Optimisation and Deployment of an AI pipeline based on a Graph Neural Network (GNN) for Track Finding at LHCb

Anthony Correia, Fotis Giasemis,
Nabil Garroum, Vava Gligorov, Bertrand Granado

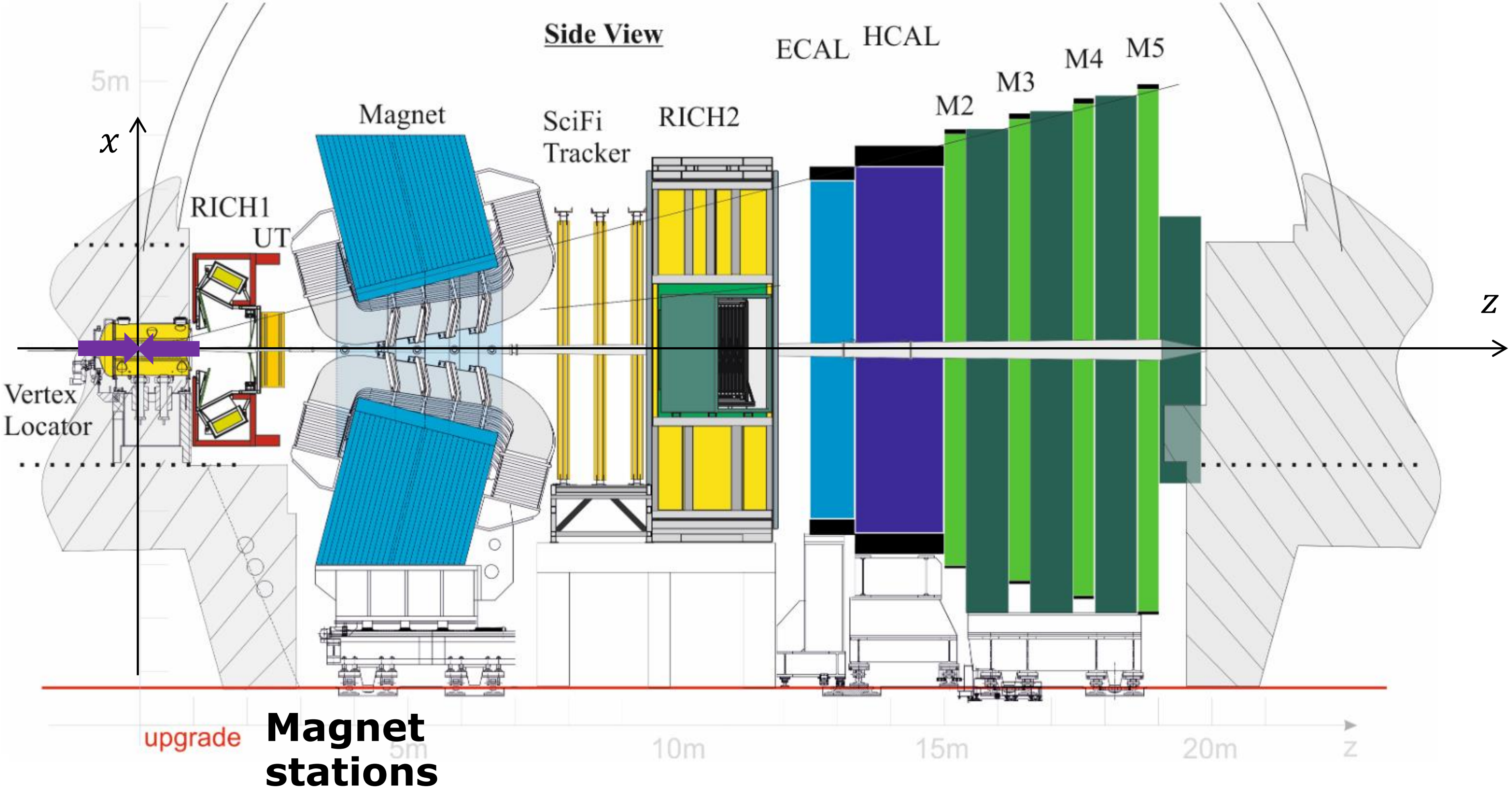
22nd March 2024

Protection zone

In collaboration with

LHCb Detector

- **2 bunches of** $\sim 10^{11}$ **protons** cross every ~**30 ns**
  → $\approx$ **30 MHz** bunch crossing rate
- $\sim$ **5 proton-proton collisions / bunch crossing**

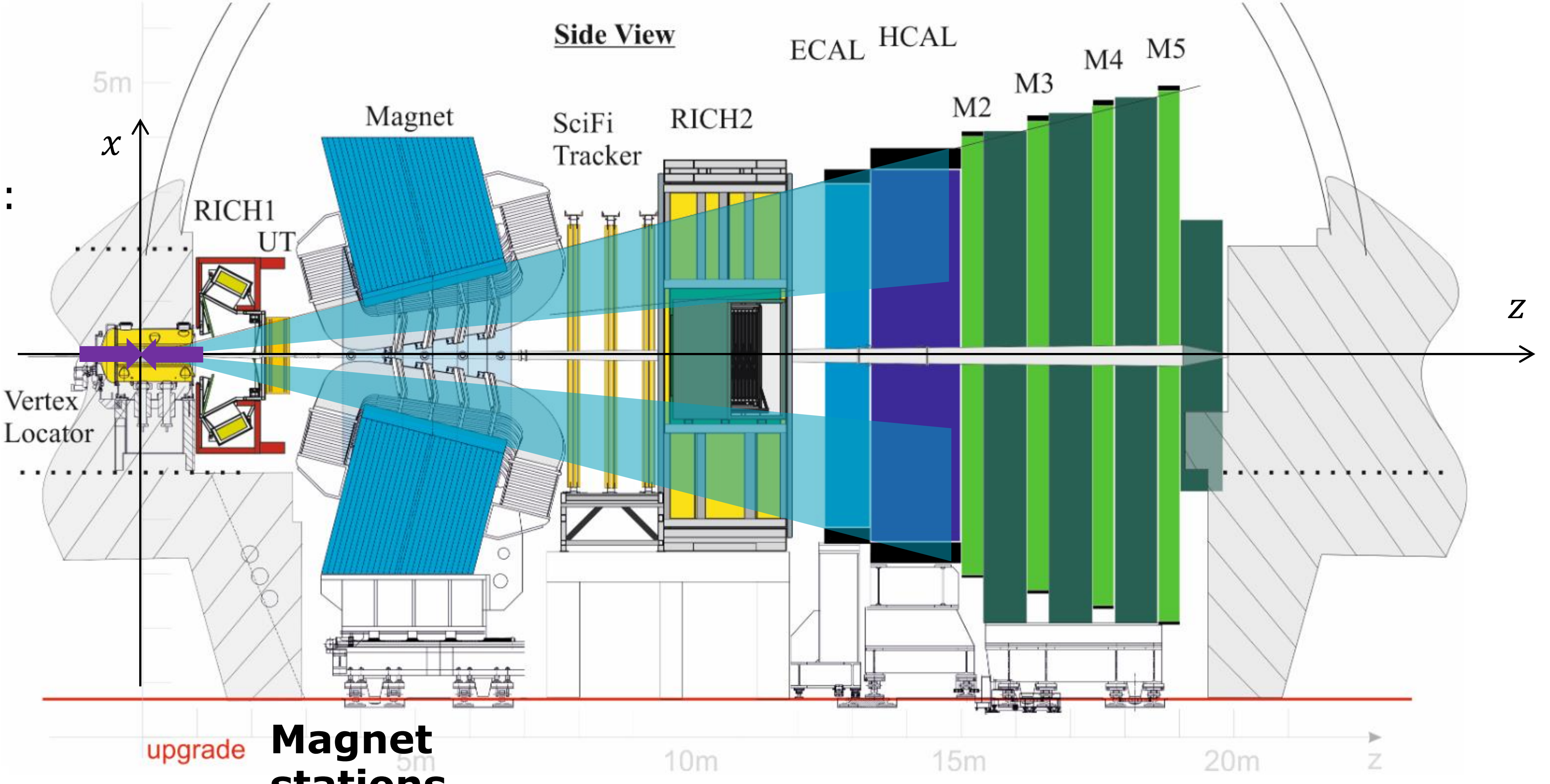


Problem Formulation

LHCb Detector

- **2 bunches of** $\sim 10^{11}$ **protons** cross every ~**30 ns**
  $\rightarrow \approx$ **30 MHz** bunch crossing rate
- $\sim$ **5 proton-proton collisions / bunch crossing**

- **Detector acceptance**:
  1°< |θ| <15°



Magnet stations

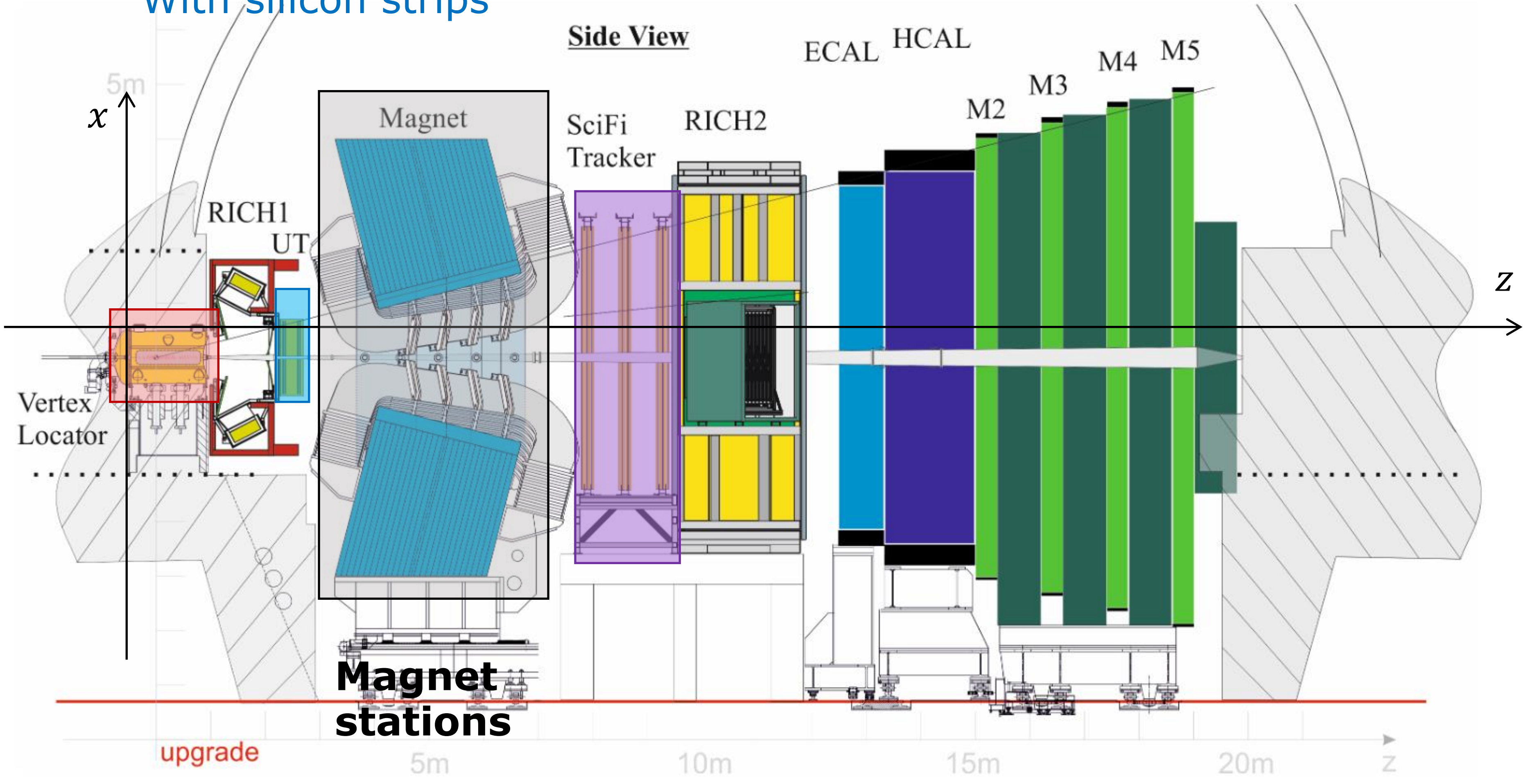
Problem Formulation

3 Tracking Detectors

Velo
Vertex **Lo**cator
With silicon pixels
No magnetic field

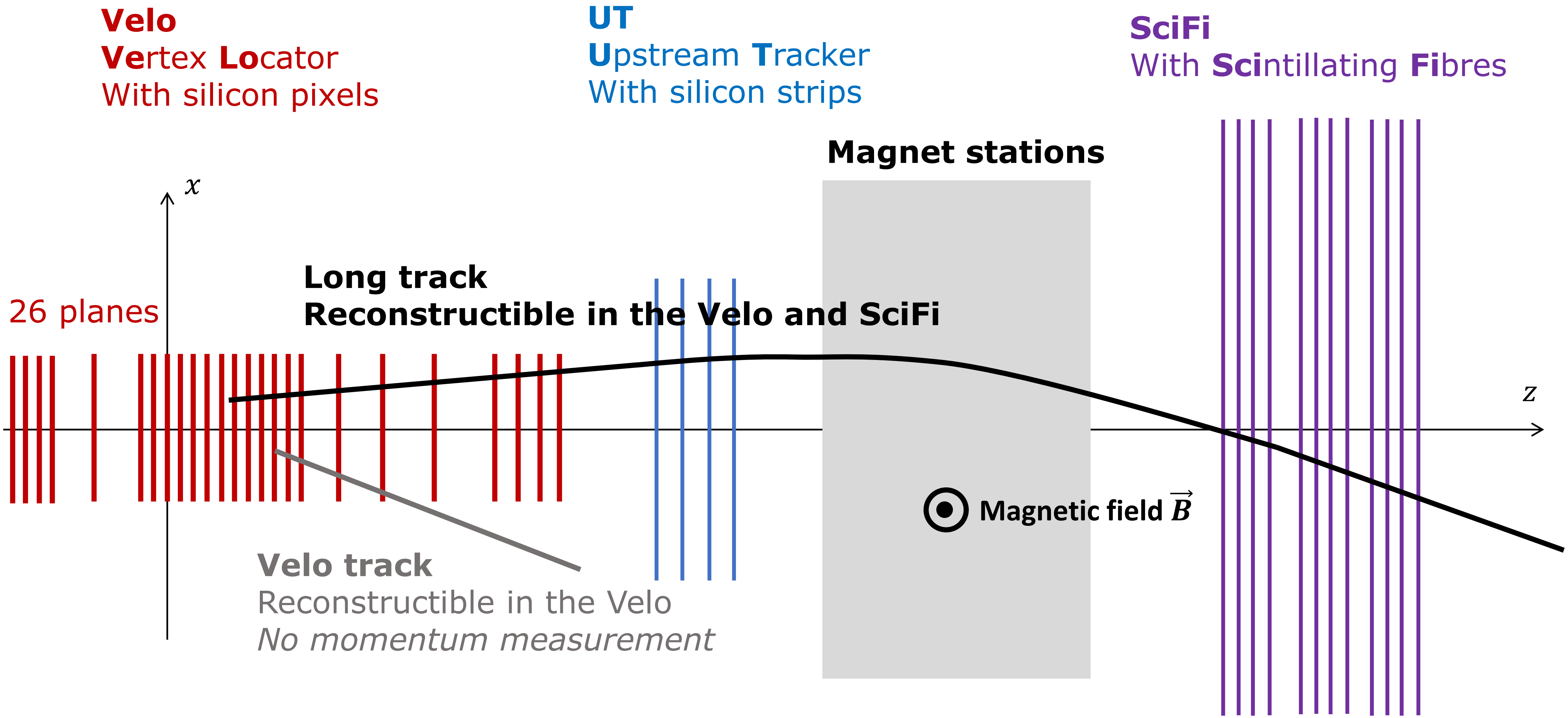
UT
Upstream **T**racker
With silicon strips

SciFi
With **Sci**ntillating **Fi**bres



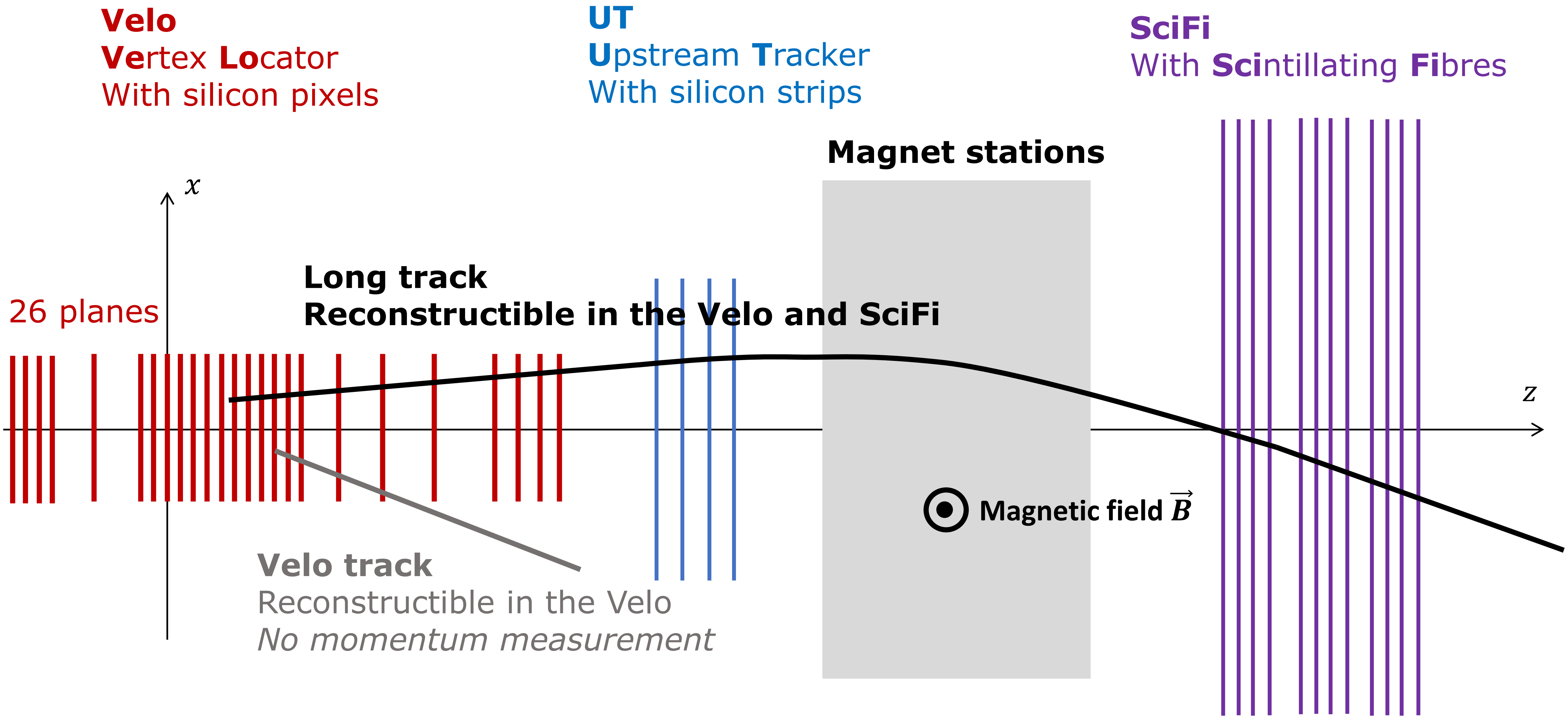
Problem Formulation

3 Tracking Detectors



Problem Formulation

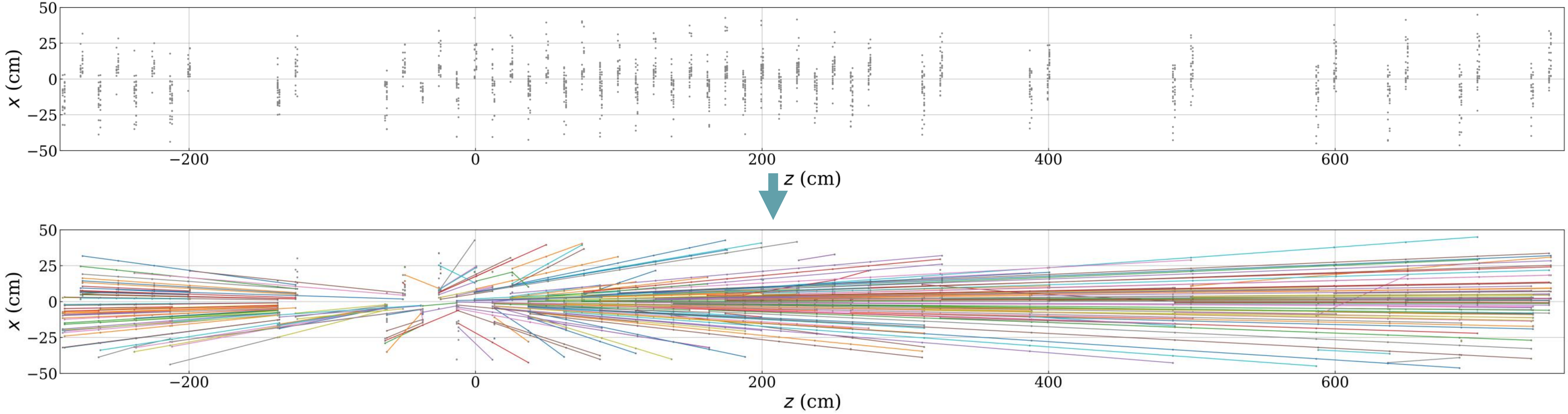
3 Tracking Detectors



Problem Formulation

Track Finding in the Velo

Track finding: find **tracks** from **hits**

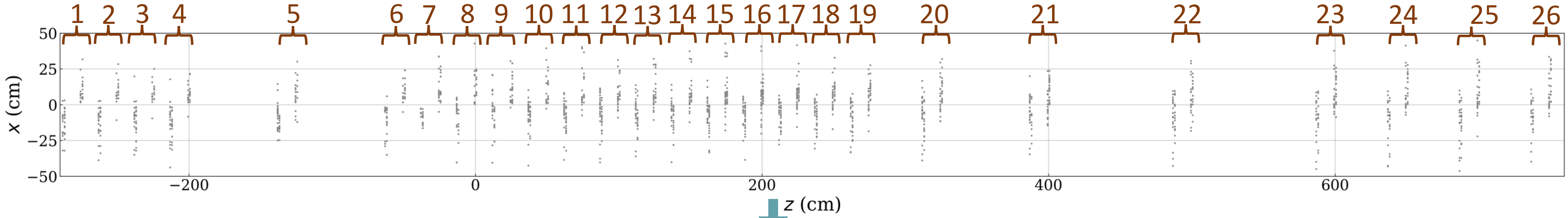


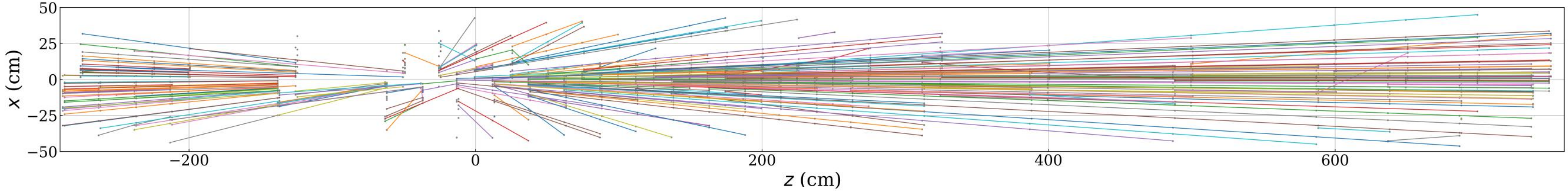
Problem Formulation

Track Finding in the Velo

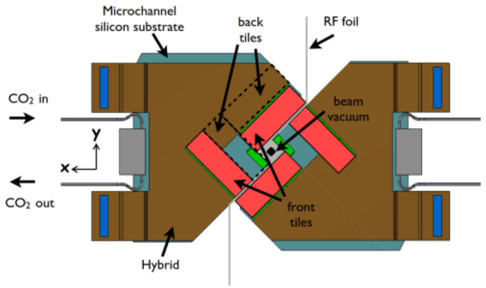
Track finding: find **tracks** from **hits**

Plane number

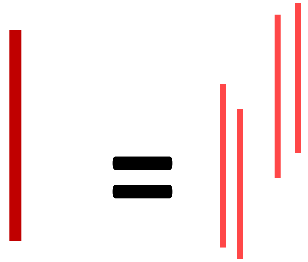




1 plane = **4 sensor planes**



1 plane **4 sensor planes**

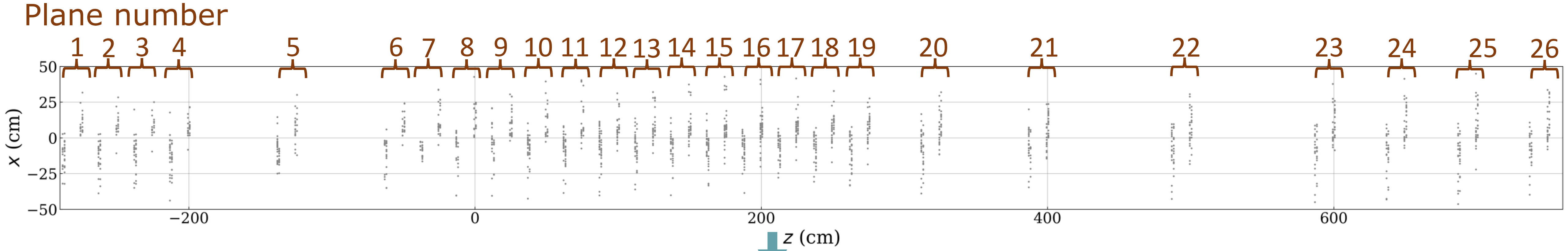


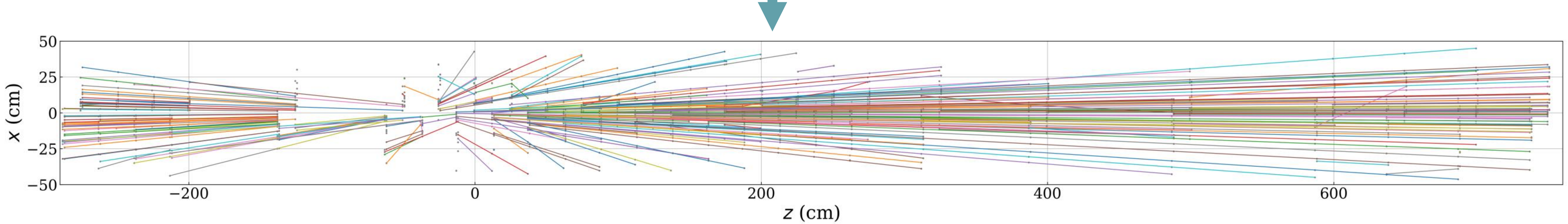
[P. C. Tsopelas, 'A Silicon Pixel Detector for LHCb', PhD Thesis, Vrije U., Amsterdam, 2016.]

