv	Cosine	High-level kinematic space	0.952
GCN	EMD	Low-level kinematic space	0.954
GraphConv	EMD	Low-level kinematic space	**0.972**
Low-level kinematic input variables			
DNN			0.919
GCN	Euclidean	Low-level kinematic space	0.826
GCN	Cosine	Low-level kinematic space	0.852
GCN	EMD	Low-level kinematic space	0.901
GraphConv	Euclidean	Low-level kinematic space	0.812
GraphConv	Cosine	Low-level kinematic space	0.804
GraphConv	EMD	Low-level kinematic space	**0.951**
GCN	Euclidean	Latent space	0.892
GCN	Cosine	Latent space	0.901
GraphConv	Euclidean	Latent space	0.892
GraphConv	Cosine	Latent space	0.873

Table 6.3: Table summarising the Area Under the ROC curves (AUC) values evaluated on the validation dataset, for the baseline DNNs and various GNN models, trained using the high-level or low-level kinematic variables as input features.