Problem Formulation

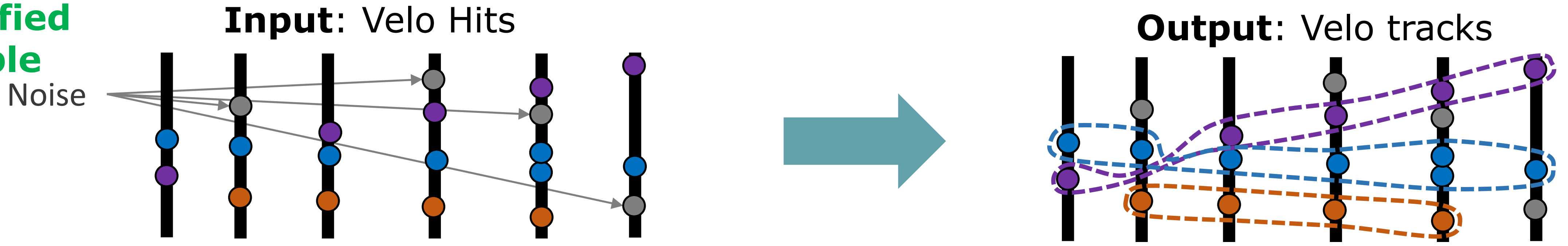
Track Finding in the Velo

Track finding: find **tracks** from **hits**





Simplified example

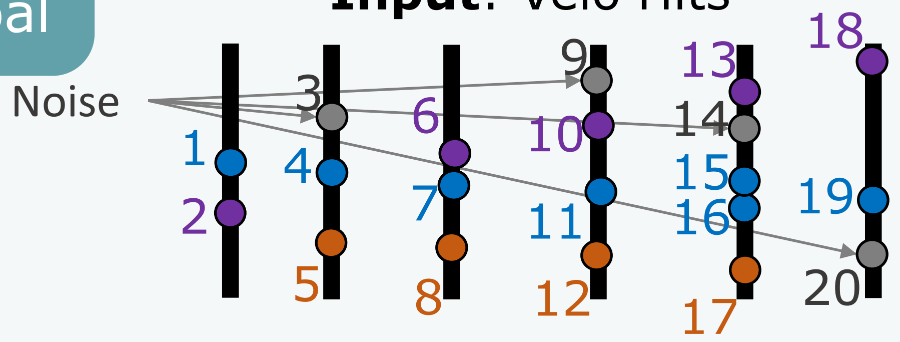


Problem Formulation

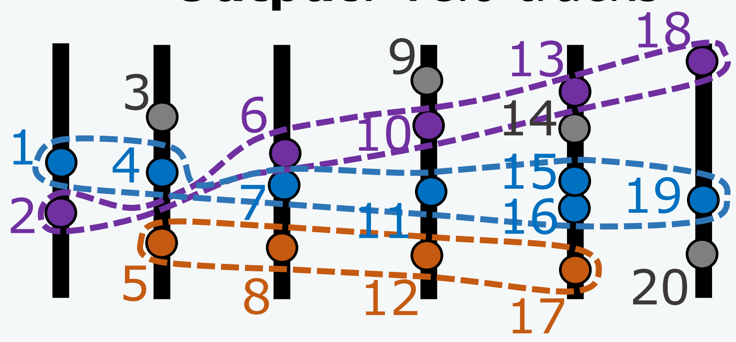
What is Track Finding?

Goal

Input: Velo Hits



Output: Velo tracks



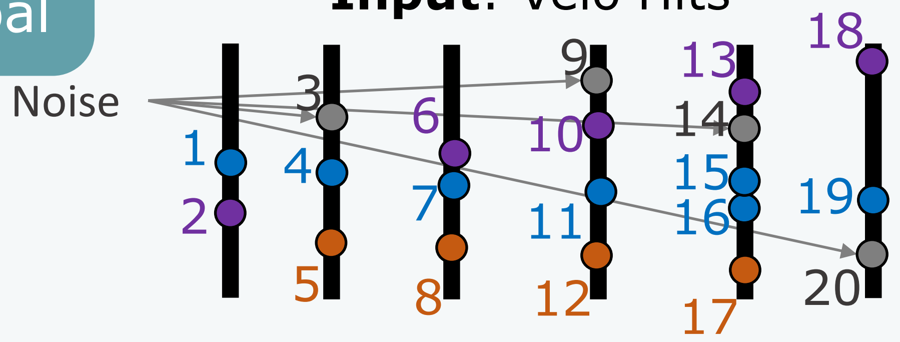
- Everything in **GPU**
- ⇒ needs to be as much **parallelised** as possible (NOT **sequential**)

Problem Formulation

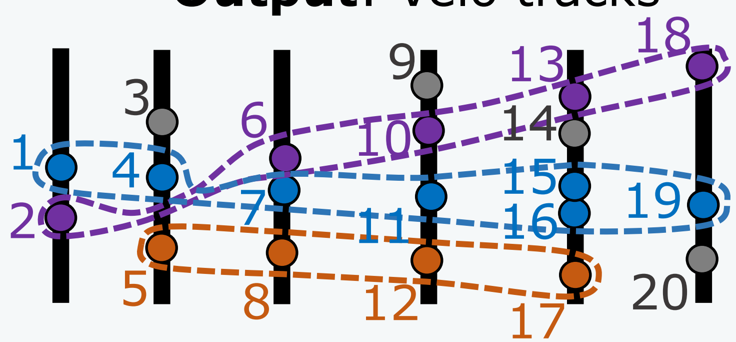
What is Track Finding?

Goal

Input: Velo Hits



Output: Velo tracks



Hit coordinates

$$X = \begin{pmatrix} r_1 & \phi_1 & z_1 \\ r_2 & \phi_2 & z_2 \\ r_3 & \phi_3 & z_3 \\ \vdots & \vdots & \vdots \\ r_{15} & \phi_{15} & z_{15} \\ r_{16} & \phi_{16} & z_{16} \end{pmatrix}$$

List of **connected components**

- {1, 4, 7, 11, 15, 16, 19}
- {2, 6, 10, 13, 18}
- {3}
- {5, 8, 12, 17}
- {9}
- {14}
- {20}

- Everything in **GPU**
- ⇒ needs to be as much **parallelised** as possible (NOT **sequential**)

Problem Formulation

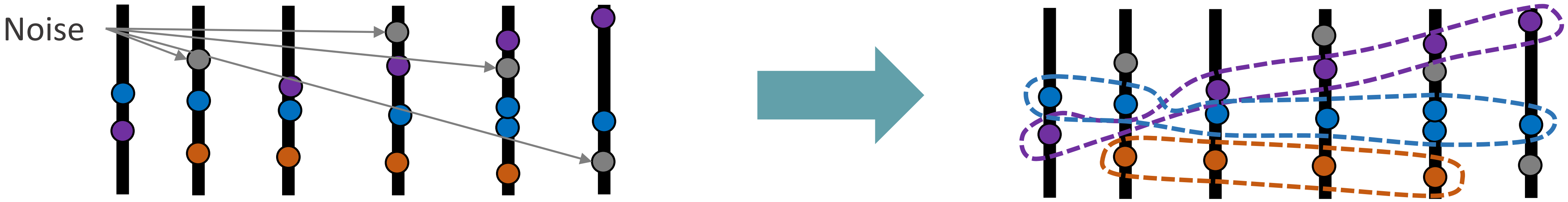
A GNN-Based Pipeline

- GNN-based pipeline is based on the work of **Exa.Trkx** (*Eur. Phys. J. C* **81**, 876 (2021)).
- With **PyTorch**

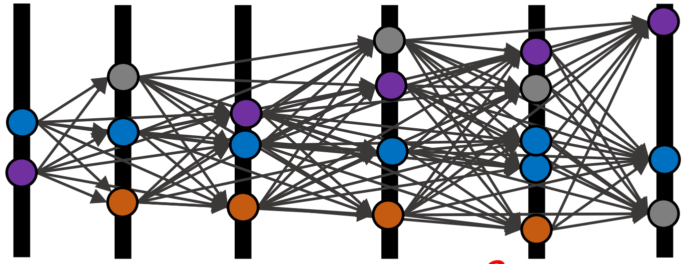
Problem Formulation

A GNN-Based Pipeline

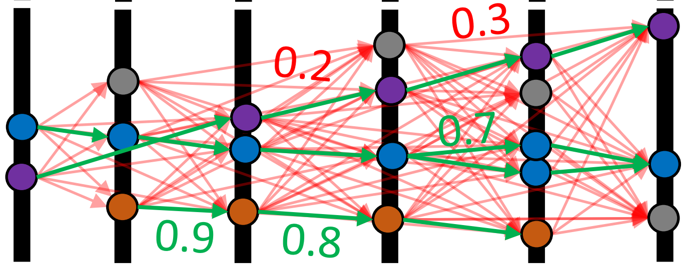
Goal



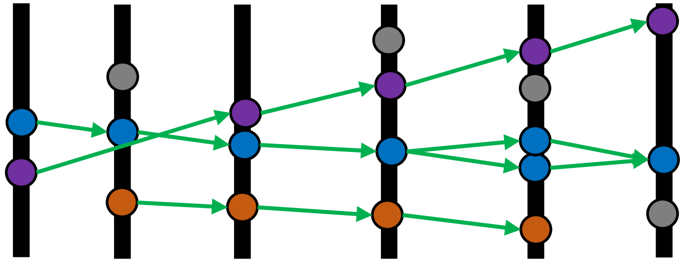
1 **Build a rough graph**
e.g., link every hits to hits on the next 2 planes.



2 **Score every edge** between 0 (fake) and 1 (genuine) using a Graph Neural Network (GNN)



3 **Discard fake edges**
by requiring edge score > $s_{\mathrm{edge,min}}$ *(for instance)*



Problem Formulation

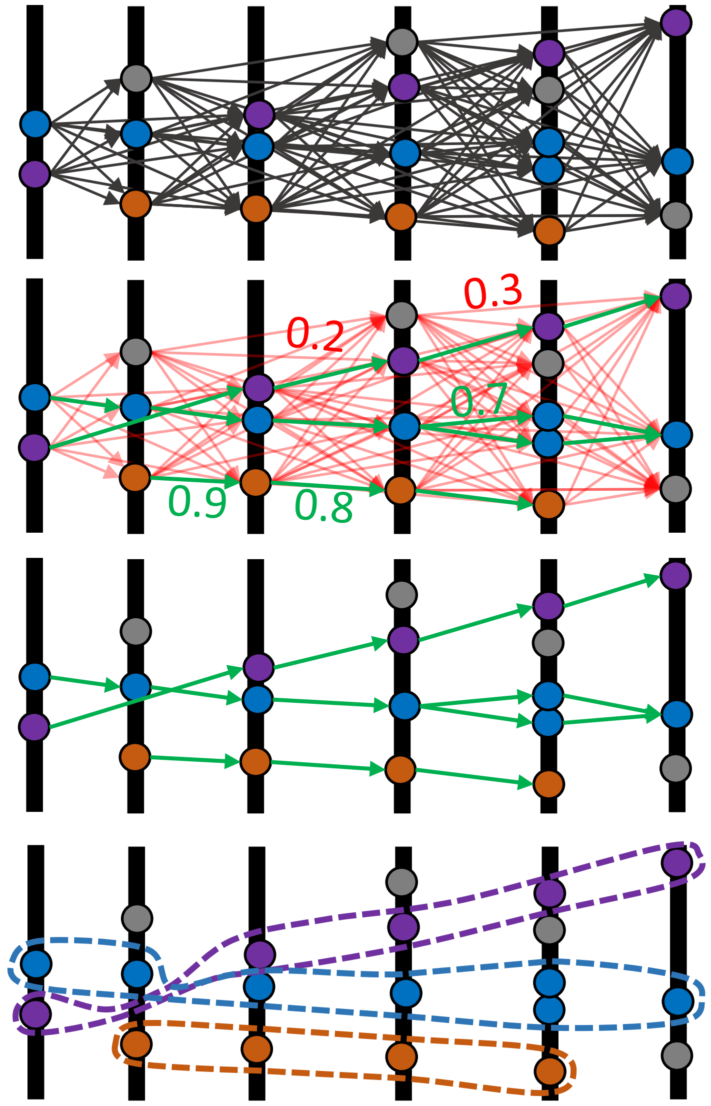
A GNN-Based Pipeline

1. **Build a rough graph**
   e.g., link every hits to hits on the next 2 planes.

2. **Score every edge** between 0 (fake) and 1 (genuine)
   using a Graph Neural Network (GNN)

3. **Discard fake edges**
   by requiring edge score > $s_{\text{edge,min}}$

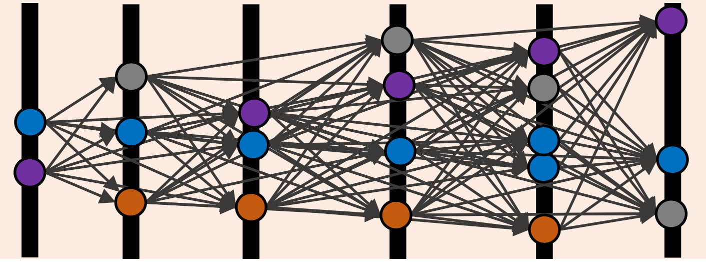
4. **Find connected components**
   using a weakly connected component algorithm
   Use of cugraph (GPU) or SciPy (CPU)



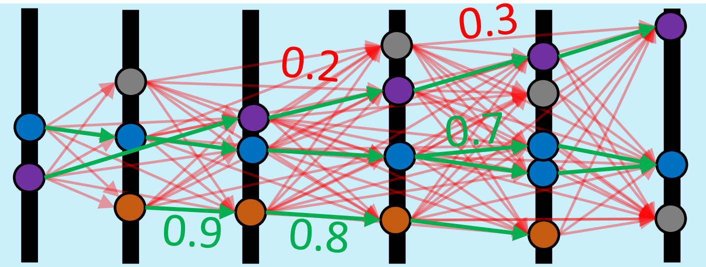
Problem Formulation

Build graph → Filter edges with GNN → Find connected components

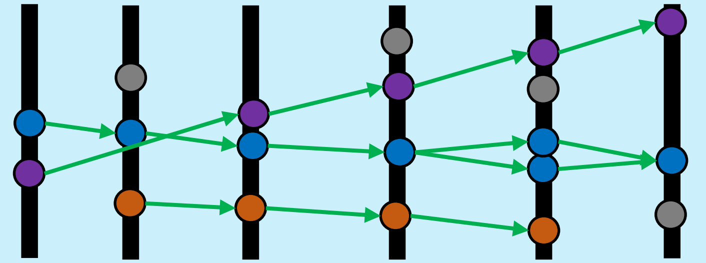
1 **Build a rough graph**
e.g., link every hits to hits on the next 2 planes.



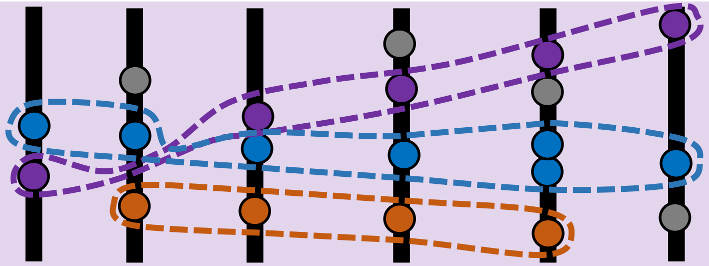
2 **Score every edge** between 0 (fake) and 1 (genuine)
using a Graph Neural Network (GNN)



3 **Discard fake edges**
by requiring edge score > $s_{\mathrm{edge,min}}$

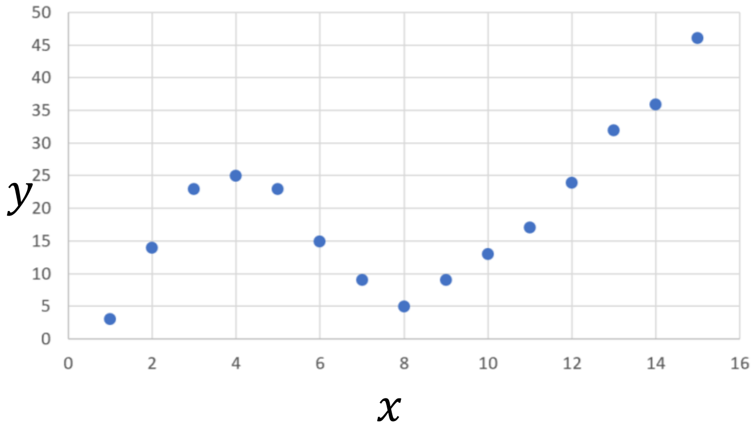


4 **Find connected components**
using a weakly connected component algorithm
Use of cugraph (GPU) or SciPy (CPU)



Interlude on Neural Network

Input x



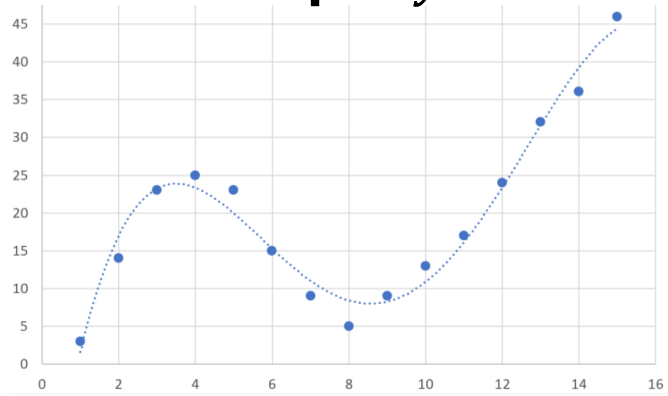
Predict y from x?

Polynomial model

5 parameters

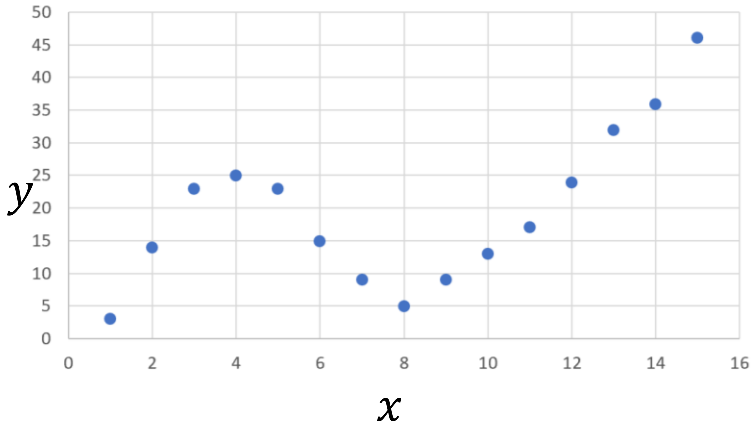
$$\hat{y} = ax^4 + bx^3 + cx^2 + dx + e$$

Output $\hat{y}$



Interlude on Neural Network

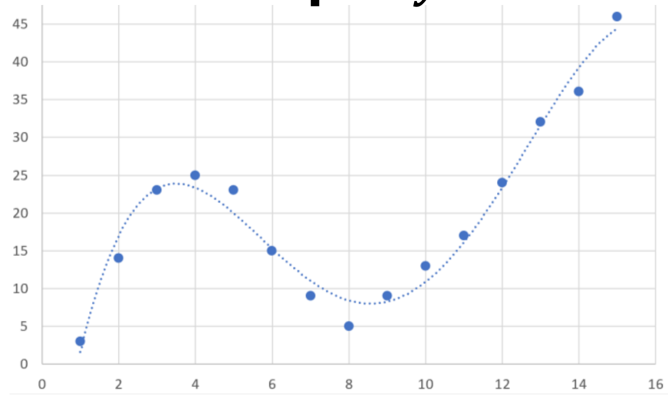
Input x



Predict y from x?

Polynomial model
5 parameters
$\hat{y} = ax^4 + bx^3 + cx^2 + dx + e$

Output $\hat{y}$



Probability of being a cat?

Deep Learning model
Many parameters
$\hat{y} = \mathbf{Model}(x)$

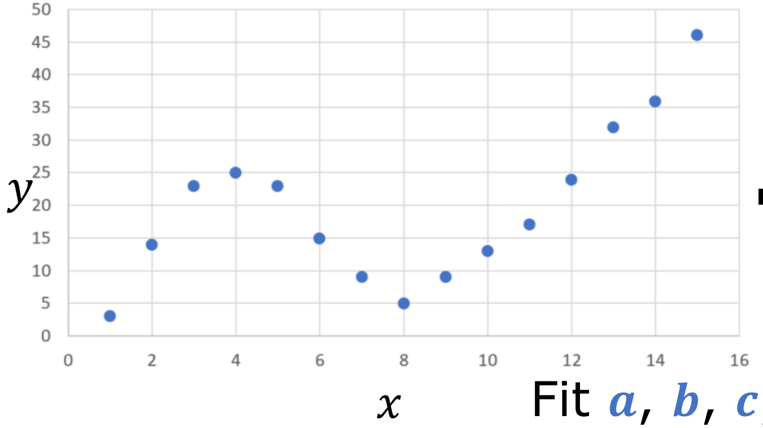
0.4 ✗

0.9 ✓

0.2 ✗

Interlude on Neural Network

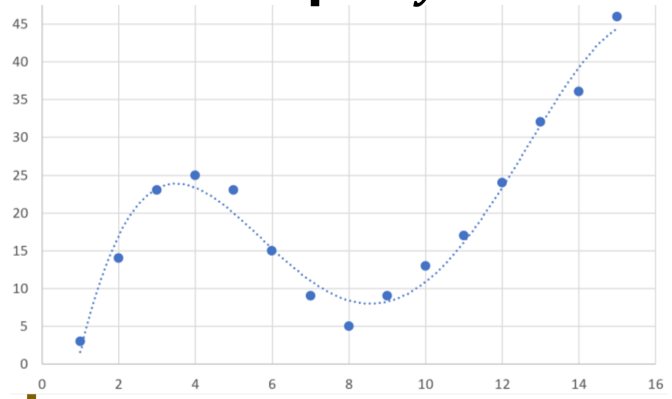
Input x



Predict y from x?

Polynomial model
5 parameters
$\hat{y} = ax^4 + bx^3 + cx^2 + dx + e$

Output $\hat{y}$



Fit $\boldsymbol{a}$, $\boldsymbol{b}$, $\boldsymbol{c}$, $\boldsymbol{d}$, $\boldsymbol{e}$ by minimising **Mean Squared Distance Loss**

$$\mathcal{L}_{\text{MSE}} = \sum_{i=1}^{N} (y_i - \hat{y}_i)^2$$

Probability of being a cat?

Deep Learning model
Many parameters
$\hat{y} = \mathbf{Model}(x)$

0.4

0.9

0.2

Fit **parameters** by minimising **Binary Cross Entropy Loss**

$$\mathcal{L} = -\sum_{i=1}^{N} (y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i))^2$$

Interlude on Neural Network

NB: For a picture, you should rather use a Convolutional Neural Network (CNN)



N pixels

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ x_{21} & \cdots & x_{2N} \\ x_{31} & \cdots & x_{3N} \end{pmatrix}$$

NB: Input features ***should be normalised***.

Probability of being a cat?

Multi-Layer Perceptron (MLP)

$$\hat{Y} = \begin{pmatrix} 0.4 \\ 0.9 \\ 0.2 \end{pmatrix}$$

Target

$$Y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

Interlude on Neural Network

NB: For a picture, you should rather use a Convolutional Neural Network (CNN)

N pixels

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ x_{21} & \cdots & x_{2N} \\ x_{31} & \cdots & x_{3N} \end{pmatrix}$$

NB: Input features ***should be normalised***.

Probability of being a cat?

Multi-Layer Perceptron (MLP)

$$\hat{Y} = \begin{pmatrix} 0.4 \\ 0.9 \\ 0.2 \end{pmatrix}$$

Target

$$Y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

1 layer

$$X_0 = X \longrightarrow \quad \longrightarrow \hat{Y} = X_1$$

$$X_1 = \sigma(X_0 \boldsymbol{W}_1 + \boldsymbol{b}_1)$$

Sigmoid function so that result $\in [0,1]$

Matrices W_1 and b_1 containing the parameters to fit

Interlude on Neural Network

NB: For a picture, you should rather use a Convolutional Neural Network (CNN)

N pixels

$$X = \begin{pmatrix} x_{11} & \cdots & x_{1N} \\ x_{21} & \cdots & x_{2N} \\ x_{31} & \cdots & x_{3N} \end{pmatrix}$$

Probability of being a cat?

Multi-Layer Perceptron (MLP)

$$\hat{Y} = \begin{pmatrix} 0.4 \\ 0.9 \\ 0.2 \end{pmatrix}$$

Target

$$Y = \begin{pmatrix} 0 \\ 1 \\ 0 \end{pmatrix}$$

3 layers

$$X_0 = \mathrm{X} \longrightarrow \hat{Y} = X_3$$

Activation function
(typically non-linear)

$$X_1 = \mathrm{ReLU}(X_0 \boldsymbol{W_1} + \boldsymbol{b_1})$$
$$X_2 = \mathrm{ReLU}(X_1 \boldsymbol{W_2} + \boldsymbol{b_2})$$
$$X_3 = \sigma(X_2 \boldsymbol{W_3} + \boldsymbol{b_3})$$

Fit W_1, b_1, W_2, b_2, W_3 and b_3 by minimising binary cross entropy loss

$$\mathcal{L} = -\sum_{i=1}^{N} (y_i \log(\hat{y}_i) - (1 - y_i)\log(1 - \hat{y}_i))^2$$

Stage 1: Graph Building

Graph Representation

Graph $\mathcal{G}$ is defined as :$(\mathcal{V}, \mathcal{E})$

- Set of **nodes / vertices** $\mathcal{V} = \{v_1, v_2, v_3, v_4, v_5, \ldots, v_N\}$
- Set of **edges** $\mathcal{E}$ ≡ connection between nodes

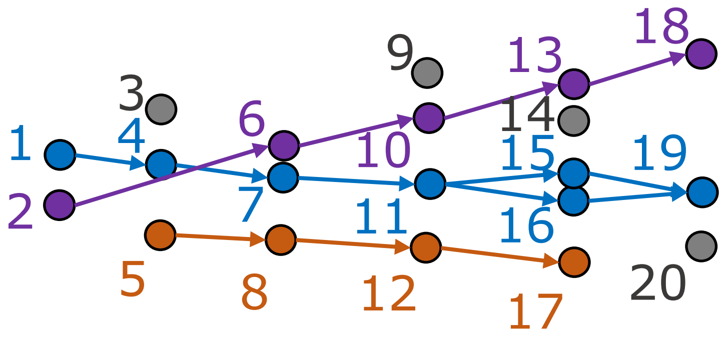
- **Features** / **attributes**:
  - **Node features**: node coordinates X
  - **Edge features**:
    concatenated node coordinates F

Graph Representation

Graph $\mathcal{G}$ is defined as :$(\mathcal{V},\mathcal{E})$

- Set of **nodes / vertices** $\mathcal{V}$ = $\{v_1, v_2, v_3, v_4, v_5, \dots, v_{19}, v_{20}\}$
- Set of **edges** $\mathcal{E}$ ≡ connection between nodes

$$\mathrm{I}_{\mathcal{E}} = \begin{pmatrix} 1 & 2 & 4 & 5 & 6 & 7 & 8 & 10 & 11 & 11 & 12 & 13 & 15 & 16 \\ 4 & 6 & 7 & 8 & 10 & 11 & 12 & 13 & 15 & 16 & 17 & 18 & 19 & 19 \end{pmatrix}$$



- **Features / attributes**:
  - **Node features**: node coordinates X

$$X = \begin{pmatrix} r_1 & \phi_1 & z_1 \\ r_2 & \phi_2 & z_2 \\ r_3 & \phi_3 & z_3 \\ \vdots & \vdots & \vdots \\ r_{19} & \phi_{19} & z_{19} \\ r_{20} & \phi_{20} & z_{20} \end{pmatrix}$$

  - **Edge features**:
    concatenated node coordinates F

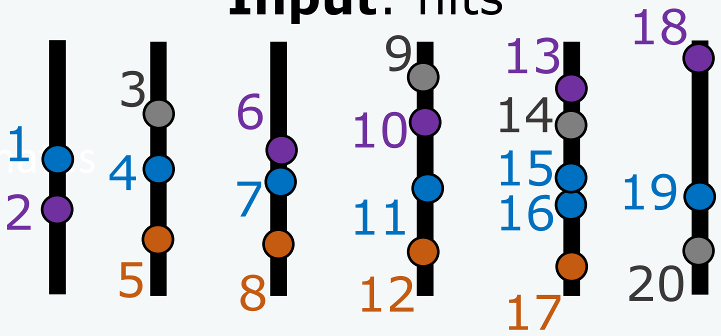
$$X_{\mathcal{E}} = \begin{pmatrix} r_1 & \phi_1 & z_1 & r_4 & \phi_4 & z_4 \\ r_2 & \phi_2 & z_2 & r_6 & \phi_6 & z_6 \\ r_4 & \phi_4 & z_4 & r_7 & \phi_7 & z_7 \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ r_{15} & \phi_{15} & z_{15} & r_{19} & \phi_{19} & z_{19} \\ r_{16} & \phi_{16} & z_{16} & r_{20} & \phi_{20} & z_{20} \end{pmatrix}$$

Stage 1: Graph Building

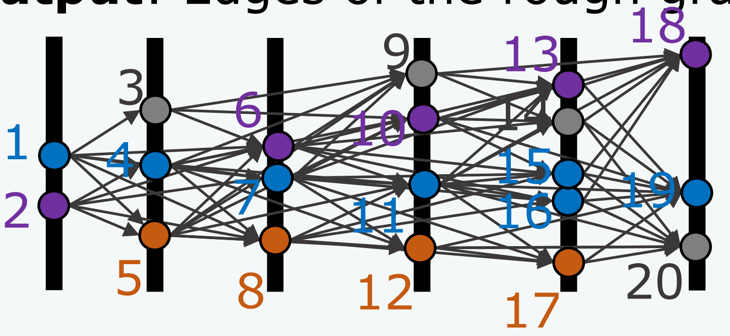
Problem Formulation

Goal

Input: hits



Output: Edges of the rough graph



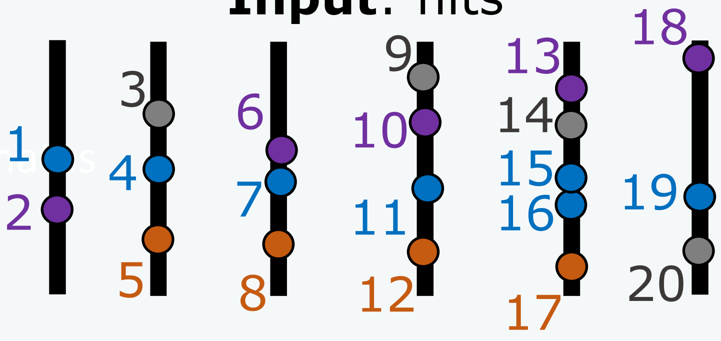
Build graph → Filter edges with GNN → Find connected components

Stage 1: Graph Building

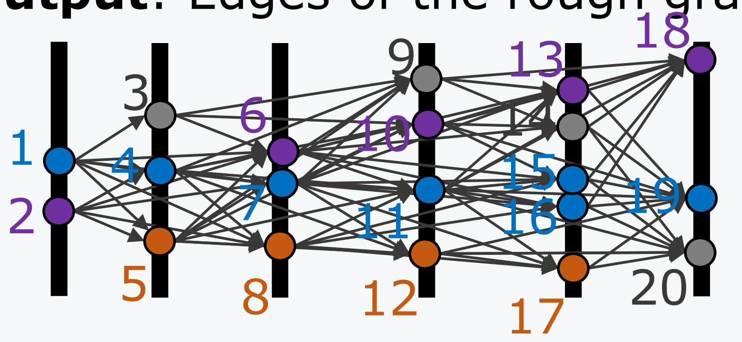
Problem Formulation

Goal

Input: hits



Output: Edges of the rough graph



Formally….

Hit coordinates

$$X = \begin{pmatrix} r_1 & \phi_1 & z_1 \\ r_2 & \phi_2 & z_2 \\ r_3 & \phi_3 & z_3 \\ \vdots & \vdots & \vdots \\ r_{19} & \phi_{19} & z_{19} \\ r_{20} & \phi_{20} & z_{20} \end{pmatrix}$$

Edge indices

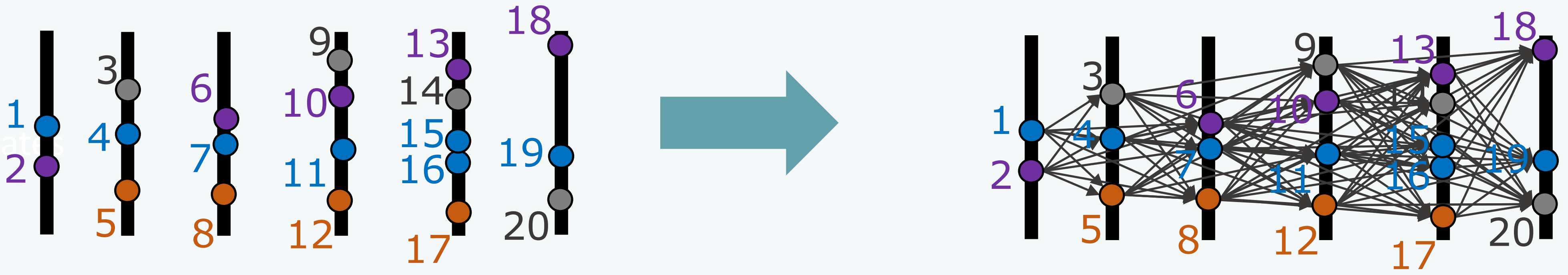
$$\mathrm{I}_{\mathcal{E}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & \cdots & 17 & 17 \\ 3 & 4 & 5 & 6 & 7 & 8 & 4 & 5 & 6 & 7 & 8 & \cdots & 19 & 20 \end{pmatrix}$$

Build graph | Filter edges with GNN | Find connected components

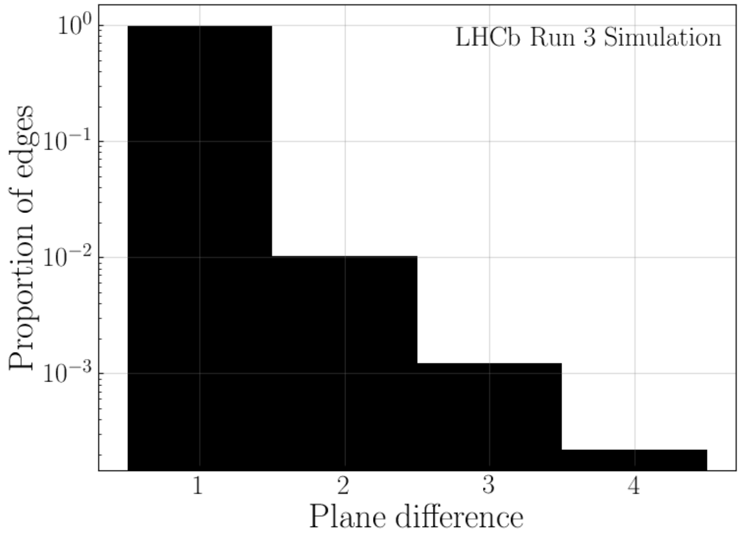
Stage 1: Graph Building

Idea

Goal



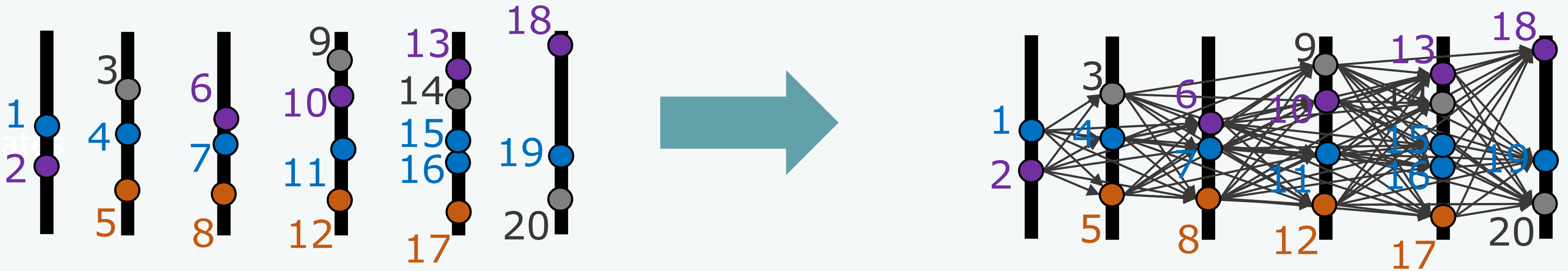
#	Idea	Observation
1	Connect **all the nodes** together.	Too many edges. 😡 99.9% of edges are ≤ 2 plane apart.
3		



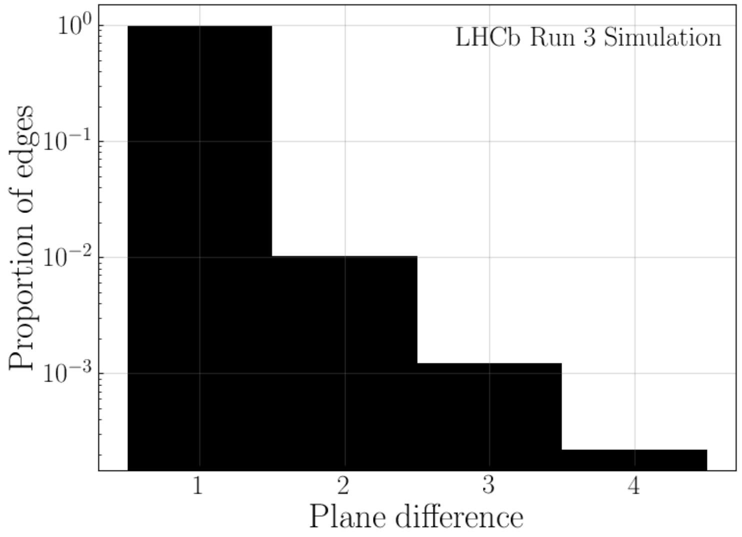
Stage 1: Graph Building

Idea

Goal



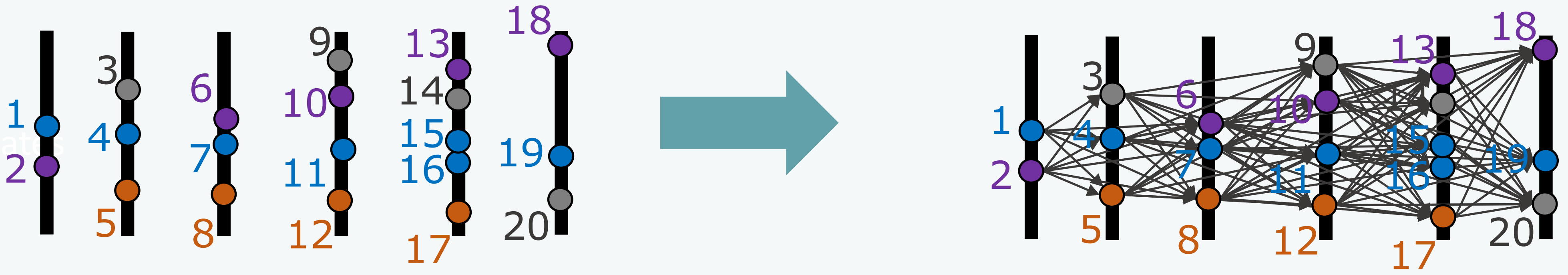
#	Idea	Observation
1	Connect **all the nodes** together.	Too many edges. 😡 99.9% of edges are ≤ 2 plane apart.
2	Connect **nodes in plane k** to **all the nodes in plane $k+1$ and $k+2$**	Still too many edges 😠
3		



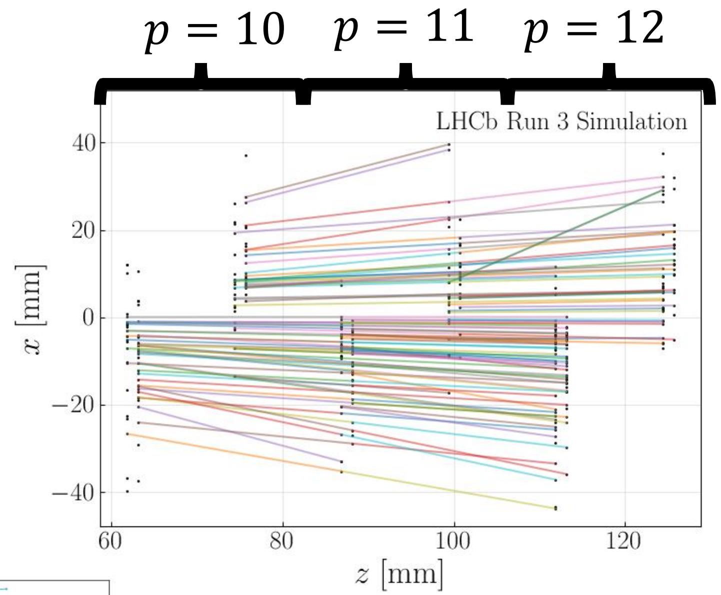
Stage 1: Graph Building

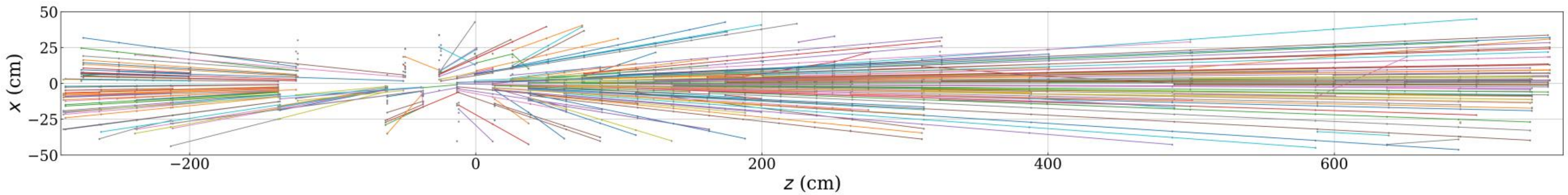
Idea

Goal



#	Idea	Observation
1	Connect **all the nodes** together.	Too many edges. 😡 99.9% of edges are ≤ 2 plane apart.
2	Connect **nodes in plane k** to **all the nodes in plane $k+1$ and $k+2$**	Still too many edges 😠 Edges tend to be: • Forward • Away from z-axis ↔ more tilted
3		

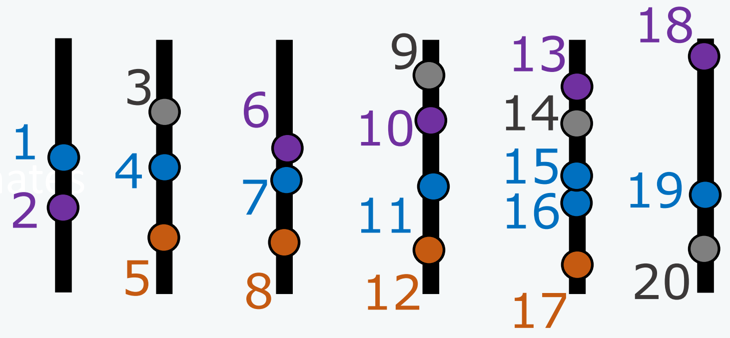


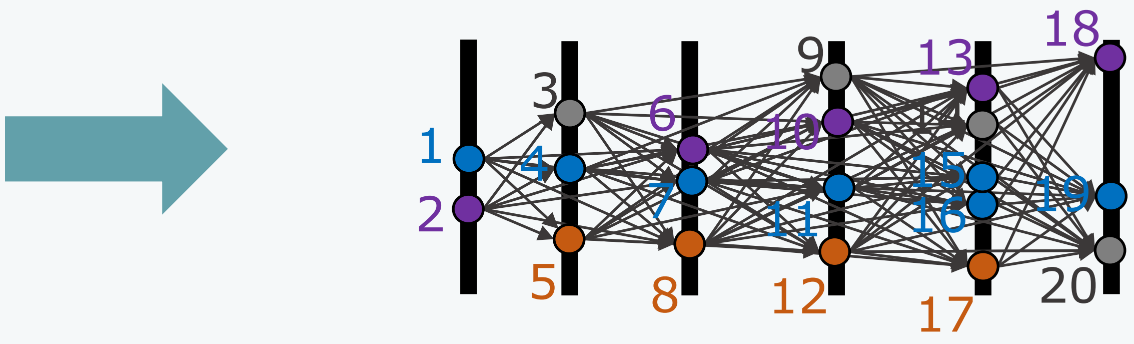


Stage 1: Graph Building

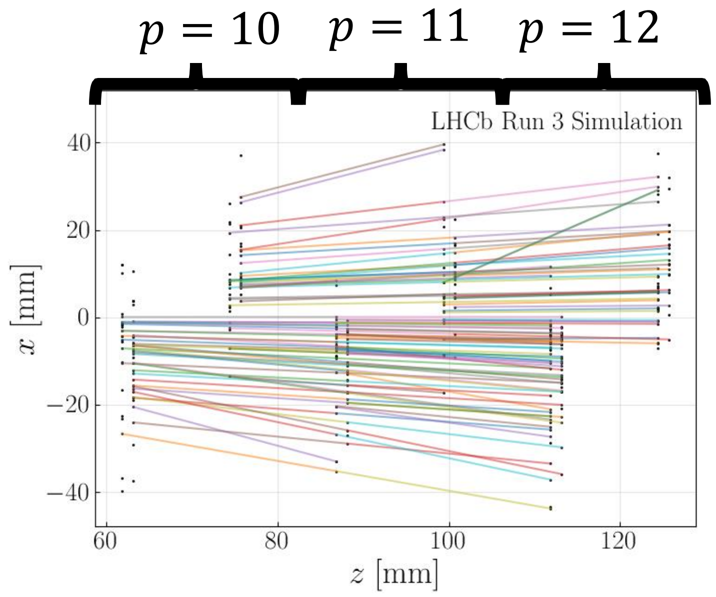
Idea

Goal





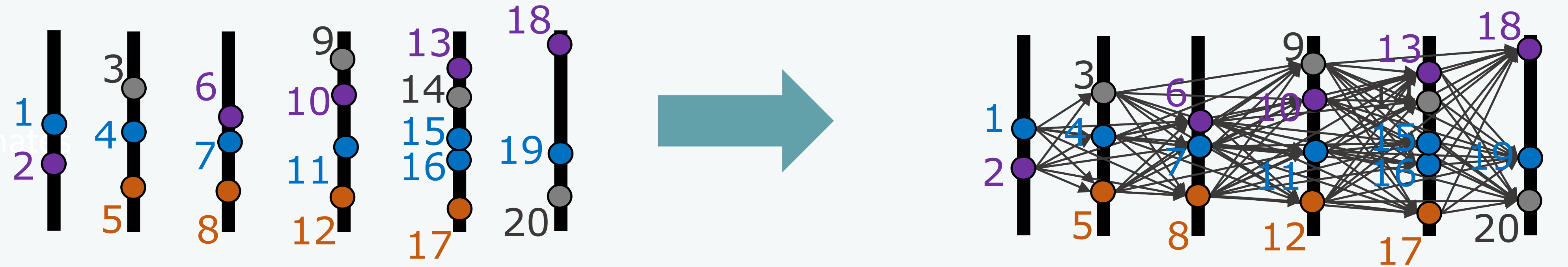
#	Idea	Observation
1	Connect **all the nodes** together.	Too many edges. 😡 99.9% of edges are ≤ 2 plane apart.
2	Connect **nodes in plane k to all the nodes in plane $k+1$ and $k+2$**	Still too many edges 😠 Edges tend to be: • Forward • Away from z-axis ↔ more tilted
3	Use a Neural Network to *capture* this trend.	🤯

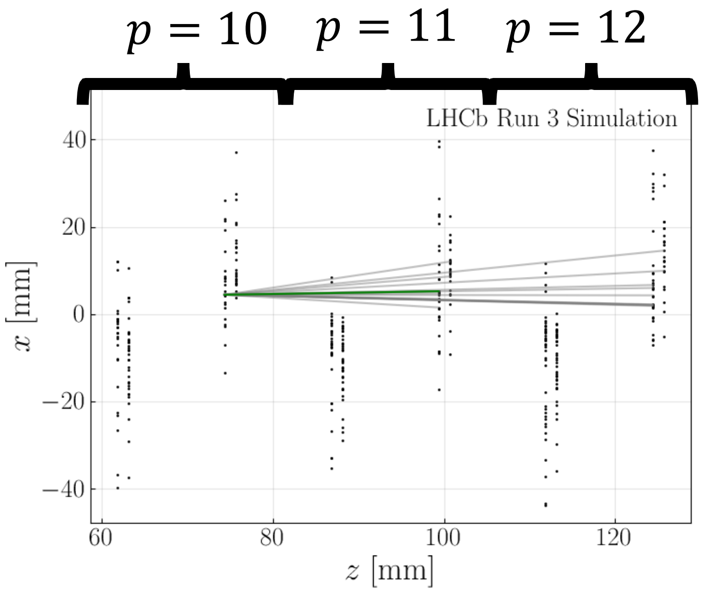


Stage 1: Graph Building

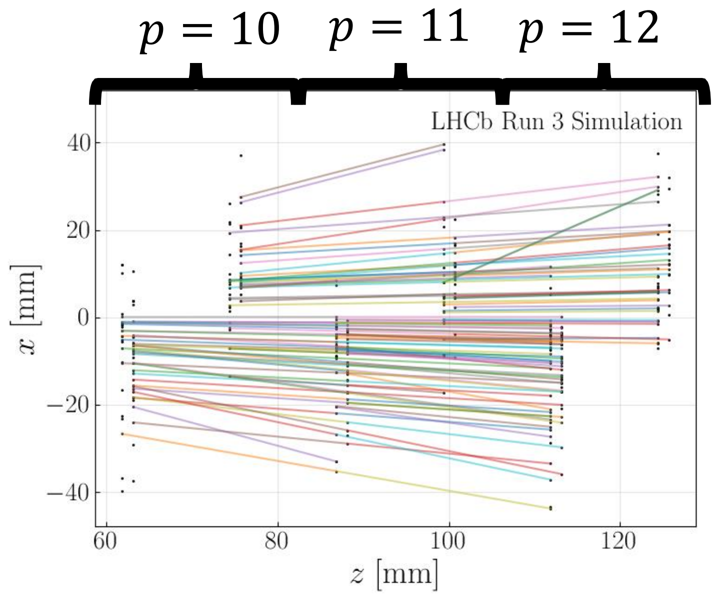
Idea

Goal





Example of edges drawn in the rough graph.
Only 1 true edge out of ~15 edges.



All the true edges

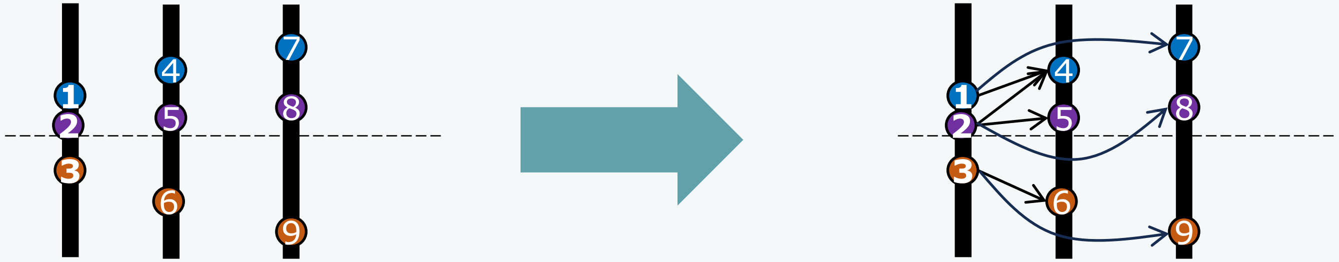
Stage 1: Graph Building

With an Embedding Network

Let's focus on connecting hits from plane p to plane $p+1$ and $p+2$

Goal

Hit coordinates



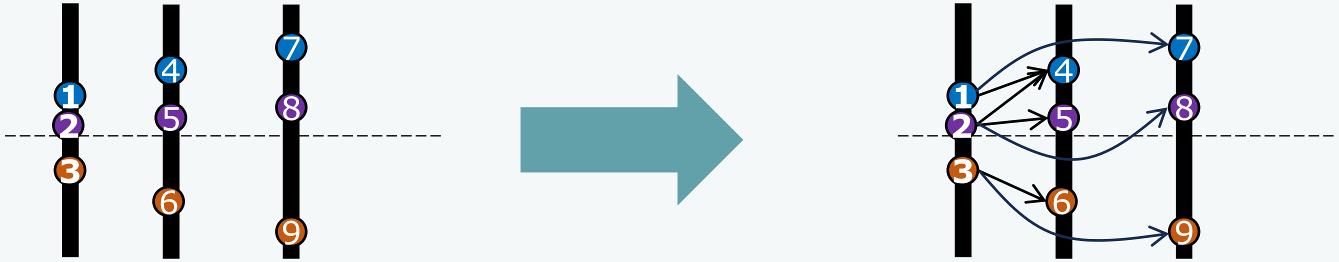
Stage 1: Graph Building

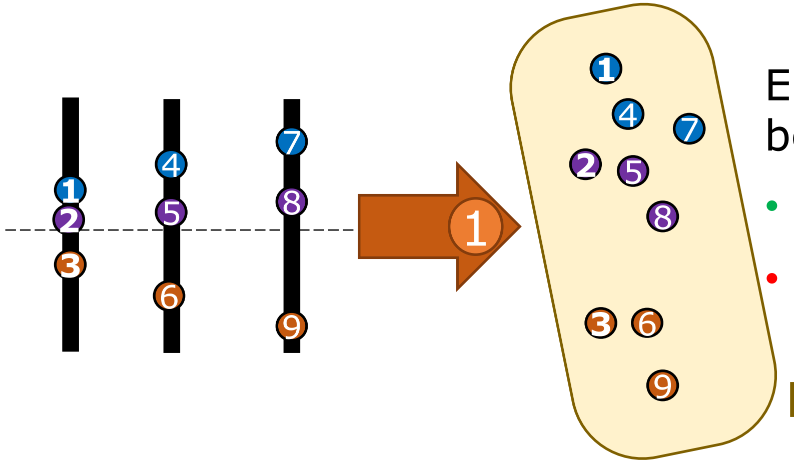
With an Embedding Network

Let's focus on connecting hits from plane p to plane $p+1$ and $p+2$

Goal

Hit coordinates





Embed each hit $\vec{x_k} \rightarrow \vec{e_k}$ so that, between nodes v_i and v_j:

- $d_{ij}^2 = \left\|\vec{e_i} - \vec{e_j}\right\|^2 < 1$: edge is likely
- $d_{ij}^2 = \left\|\vec{e_i} - e_j\right\|^2 > 1$: edge is unlikely

Embedding space

$$\vec{x_k} = \begin{pmatrix} r_k \\ \phi_k \\ z_k \end{pmatrix}$$

1

MLP

251 parameters

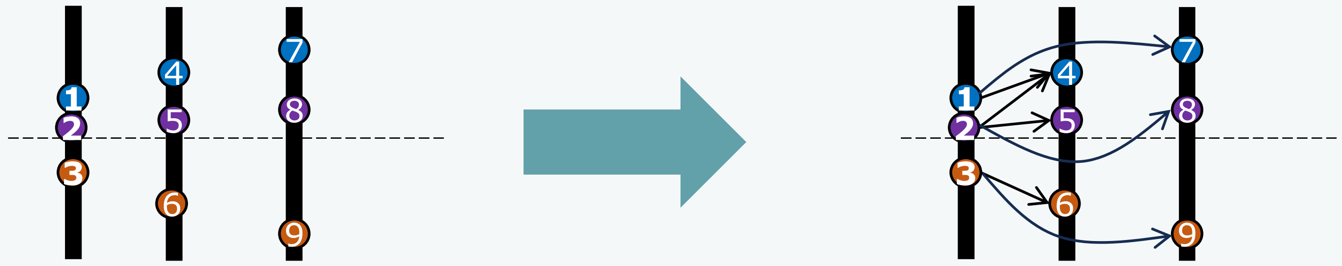
$$\vec{e_k} = \begin{pmatrix} e_{k1} \\ e_{k2} \\ e_{k3} \end{pmatrix}$$

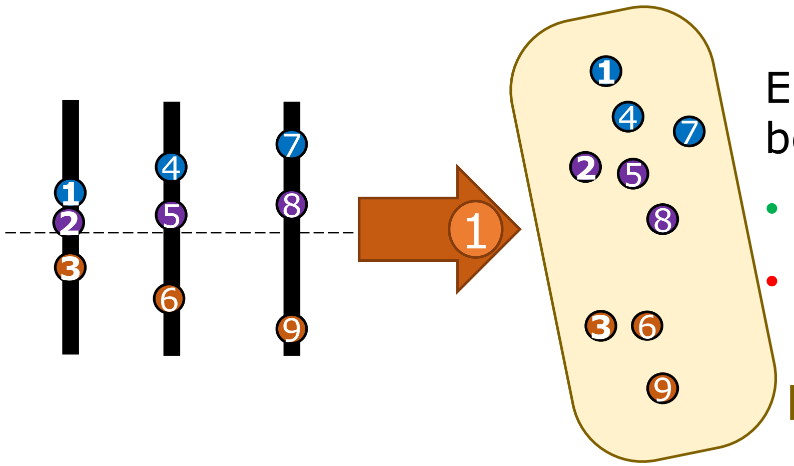
Stage 1: Graph Building

With an Embedding Network

Let's focus on connecting hits from plane p to plane $p+1$ and $p+2$

Goal

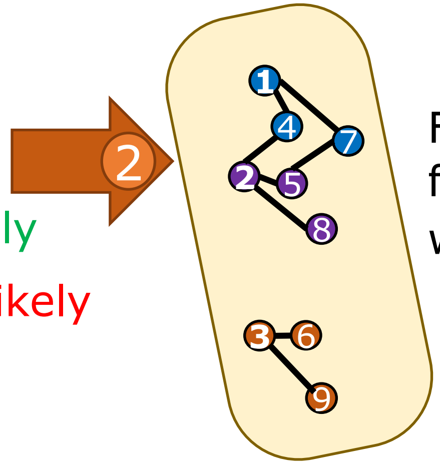




Embed each hit $\vec{x_k} \rightarrow \vec{e_k}$ so that, between nodes v_i and v_j:

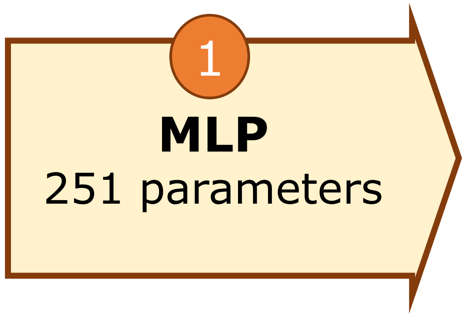
- $d_{ij}^2 = \left\|\vec{e_i} - \vec{e_j}\right\|^2 < 1$: edge is likely
- $d_{ij}^2 = \left\|\vec{e_i} - e_j\right\|^2 > 1$: edge is unlikely

Embedding space



For each node v_i in plane p, find $k_{\max}$ nearest nodes $\{v_j\}$ within sphere of radius $d_{\max}^2$

$$\vec{x_k} = \begin{pmatrix} r_k \\ \phi_k \\ z_k \end{pmatrix}$$



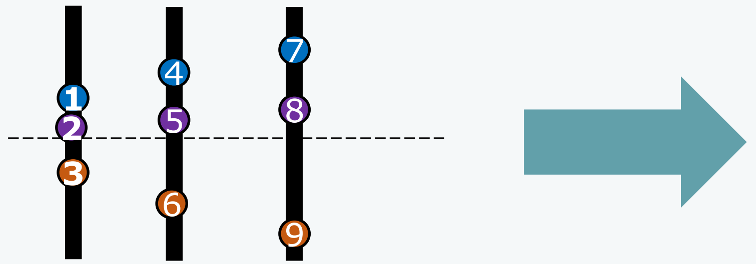
$$\vec{e_k} = \begin{pmatrix} e_{k1} \\ e_{k2} \\ e_{k3} \end{pmatrix}$$

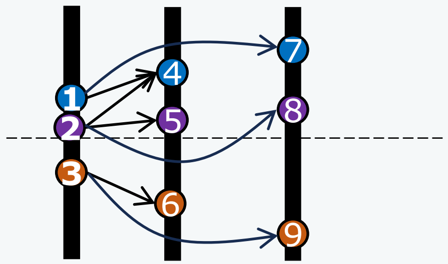
Stage 1: Graph Building

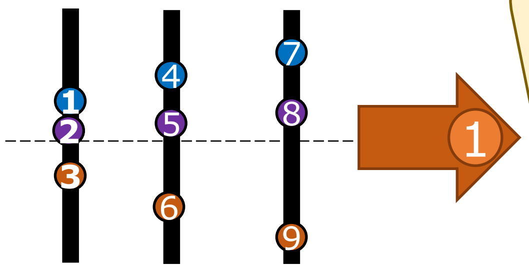
With an Embedding Network

Let's focus on connecting hits from plane p to plane $p+1$ and $p+2$

Goal



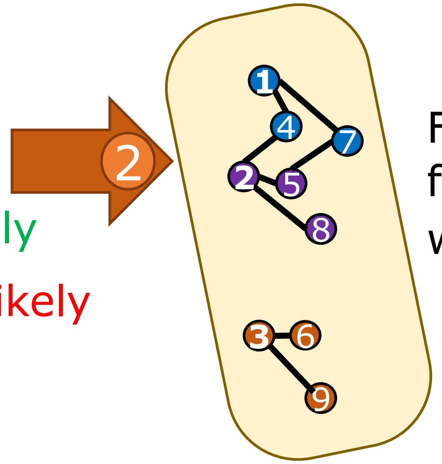




Embed each hit $\vec{x_k} \rightarrow \vec{e_k}$ so that, between nodes v_i and v_j:

- $d_{ij}^2 = \left\|\vec{e_i} - \vec{e_j}\right\|^2 < 1$: edge is likely
- $d_{ij}^2 = \left\|\vec{e_i} - e_j\right\|^2 > 1$: edge is unlikely

Embedding space



For each node v_i in plane p, find $k_{\max}$ nearest nodes $\{v_j\}$ within sphere of radius $d_{\max}^2$

$\vec{x_k} = \begin{pmatrix} r_k \\ \phi_k \\ z_k \end{pmatrix}$

1 **MLP** 251 parameters

$\vec{e_k} = \begin{pmatrix} e_{k1} \\ e_{k2} \\ e_{k3} \end{pmatrix}$

2

a. Apply $k_{\max}$-Nearest Neighbours (**kNN**) from plane p to planes $\{p+1, p+2\}$

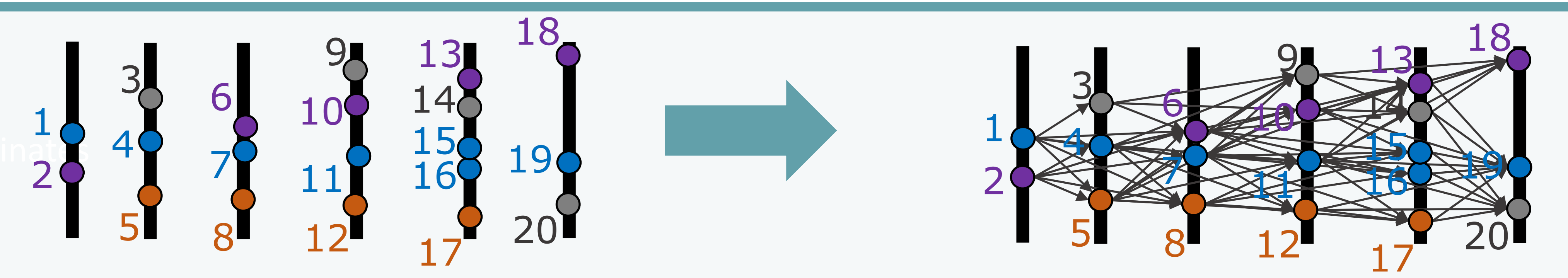
b. Only keep edges for which $d_{ij}^2 < d_{\max}^2$

$I_{\mathcal{E}} = \begin{pmatrix} 1 & 1 & 2 & 2 & 2 & 3 & 3 \\ 4 & 7 & 4 & 5 & 8 & 6 & 9 \end{pmatrix}$

Stage 1: Graph Building

With an Embedding Network

Goal

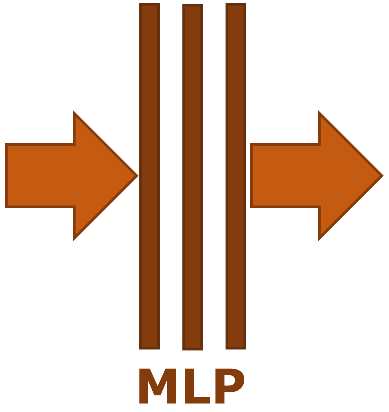


Recap'

1 **Embedding Network**

So that $d\left(v_i, v_j\right)^2 = \left\|\vec{e_i} - \vec{e_j}\right\|^2 < 1$
if $\left(v_i, v_j\right)$ likely to be an edge

$$X = \begin{pmatrix} r_1 & \phi_1 & z_1 \\ r_2 & \phi_2 & z_2 \\ r_3 & \phi_3 & z_3 \\ \vdots & \vdots & \vdots \\ r_{19} & \phi_{19} & z_{19} \\ r_{20} & \phi_{20} & z_{20} \end{pmatrix}$$



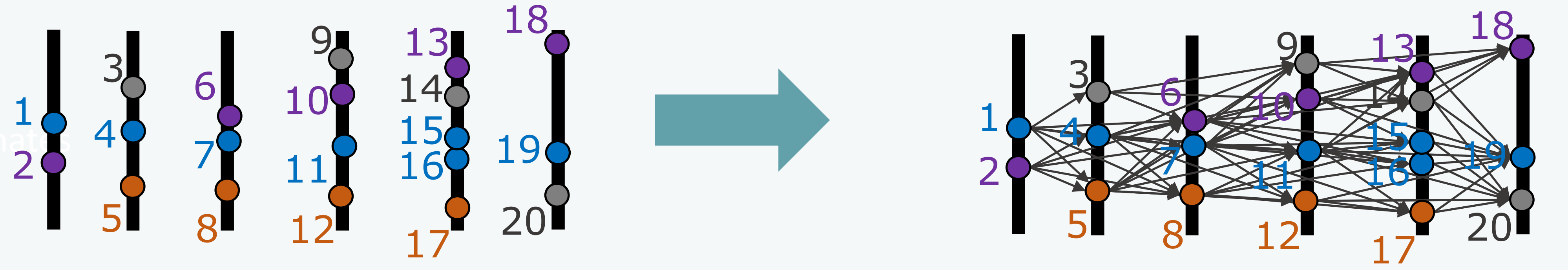
$$\boldsymbol{E} = \begin{pmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \\ \vdots & \vdots & \vdots \\ e_{19,1} & e_{19,3} & e_{19,3} \\ e_{20,1} & e_{20,2} & e_{20,3} \end{pmatrix}$$

Train with **hinge embedding loss** (see **annexe**)

Stage 1: Graph Building

With an Embedding Network

Goal

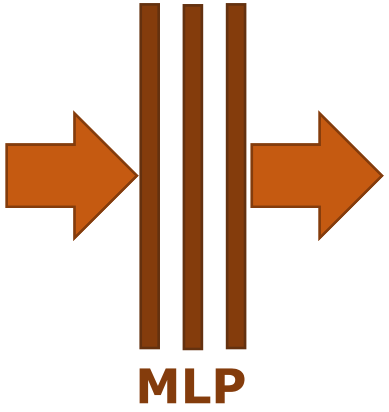


Recap'

1. **Embedding Network**

So that $d(v_i, v_j)^2 = \|\vec{e_i} - \vec{e_j}\|^2 < 1$
if (v_i, v_j) likely to be an edge

$$X = \begin{pmatrix} r_1 & \phi_1 & z_1 \\ r_2 & \phi_2 & z_2 \\ r_3 & \phi_3 & z_3 \\ \vdots & \vdots & \vdots \\ r_{19} & \phi_{19} & z_{19} \\ r_{20} & \phi_{20} & z_{20} \end{pmatrix}$$



$$E = \begin{pmatrix} e_{1,1} & e_{1,2} & e_{1,3} \\ e_{2,1} & e_{2,2} & e_{2,3} \\ e_{3,1} & e_{3,2} & e_{3,3} \\ \vdots & \vdots & \vdots \\ e_{19,1} & e_{19,3} & e_{19,3} \\ e_{20,1} & e_{20,2} & e_{20,3} \end{pmatrix}$$

Train with **hinge embedding loss** (see **annexe**)

2. **kNNs plane by plane**

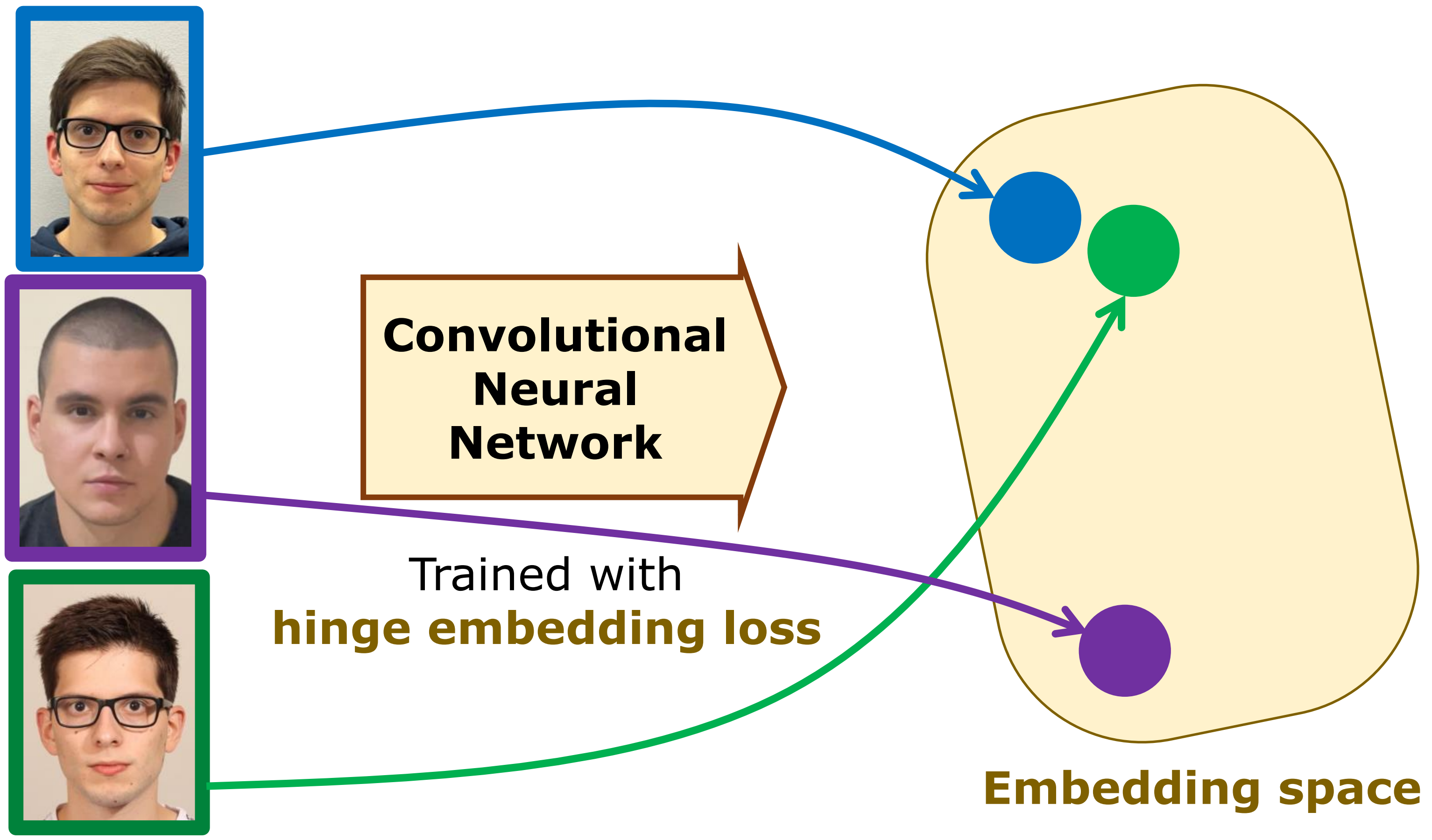
Use faiss library

a) Apply every plane $p \in \{1, \dots, n_{\text{planes}} - 1\}$,
apply $\boldsymbol{k}_{\text{max}}$NN from plane p to next 2 planes $p+1$ and $p+2$
b) Only keep edges for which $d(v_i, v_j) < \boldsymbol{d}^2_{\text{max}}$

⇒ 2 parameters to choose for inference: $\boldsymbol{k}_{\text{max}}$ and $\boldsymbol{d}^2_{\text{max}}$ (in annexe)

Siamese Network for One-Shot Face Recognition

Embedding network seen earlier can be used for **face recognition**



⇒ save only 1 picture / person in database

Paper (2015)

Stage 1: Graph Building

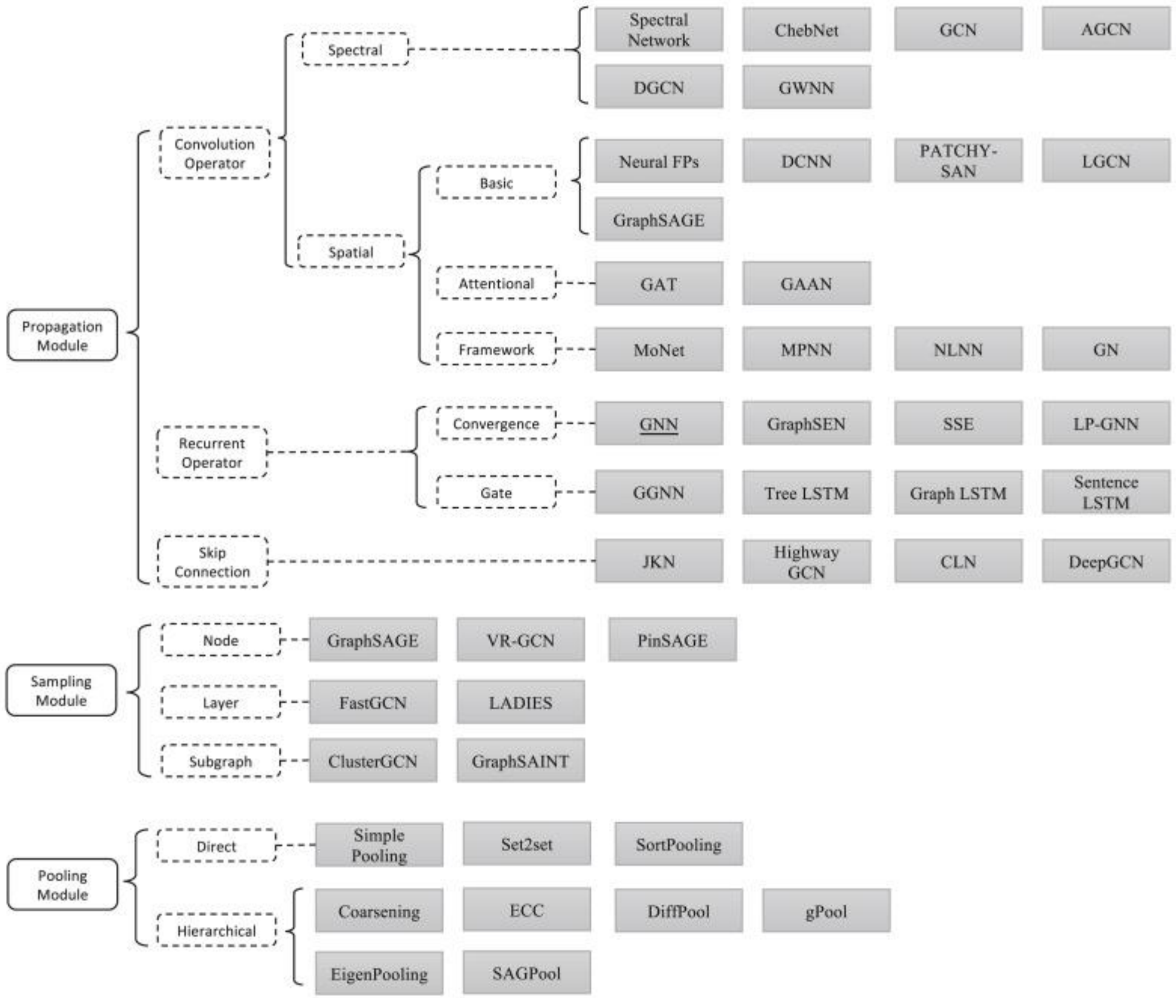
Choice of k_{max} and d^2_{max}

Stage 2: Graph Neural Networks

Definition

- **Graph Neural Network** (**GNN**): Neural Network architecture that operates on graphs
  → Problem needs **to be formulated with** a graph.

- **MANY** GNN architectures exist
  - [List of GNNs](#) implemented in PyTorch Geometric



[Graph neural networks: A review of methods and applications](#) (2020)

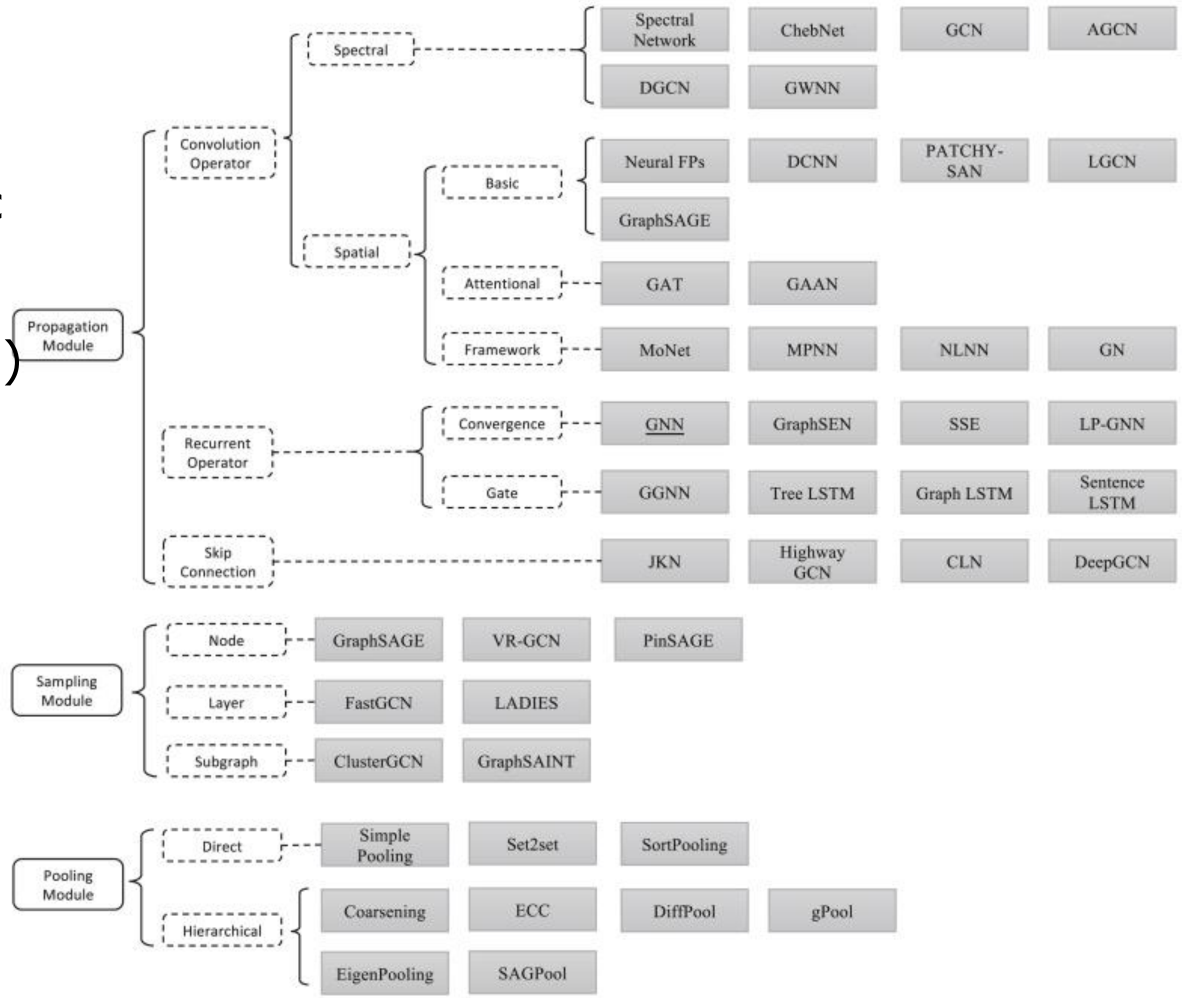
Stage 2: Graph Neural Networks

Definition

- **Graph Neural Network** (**GNN**): Neural Network architecture that operates on graphs
  → Problem needs **to be formulated with** a graph.

- **MANY** GNN architectures exist
  - List of GNNs implemented in PyTorch Geometric

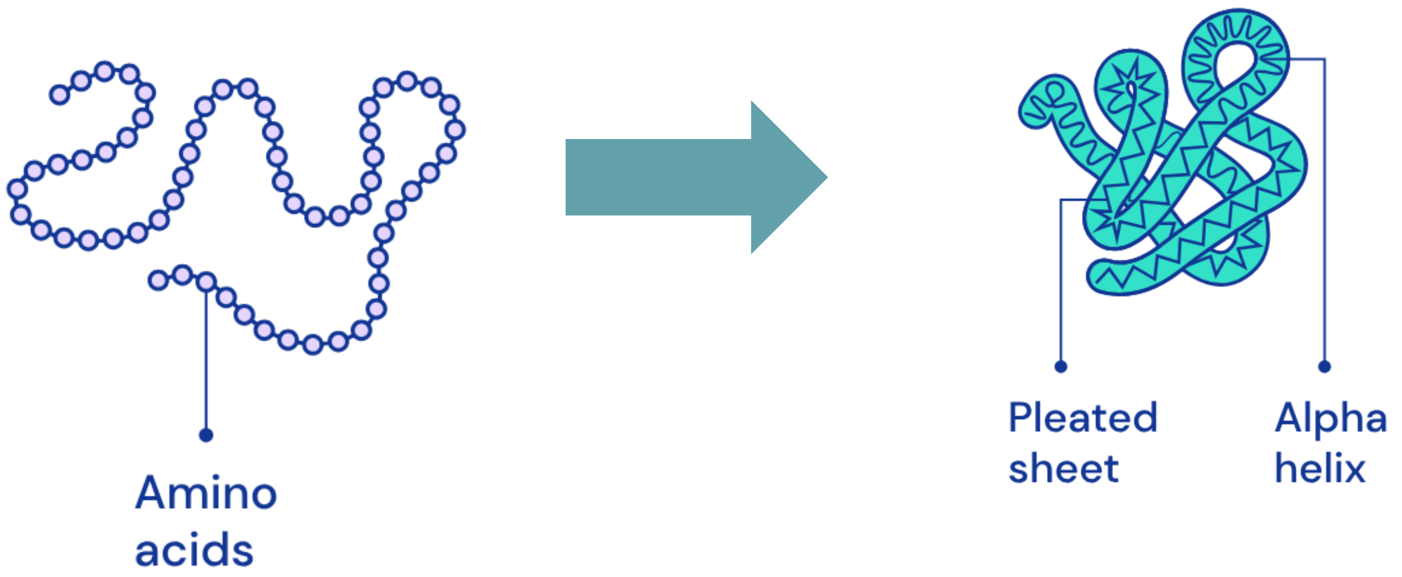
- GNN we used was described from this paper (2021)
  - Used by the Exa.TrkX collaboration
  - Follows the **Interaction Network** (**IN**) architecture
  - Can be described by the **Message Passing Neural Network** (**MPNN**) framework



Graph neural networks: A review of methods and applications (2020)

Examples of GNNs

Task level	**Example**	**Graph**	**Task**
Node	AlphaFold 2 (2020) **Input**: protein = amino acid sequence **Target**: 3D structure of folded protein	**Graph** = proteine • **Node**: amino acid • **Edge**: if in proximity	Predict **node coordinates**.
Edge			
Graph or subgraph			

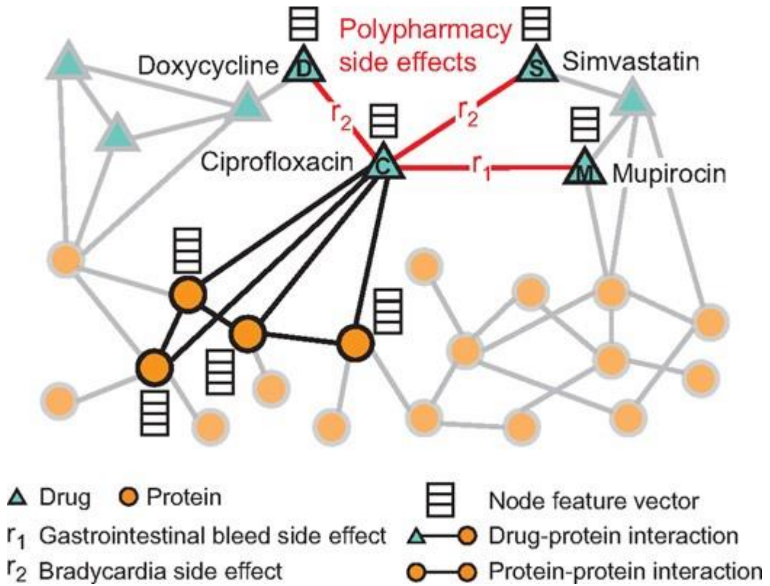


Description of these example cases are oversimplified.

Stage 2: Graph Neural Networks

Examples of GNNs

Task level	**Example**	**Graph**	**Task**
Node	AlphaFold 2 (2020) **Input**: protein = amino acid sequence **Target**: 3D structure of folded protein	**Graph =** proteine • **Node**: amino acid • **Edge**: if in proximity	Predict **node coordinates**.
Edge	Decagon (2018) **Input**: side effects between various drug combinations **Target**: *(unknown)* side effect between a 2 drug combinations	**Graph** of side effects • **Node**: drug • **Edge**: side effect	Predict **probability** that an edge corresponds to a given side effect.
Graph Or subgraph			



Description of these example cases are oversimplified.

Examples of GNNs

Task level	Example	Graph	Task
Node	AlphaFold 2 (2020) **Input**: protein = amino acid sequence **Target**: 3D structure of folded protein	**Graph =** proteine • **Node**: amino acid • **Edge**: if in proximity	Predict **node coordinates**.
Edge	Decagon (2018) **Input**: side effects between various drug combinations **Target**: *(unknown)* side effect between a 2 drug combinations	**Graph** of side effects • **Node**: drug • **Edge**: side effect	Predict **probability** that an edge corresponds to a given side effect.
Graph Or subgraph	Google Maps (2020) **Input**: road network **Target**: travel time of a chunk of road	**Graph** = road network • **Node**: route segment • **Edge**: if route segments are consecutive	Predict **travel time of a supersegment** = multiple adjacent segments.

Description of these example cases are oversimplified.

Stage 2: Graph Neural Networks

Examples of GNNs

Task level	**Example**	**Graph**	**Task**
Node	AlphaFold 2 (2020) **Input**: protein = amino acid sequence **Target**: 3D structure of folded protein	**Graph = proteine** • **Node**: amino acid • **Edge**: if in proximity	Predict **node coordinates**.
Edge	Decagon (2018) **Input**: side effects between various drug combinations **Target**: *(unknown)* side effect between a 2 drug combinations	**Graph** of side effects • **Node**: drug • **Edge**: side effect	Predict **probability** that an edge corresponds to a given side effect.
Graph Or subgraph	Google Maps (2020) **Input**: road network **Target**: travel time of a chunk of road	**Graph** = road network • **Node**: route segment • **Edge**: if route segments are consecutive	Predict **travel time of a supersegment** = multiple adjacent segments.

⚠ Description of these example cases are oversimplified.

Examples of GNNs

Task level	**Example**	**Graph**	**Task**
Node	AlphaFold 2 (2020) **Input**: protein = amino acid sequence **Target**: 3D structure of folded protein	**Graph** = proteine • **Node**: amino acid • **Edge**: if in proximity	Predict **node coordinates**.
Edge	Decagon (2018) **Input**: side effects between various drug combinations **Target**: *(unknown)* side effect between a 2 drug combinations	**Graph** of side effects • **Node**: drug • **Edge**: side effect	Predict **probability** that an edge corresponds to a given side effect.
Graph Or subgraph	Google Maps (2020) **Input**: road network **Target**: travel time of a chunk of road	**Graph** = road network • **Node**: route segment • **Edge**: if route segments are consecutive	Predict **travel time of a supersegment** = multiple adjacent segments.

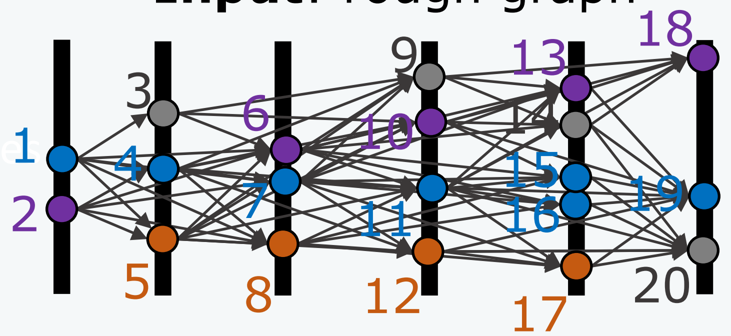
Other applications: recommender system (e.g., PinSage (2018)), fraud detection, novel molecule generation with desirable properties, physics simulation with many particles, weather forecasting [2023], etc.

Stage 2: Graph Neural Networks

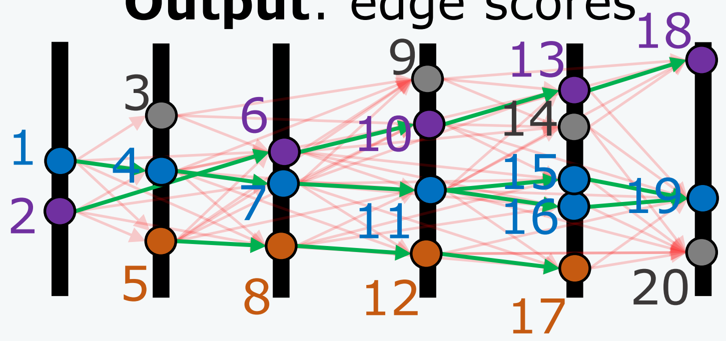
Problem Formulation

Goal

Input: rough graph



Output: edge scores



Input

Hit coordinates $X = \begin{pmatrix} r_1 & \phi_1 & z_1 \\ r_2 & \phi_2 & z_2 \\ r_3 & \phi_3 & z_3 \\ \vdots & \vdots & \vdots \\ r_{19} & \phi_{19} & z_{19} \\ r_{20} & \phi_{20} & z_{20} \end{pmatrix}$

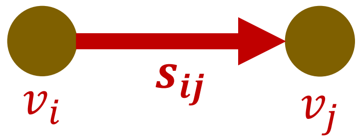
Edge indices $\mathrm{I}_{\mathcal{E}} = \begin{pmatrix} 1 & 1 & 1 & 1 & 1 & 1 & 2 & 2 & 2 & 2 & 2 & \cdots & 17 & 17 \\ 3 & 4 & 5 & 6 & 7 & 8 & 4 & 5 & 6 & 7 & 8 & \cdots & 19 & 20 \end{pmatrix}$

Edge scores $S = \begin{pmatrix} s_{1\to3} \\ s_{1\to4} \\ s_{1\to5} \\ s_{1\to6} \\ s_{1\to7} \\ s_{1\to8} \\ s_{2\to4} \\ s_{2\to5} \\ s_{2\to6} \\ s_{2\to7} \\ s_{2\to8} \\ \vdots \\ s_{17\to19} \\ s_{17\to20} \end{pmatrix} = \begin{pmatrix} 0 \\ 1 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \\ 1 \\ 0 \\ 0 \\ \vdots \\ 0 \\ 0 \end{pmatrix}$

Build graph | **Filter edges with GNN** | Find connected components

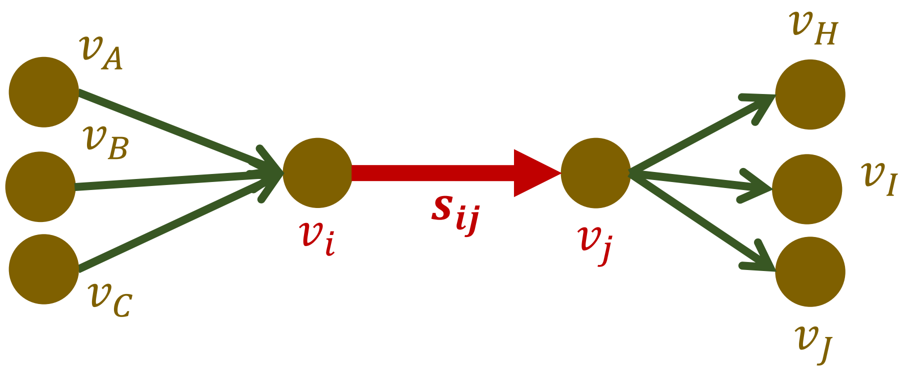
Explanation

Typical graph
around edge $i \to j$



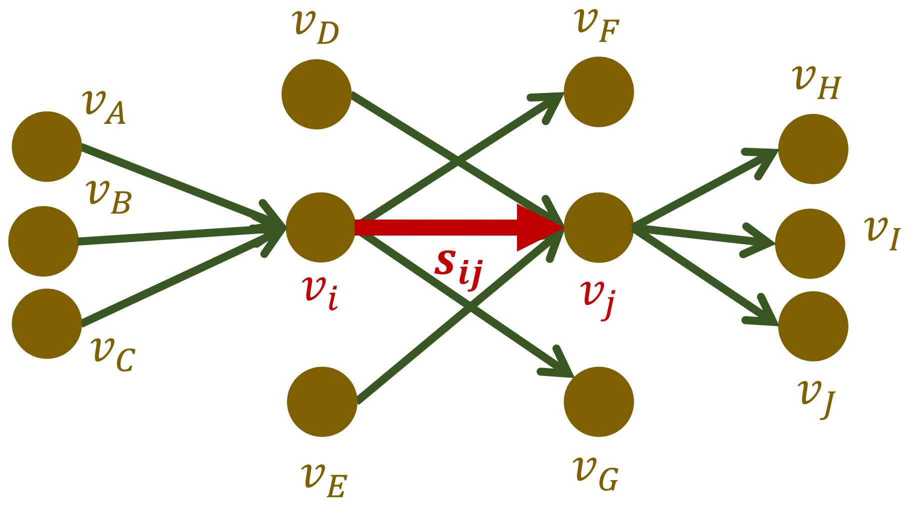
Explanation

Typical graph
around edge $i \rightarrow j$



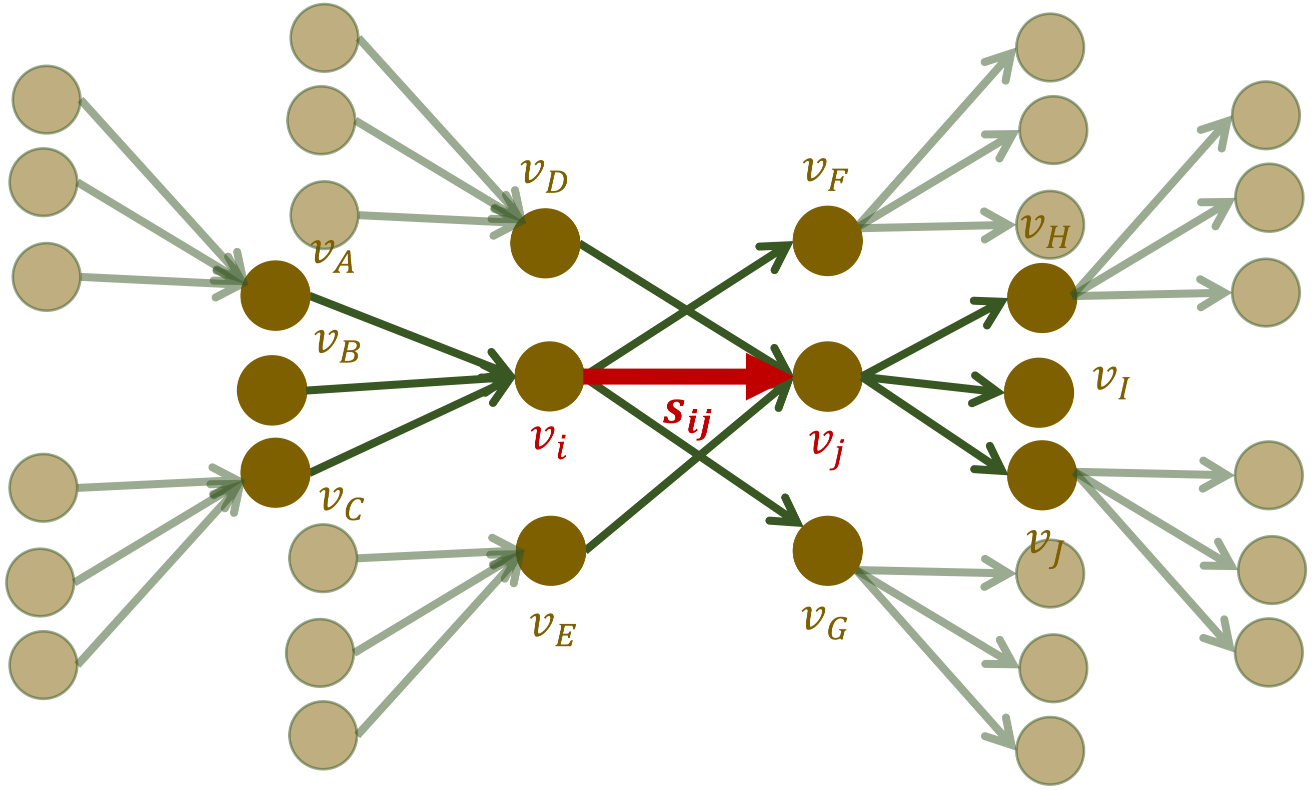
Explanation

Typical graph
around edge $i \rightarrow j$



Explanation

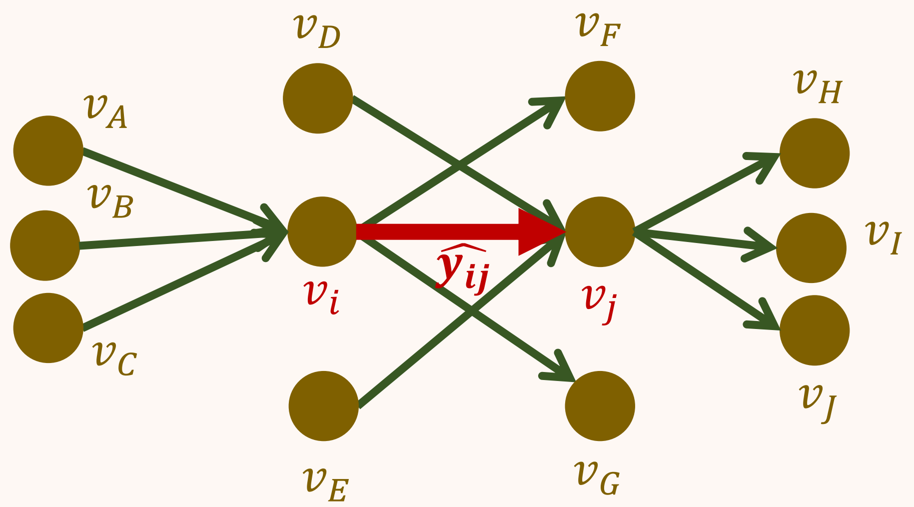
Typical graph
around edge $i \rightarrow j$



Stage 2: Graph Neural Networks

Explanation

Typical graph



Hit coordinates $X = \begin{pmatrix} \overrightarrow{x_1}^T \\ \vdots \\ \overrightarrow{x_N}^T \end{pmatrix}$ → **GNN** → **Edge score** $s_{ij} \in [0, 1]$

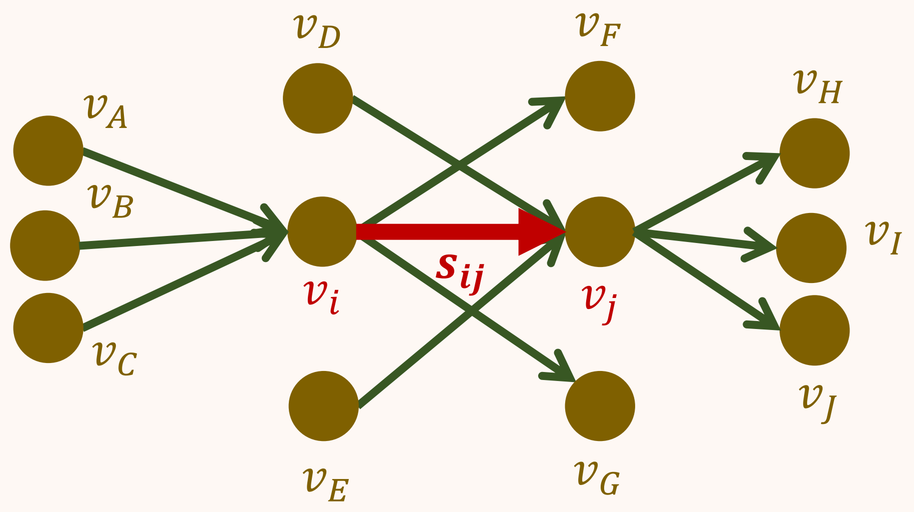
Edge indices $I_\mathcal{E}$

Trained with **binary cross entropy loss** or [**sigmoid focal loss**]()

Idea: work with intermediate « **edge encoding** »

Explanation

Typical graph



Edge features

$$\overrightarrow{f_{ij}} = \begin{Bmatrix} \overrightarrow{x_i} \\ \overrightarrow{x_j} \end{Bmatrix} = \begin{pmatrix} r_i \\ \phi_i \\ z_i \\ r_j \\ \phi_j \\ z_j \end{pmatrix}$$

Concatenated node features

Edge encoder

Edge encodings

$\overrightarrow{e_{ij}^0} \in \mathbb{R}^{128}$

Update edge encodings

Updated edge encodings

$\overrightarrow{e_{ij}^{\text{final}}} \in \mathbb{R}^{128}$

Edge classifier

Edge score

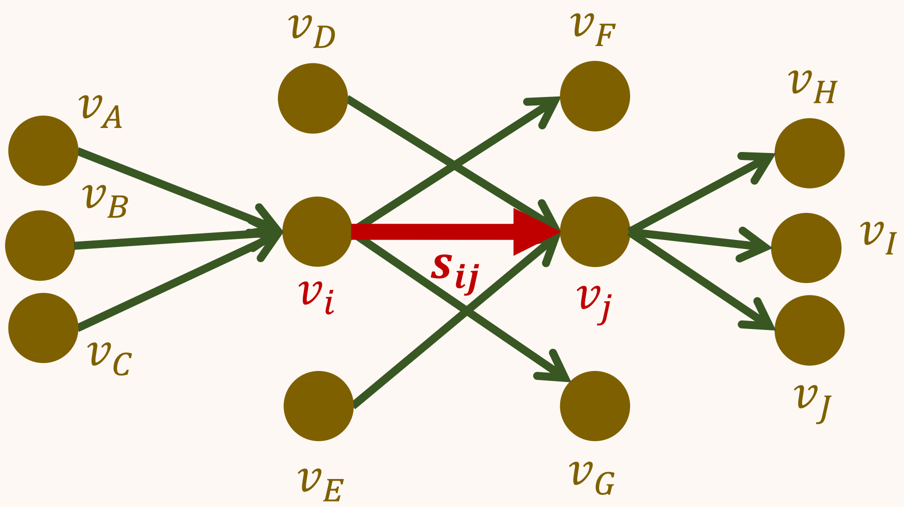
$s_{ij} \in [0, 1]$

Intermediate 128-dimensional edge encoding representation

Stage 2: Graph Neural Networks

Explanation

Typical graph



Edge encodings
$\overrightarrow{e_{ij}^{0}} \in \mathbb{R}^{128}$

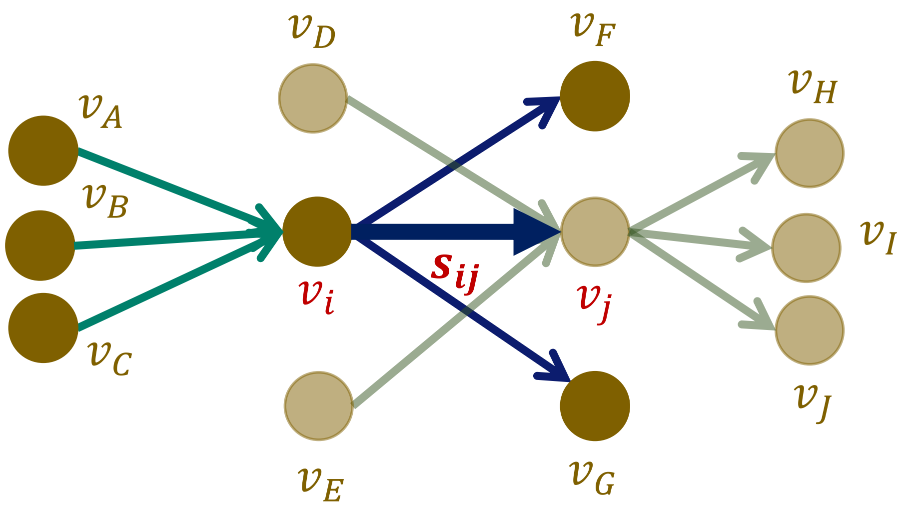
Update edge encodings

Updated edge encodings
$\overrightarrow{e_{ij}^{\text{final}}} \in \mathbb{R}^{128}$

Goal: update edge encoding $\overrightarrow{e_{ij}^{0}}$ according to edge encodings of connected edges

Stage 2: Graph Neural Networks

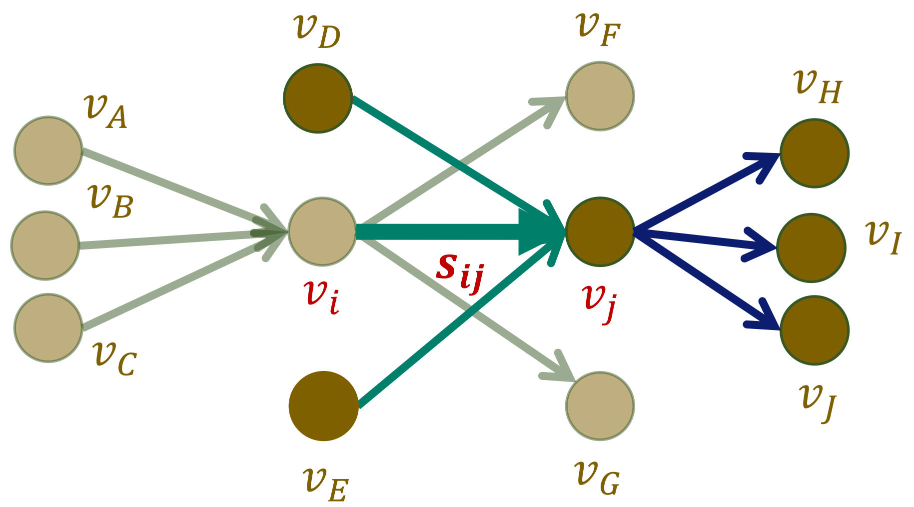
Explanation



1. **Build message for v_i:**
   Aggregate neighbour information
   - Node-ordering invariant
   - Separate incoming/outgoing nodes

$$\overrightarrow{m_i} = \left\{ \begin{array}{l} \overrightarrow{e^0_{A\to i}} + \overrightarrow{e^0_{B\to i}} + +\overrightarrow{e^0_{C\to i}} \\ \overrightarrow{e^0_{i\to F}} + \overrightarrow{e_{i\to j}} + \overrightarrow{e^0_{i\to G}} \end{array} \right\} \begin{array}{l} \longleftarrow \textbf{incoming} \\ \longleftarrow \textbf{outgoing} \end{array}$$

Explanation



1. **Build message for v_i:**
   Aggregate neigbhour information
   - Node-ordering invariant
   - Separate incoming/outgoing nodes

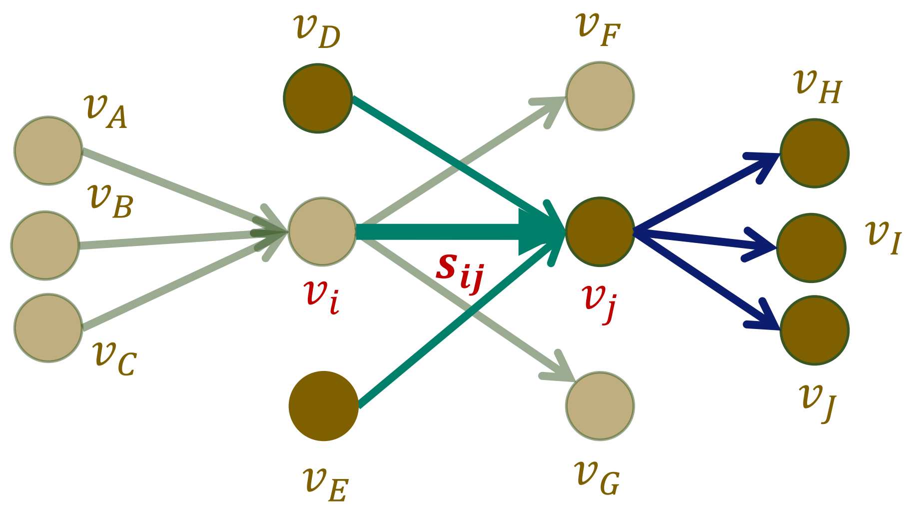
2. **Build message for v_j:**
   in a similar fashion

$$\overrightarrow{m_i} = \begin{cases} \overrightarrow{e^0_{A\to i}} + \overrightarrow{e^0_{B\to i}} + +\overrightarrow{e^0_{C\to i}} & \longleftarrow \textbf{incoming} \\ \overrightarrow{e^0_{i\to F}} + \overrightarrow{e_{i\to j}} + \overrightarrow{e^0_{i\to G}} & \longleftarrow \textbf{outgoing} \end{cases}$$

$$\overrightarrow{m_j} = \begin{cases} \overrightarrow{e^0_{D\to j}} + \overrightarrow{e^0_{i\to j}} + +\overrightarrow{e^0_{E\to j}} & \longleftarrow \textbf{incoming} \\ \overrightarrow{e^0_{j\to H}} + \overrightarrow{e_{j\to I}} + \overrightarrow{e^0_{j\to J}} & \longleftarrow \textbf{outgoing} \end{cases}$$

Stage 2: Graph Neural Networks

Explanation



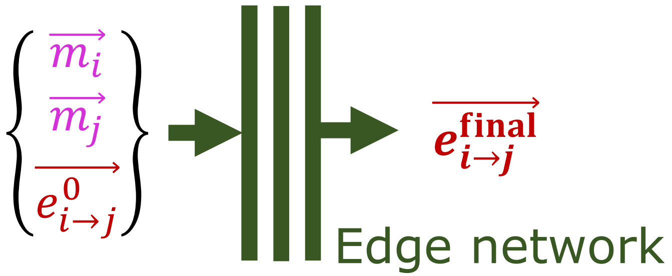
1. **Build message for v_i:**
   Aggregate neigbhour information
   - Node-ordering invariant
   - Separate incoming/outgoing nodes

$$\overrightarrow{m_i} = \begin{Bmatrix} \overrightarrow{e^0_{A\to i}} + \overrightarrow{e^0_{B\to i}} + +\overrightarrow{e^0_{C\to i}} \\ \overrightarrow{e^0_{i\to F}} + \overrightarrow{e_{i\to j}} + \overrightarrow{e^0_{i\to G}} \end{Bmatrix} \begin{array}{l} \longleftarrow \textbf{incoming} \\ \longleftarrow \textbf{outgoing} \end{array}$$

2. **Build message for v_j:**
   in a similar fashion

$$\overrightarrow{m_j} = \begin{Bmatrix} \overrightarrow{e^0_{D\to j}} + \overrightarrow{e^0_{i\to j}} + +\overrightarrow{e^0_{E\to j}} \\ \overrightarrow{e^0_{j\to H}} + \overrightarrow{e_{j\to I}} + \overrightarrow{e^0_{j\to J}} \end{Bmatrix} \begin{array}{l} \longleftarrow \textbf{incoming} \\ \longleftarrow \textbf{outgoing} \end{array}$$

3. **Infer** $\overrightarrow{\boldsymbol{e}^{\mathbf{final}}_{i\to j}}$ from $\overrightarrow{m_i}$, $\overrightarrow{m_j}$ and $\overrightarrow{e^0_{i\to j}}$ using a **MLP**

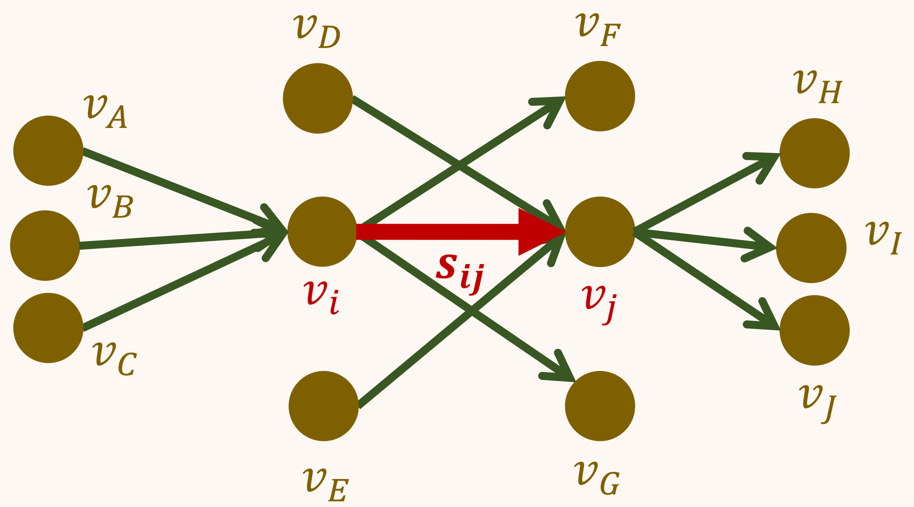


This is called **message passing**.

Stage 2: Graph Neural Networks

Explanation

Typical graph



Edge encodings
$\overrightarrow{e_{ij}^{0}} \in \mathbb{R}^{128}$

Update edge encodings

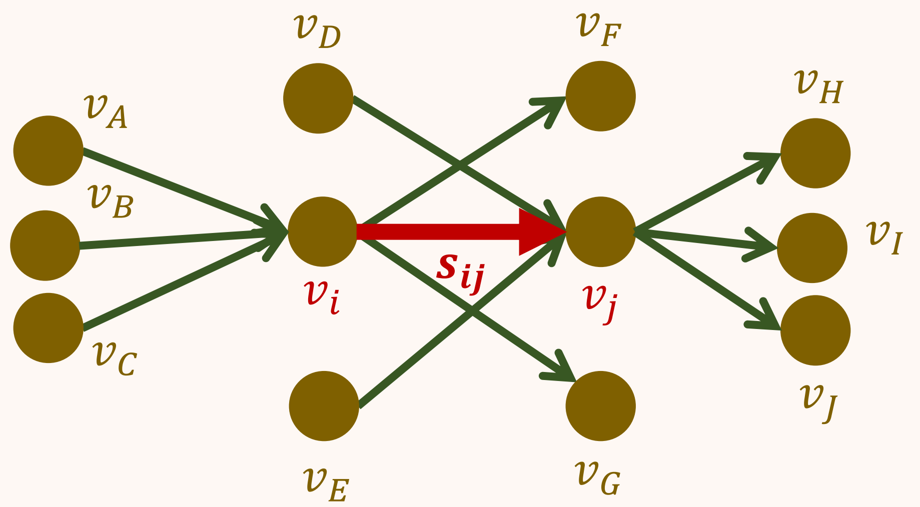
Updated edge encodings
$\overrightarrow{e_{ij}^{\text{final}}} \in \mathbb{R}^{128}$

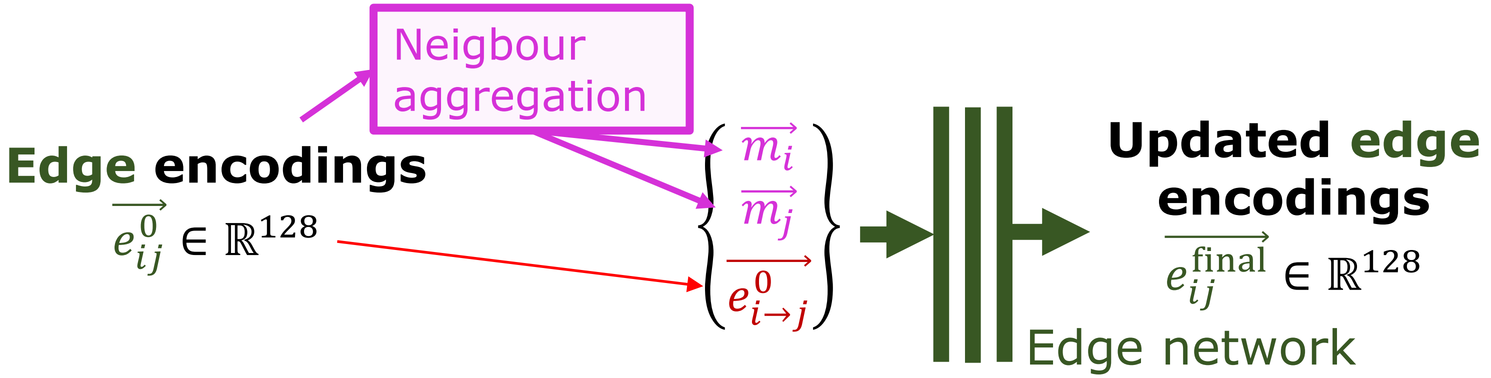
Goal: update edge encoding $\overrightarrow{e_{ij}^{0}}$ according to edge encodings of connected edges

Stage 2: Graph Neural Networks

Explanation

Typical graph



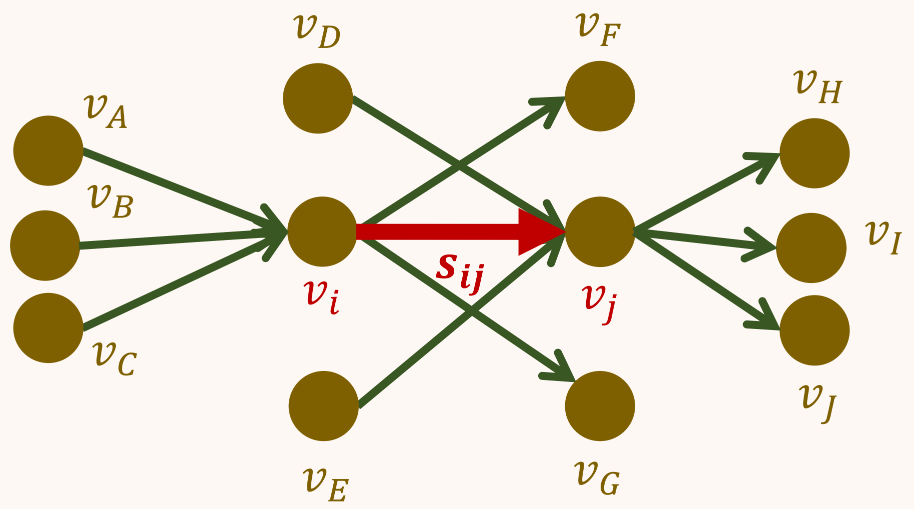


Goal: update edge encoding $\overrightarrow{e_{ij}^0}$ according to edge encodings of connected edges

Stage 2: Graph Neural Networks

Explanation

Typical graph



Edge features

$$\overrightarrow{f_{ij}} = \begin{Bmatrix} \overrightarrow{x_i} \\ \overrightarrow{x_j} \end{Bmatrix} = \begin{pmatrix} r_i \\ \phi_i \\ z_i \\ r_j \\ \phi_j \\ z_j \end{pmatrix}$$

Concatenated node features

Edge encoder

Edge encodings

$\overrightarrow{e_{ij}^0} \in \mathbb{R}^{128}$

Neigbour aggregation

$$\begin{Bmatrix} \overrightarrow{m_i} \\ \overrightarrow{m_j} \\ \overrightarrow{e_{i \to j}^0} \end{Bmatrix}$$

Edge network

Updated edge encodings

$\overrightarrow{e_{ij}^{\text{final}}} \in \mathbb{R}^{128}$

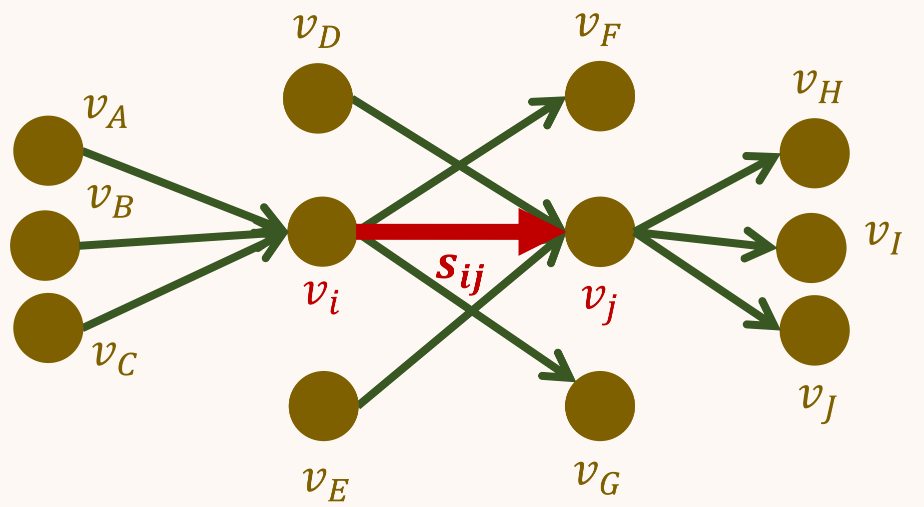
Edge classifier

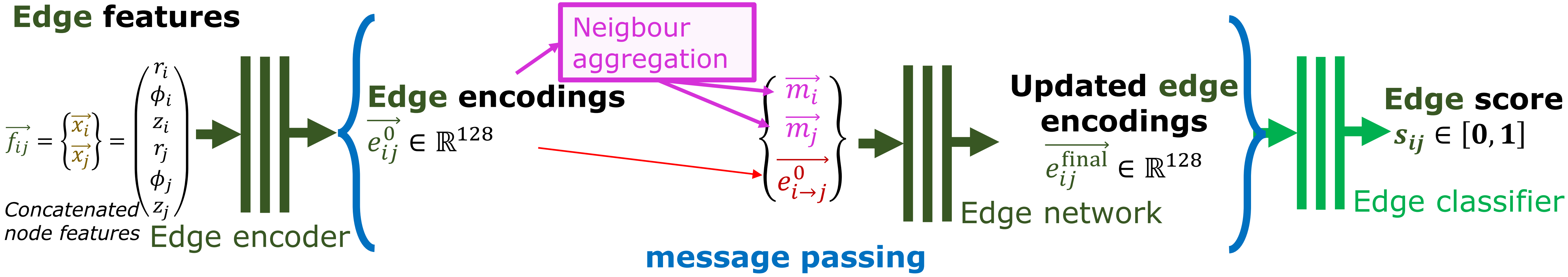
Edge score

$s_{ij} \in [\mathbf{0}, \mathbf{1}]$

Explanation

Typical graph

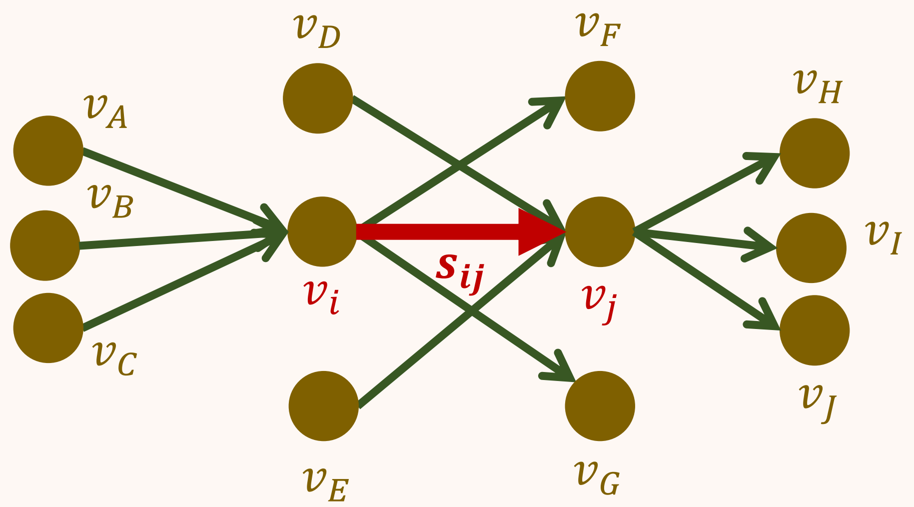


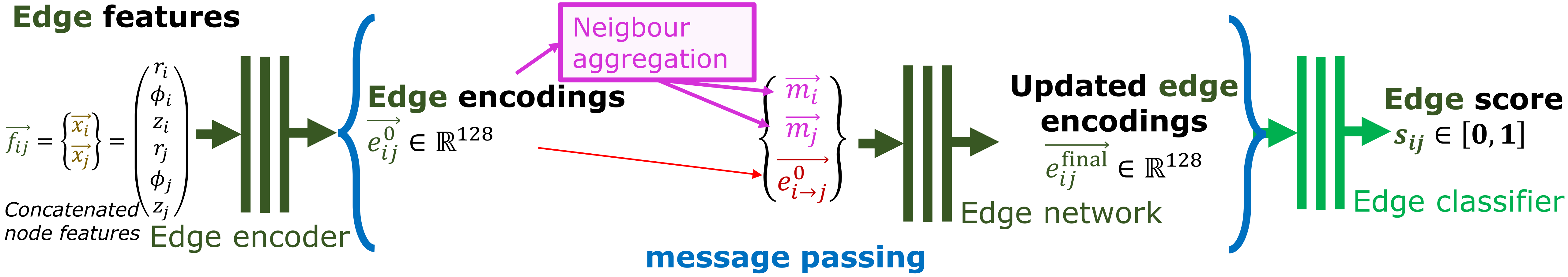


Stage 2: Graph Neural Networks

Explanation

Typical graph





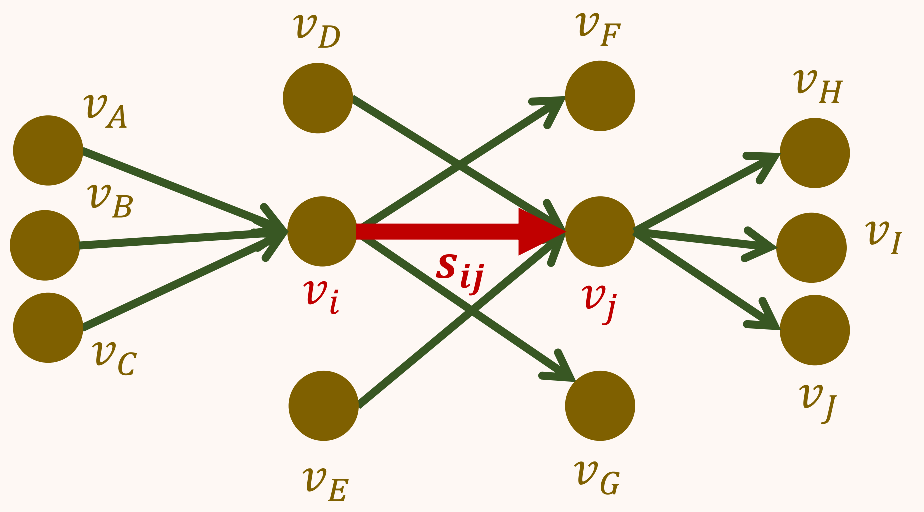
However, edge encoding only updated w.r.t. **immediate neighbours**.
→ How to update edge encoding as a function of **indirect** neighbours?

Just repeat message passing

Stage 2: Graph Neural Networks

Explanation

Typical graph



Edge features

$$\vec{f_{ij}} = \begin{Bmatrix} \vec{x_i} \\ \vec{x_j} \end{Bmatrix} = \begin{pmatrix} r_i \\ \phi_i \\ z_i \\ r_j \\ \phi_j \\ z_j \end{pmatrix}$$

Concatenated node features

Edge encoder

Edge encodings

$\vec{e_{ij}^{0}} \in \mathbb{R}^{128}$

Neigbour aggregation

$$\begin{Bmatrix} \vec{m_i} \\ \vec{m_j} \\ \vec{e^{0}_{i\to j}} \end{Bmatrix}$$

Edge network

Updated edge encodings

$\vec{e^{\text{final}}_{ij}} \in \mathbb{R}^{128}$

message passing

Edge classifier

Edge score

$s_{ij} \in [0,1]$

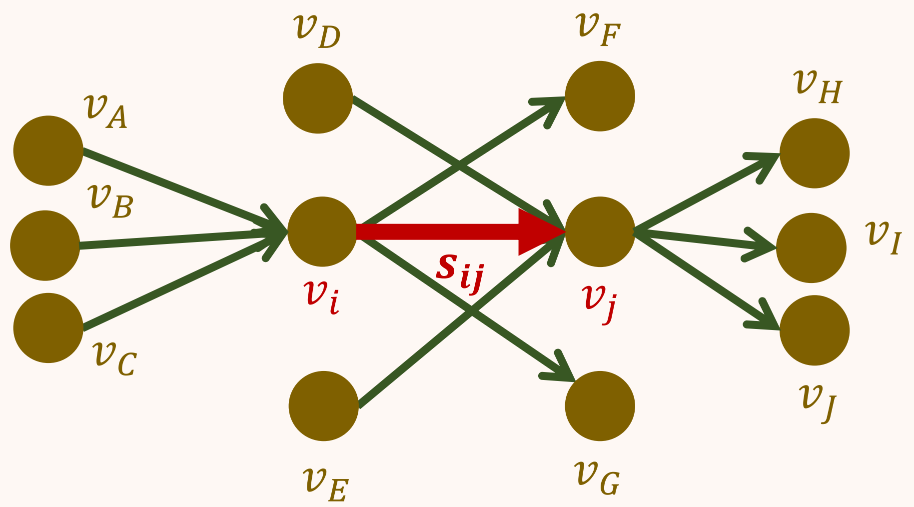
However, edge encoding only updated w.r.t. **immediate neighbours**.
→ How to update edge encoding as a function of **indirect** neighbours?

Just repeat
message passing

Stage 2: Graph Neural Networks

Explanation

Typical graph



Edge features

$\overrightarrow{f_{ij}} = \begin{Bmatrix} \overrightarrow{x_i} \\ \overrightarrow{x_j} \end{Bmatrix} = \begin{pmatrix} r_i \\ \phi_i \\ z_i \\ r_j \\ \phi_j \\ z_j \end{pmatrix}$

Concatenated node features

Edge encoder

Neigbour aggregation

Edge encodings

$\overrightarrow{e_{ij}^{k-1}} \in \mathbb{R}^{128}$

$\begin{Bmatrix} \overrightarrow{m_i^{k-1}} \\ \overrightarrow{m_j^{k-1}} \\ \overrightarrow{e_{i\to j}^{k-1}} \end{Bmatrix}$

Edge network

Updated edge encodings

$\overrightarrow{e_{ij}^{\mathrm{k}}} \in \mathbb{R}^{128}$

message passing, repeat for $k \in \{1, \dots, 5\}$

Edge score

$s_{ij} \in [0, 1]$

Edge classifier

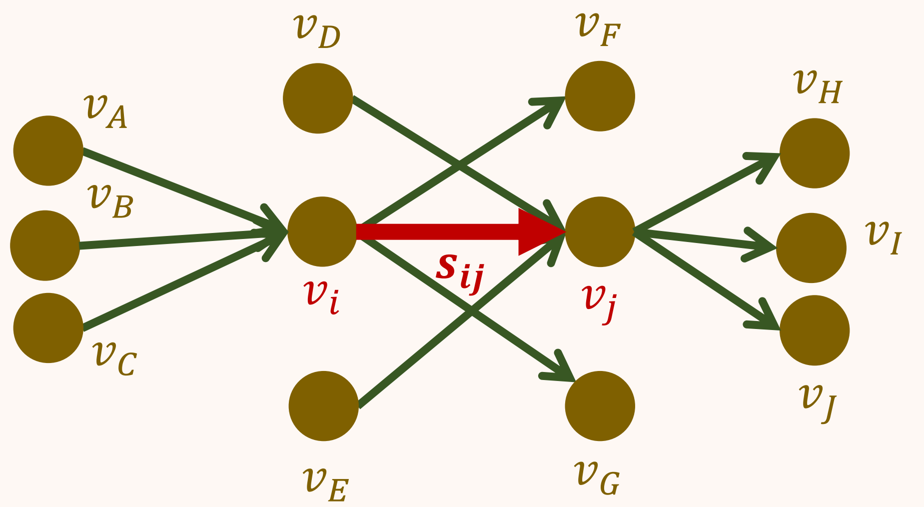
However, edge encoding only updated w.r.t. **immediate neighbours**.
→ How to update edge encoding as a function of **indirect** neighbours?

Just repeat
message passing

Stage 2: Graph Neural Networks

Explanation

Typical graph



Edge features

$\overrightarrow{f_{ij}} = \begin{Bmatrix} \overrightarrow{x_i} \\ \overrightarrow{x_j} \end{Bmatrix} = \begin{pmatrix} r_i \\ \phi_i \\ z_i \\ r_j \\ \phi_j \\ z_j \end{pmatrix}$

Concatenated node features

Edge encoder

Edge encodings

$\overrightarrow{e_{ij}^{k-1}} \in \mathbb{R}^{128}$

Neigbour aggregation

$\begin{Bmatrix} \overrightarrow{m_i^{k-1}} \\ \overrightarrow{m_j^{k-1}} \\ \overrightarrow{e_{i\to j}^{k-1}} \end{Bmatrix}$

Edge network

Updated edge encodings

$\overrightarrow{e_{ij}^{\mathrm{k}}} \in \mathbb{R}^{128}$

Edge classifier

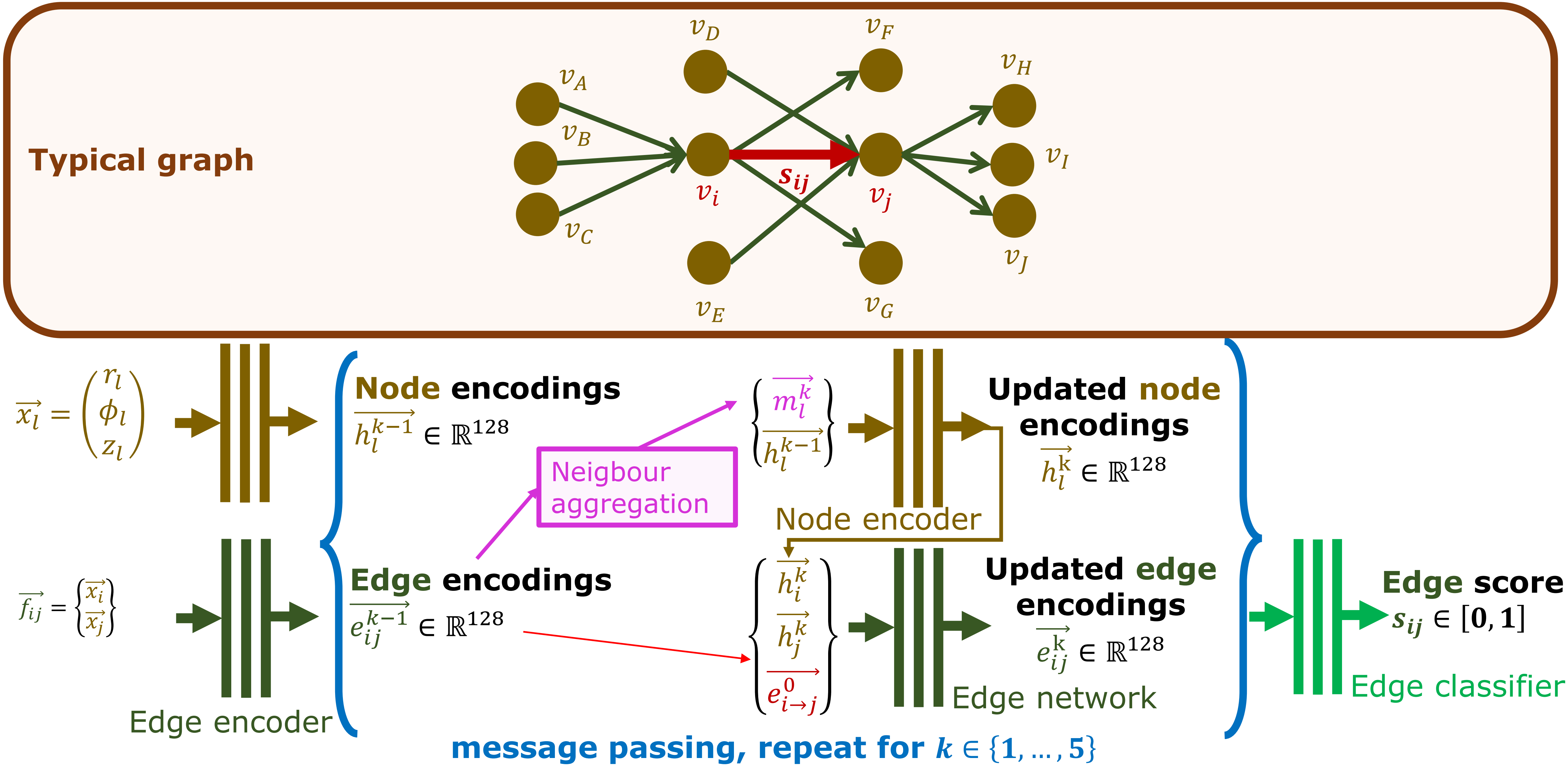
message passing, repeat for $k \in \{1, \ldots, 5\}$

Message $\overrightarrow{m_i}$
- Is **node-related**
- **Does not contain any node information**!

Combine it with **node encodings**.

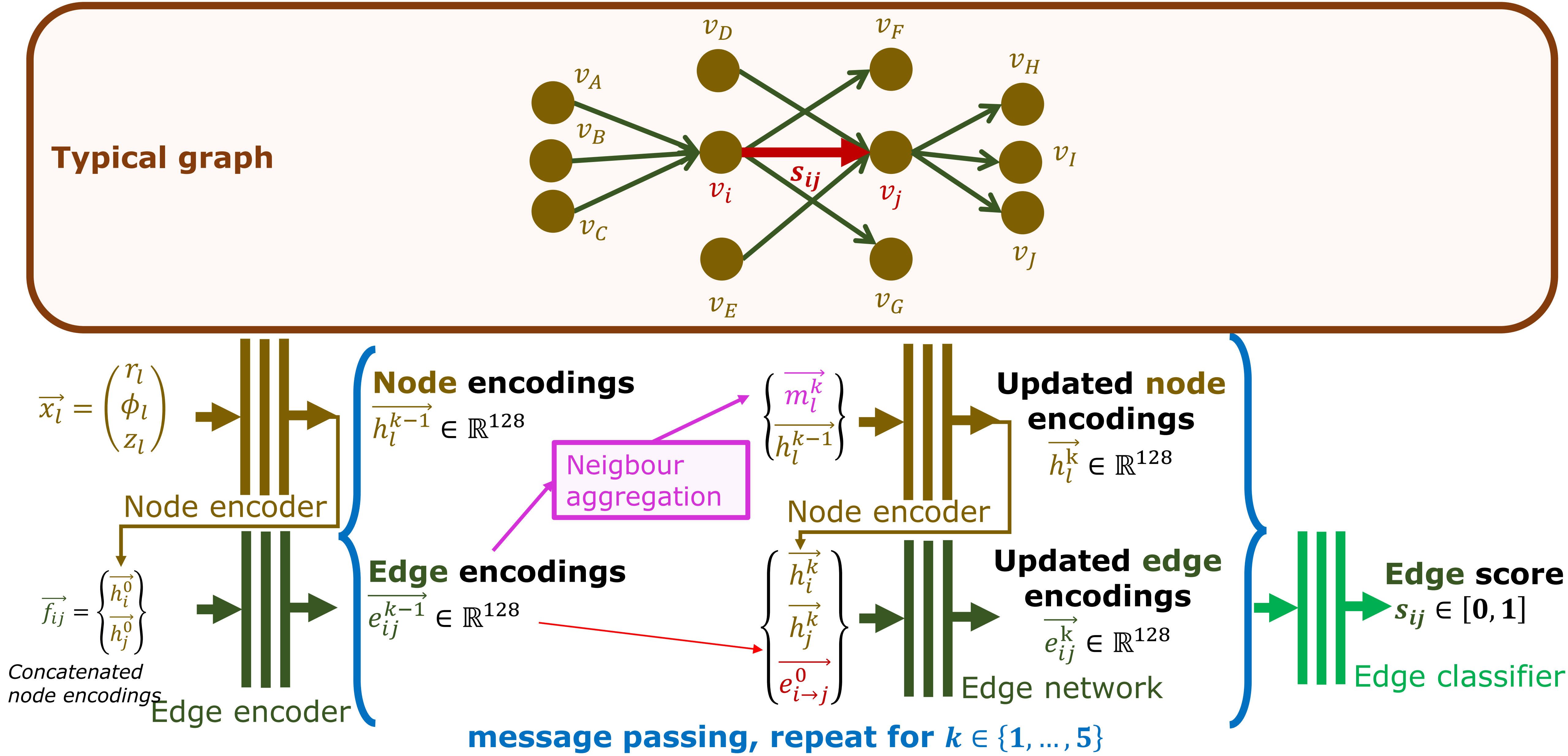
Stage 2: Graph Neural Networks

Explanation



Stage 2: Graph Neural Networks

Explanation



Shared hits

Electron Performance

Graph Building → GNN to filter edges → Build tracks from graph

But if you do this… **track efficiency on** long electrons is **terrible!**

Metric	**Default GPU algorithm**	**ETX4VELO**
Efficiency	**98.17%**	**46.23%**

(evaluated on 1000 events)

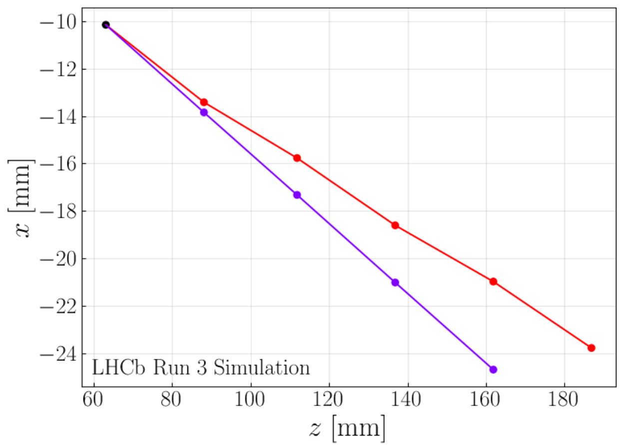
Shared hits

Electron Performance

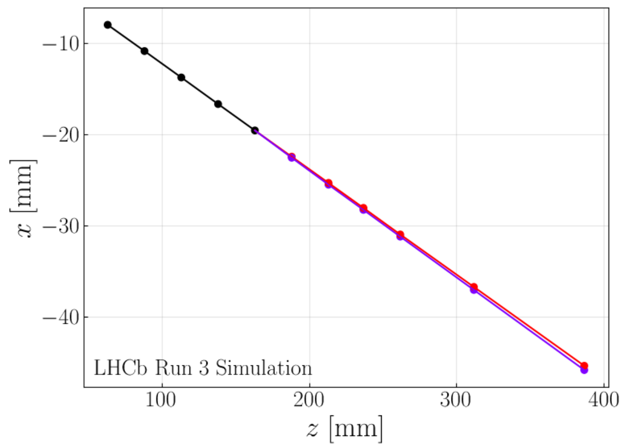
Observations

- $\sim$ 55 % electrons share hits with another electron
- The 2 electrons share ≥ 1 hit(s) before splitting up

Example 1: share the first hit only



Example 2: share several hits before splitting up

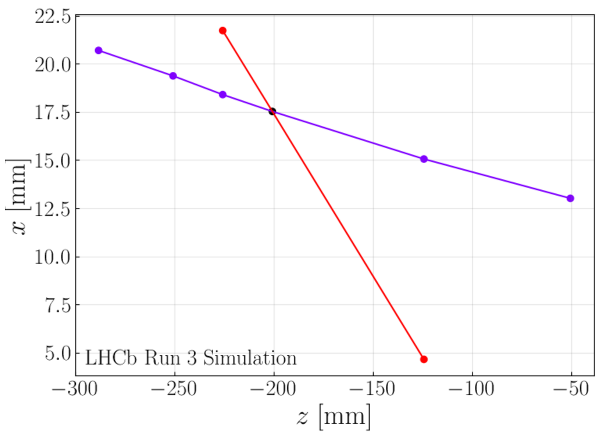


⇒ the **connected component algorithm** consider the **2** electron tracks as a **single** track

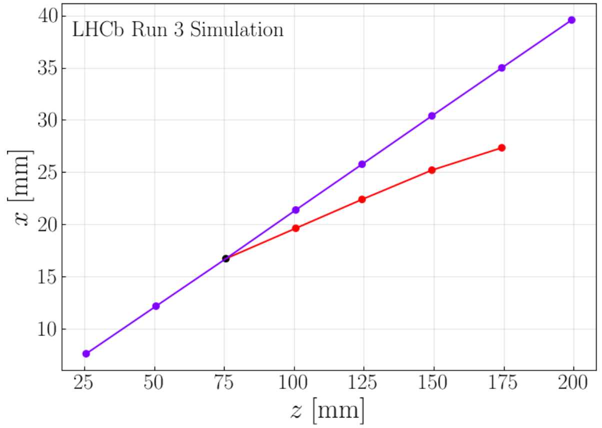
Shared hits

Other Tracks With Shared Hits

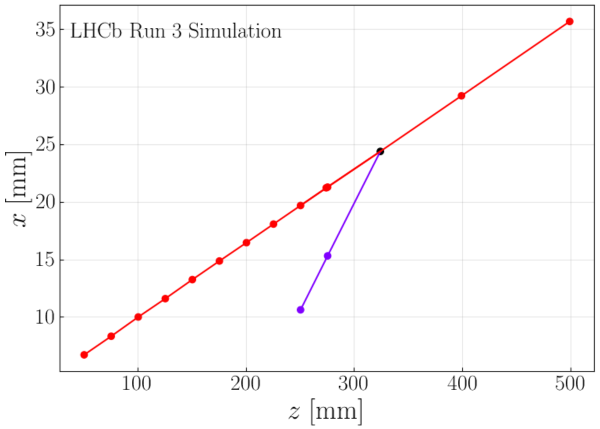
- **Tracks crossing** (> 524 in 1000 events)



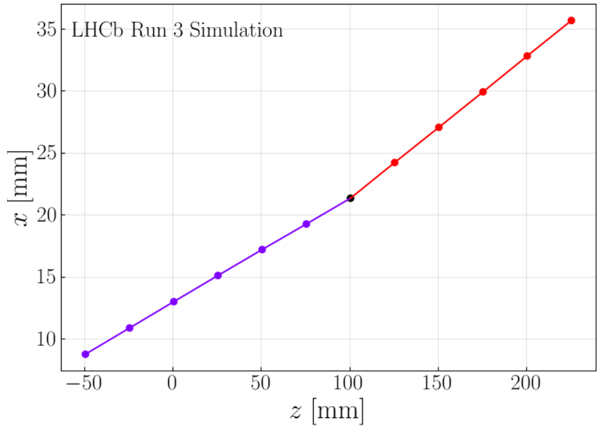
- **Track starts on a shared hit**



- **Track ends on a shared hit**

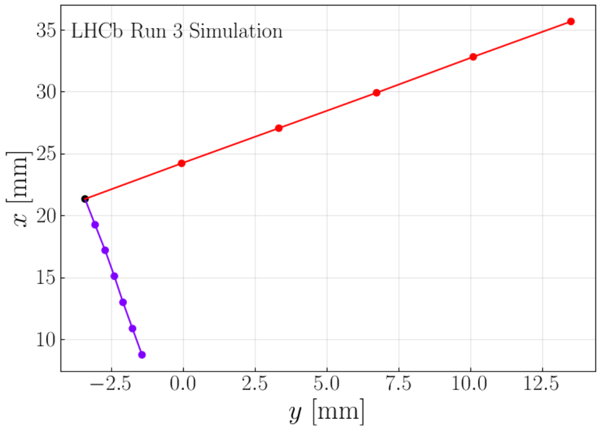


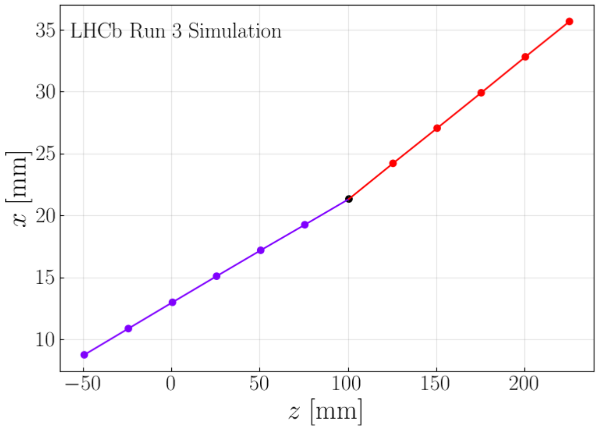
- **The last hit of a track is the first hit of another track** (>141 in 1000 events)



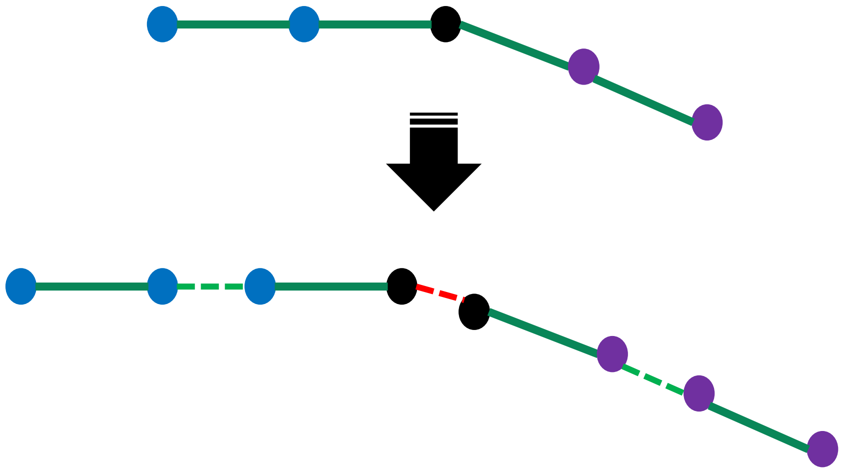
Shared hits

Edge-Edge Connections





In this case, one cannot even guess that there are *possibly* 2 tracks!

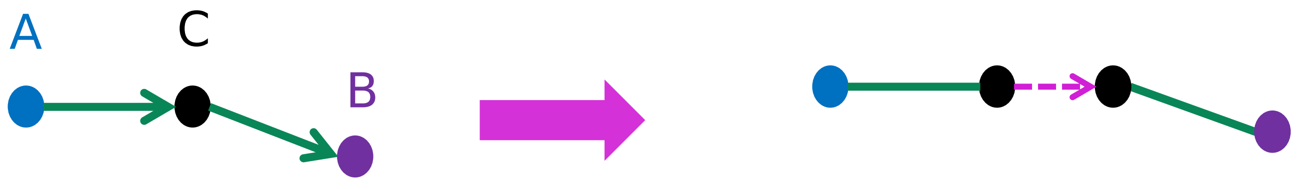


Hit-hit connection is not enough
⇒ need **edge-edge connections**

Edge-Edge Connections

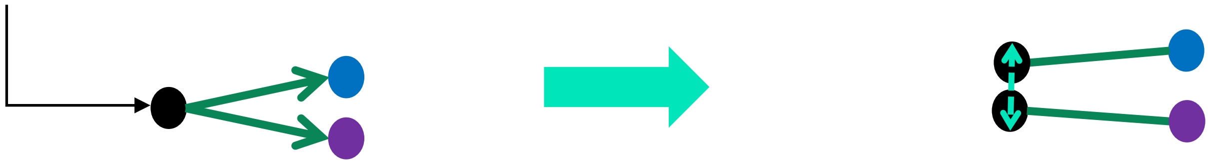
3 kind of **edge-edge connections** (or ***triplets***) are possible

Articulation

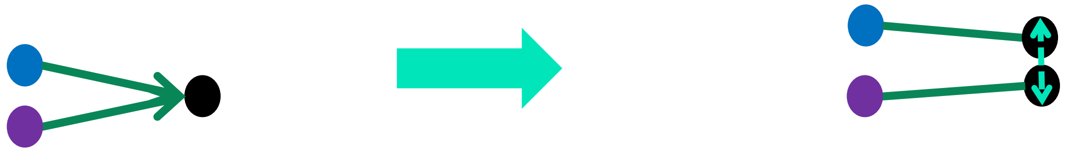


Could be a shared hit

Left elbow



Right elbow



Shared hits

Updated Pipeline

New simplified example to take into account **shared hits**

Goal

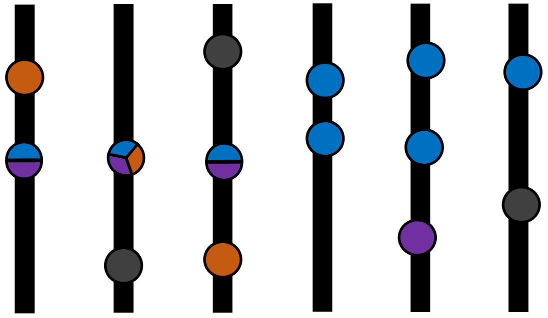
First 2 steps are the same as before

Shared hits

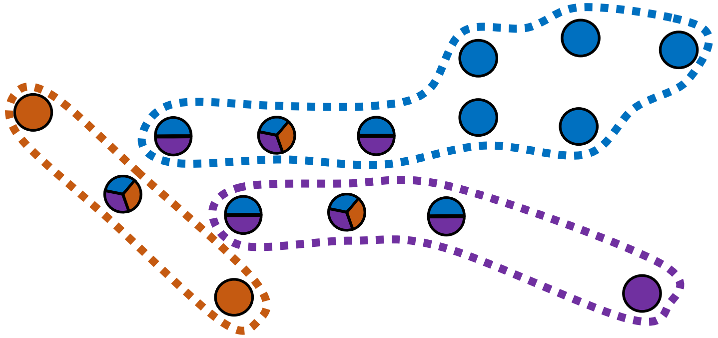
Updated Pipeline

New simplified example to take into account **shared hits**

Goal

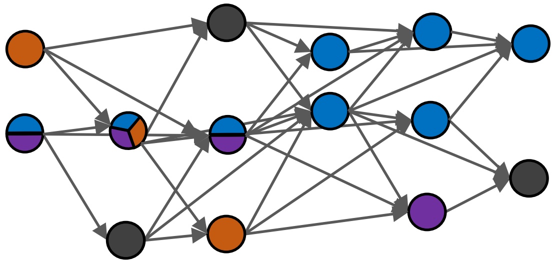


Track Finding

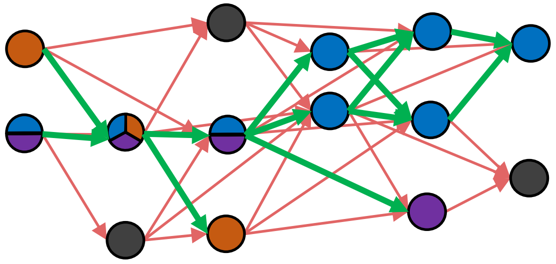


First 2 steps are the same as before

1. Build a **"rough" graph**
   → Embedding Network + kNN



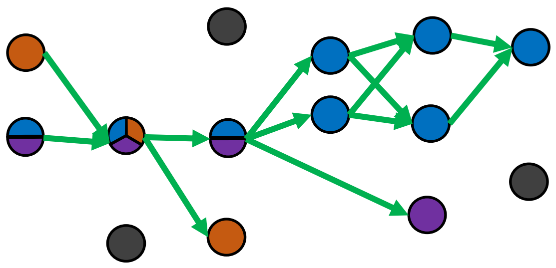
2. **Classify the edges** as genuine or fake
   → Graph Neural Network



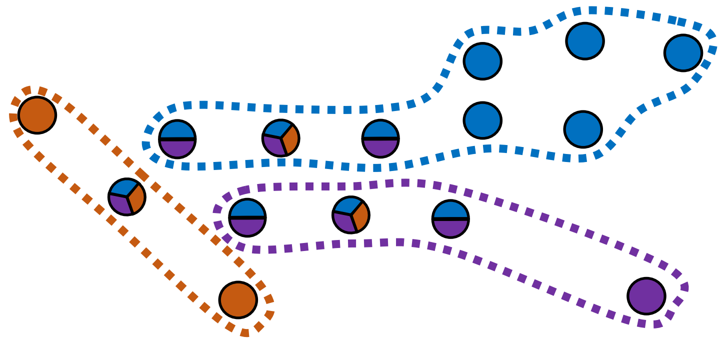
Shared hits

Updated Pipeline

Goal



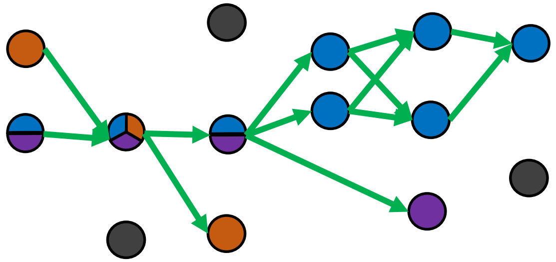
Handle Shared Hits



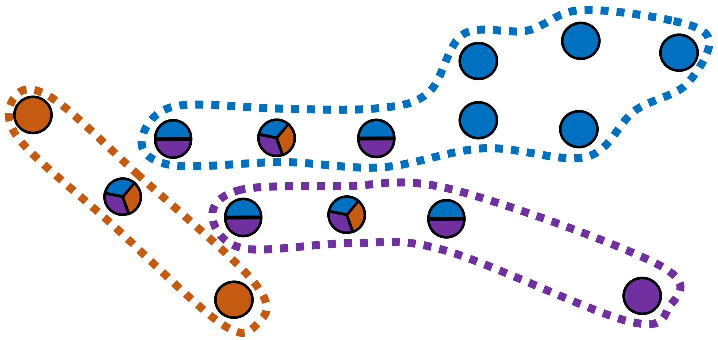
Shared hits

Updated Pipeline

Goal

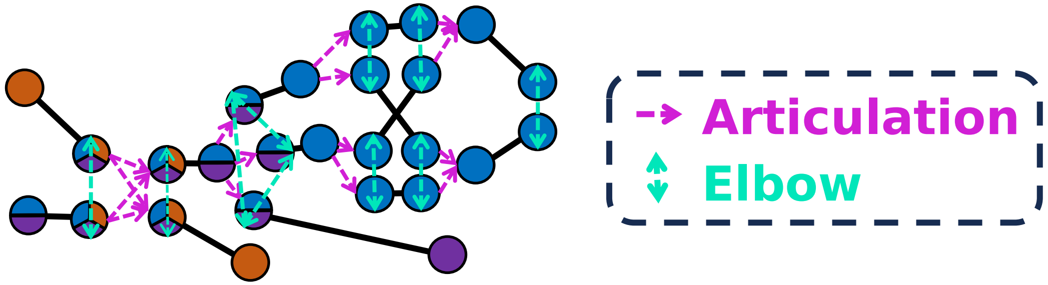


Handle Shared Hits



3

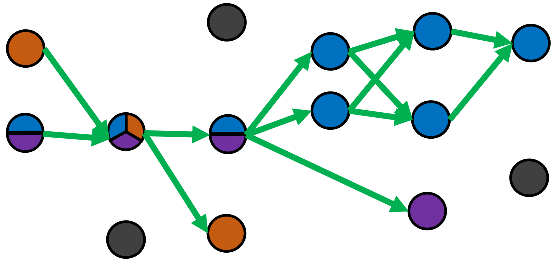
Build **edge-edge connections** (or **triplets**)
Use dataframes with Pandas (CPU) or cudf (GPU)



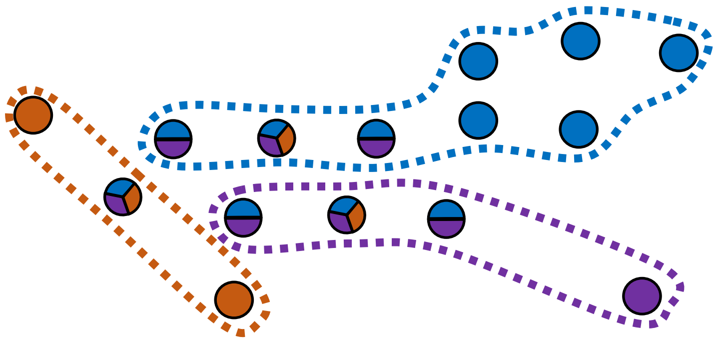
Shared hits

Updated Pipeline

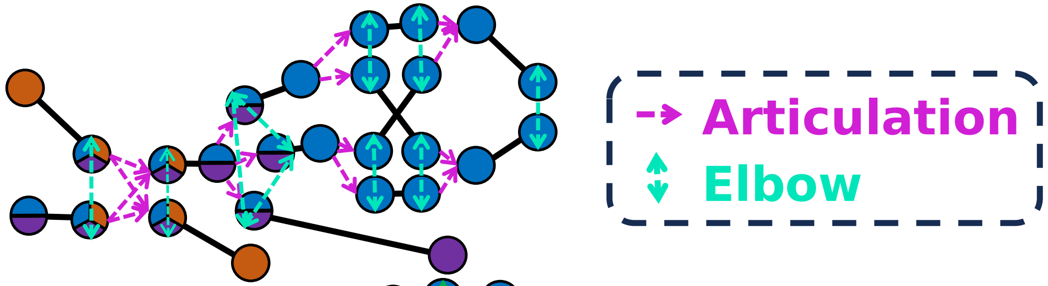
Goal



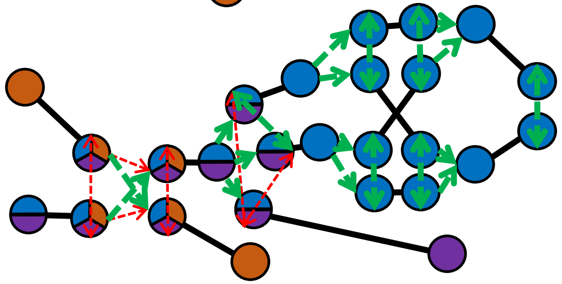
Handle Shared Hits



3 Build **edge-edge connections** (or **triplets**)
Use dataframes with Pandas (CPU) or cudf (GPU)



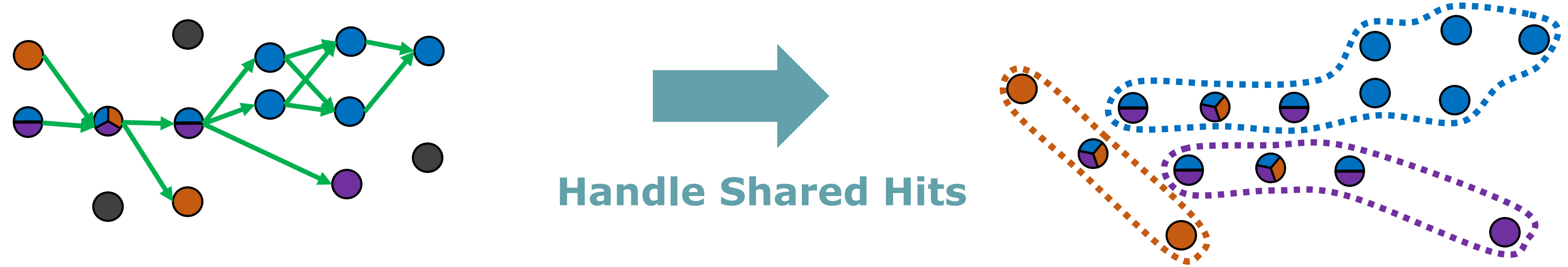
4 **Classify the triplets** with the GNN
Filter out the fake triplets



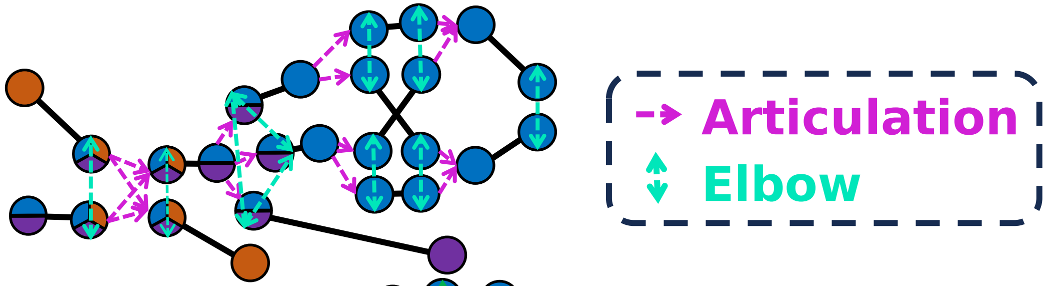
Shared hits

Updated Pipeline

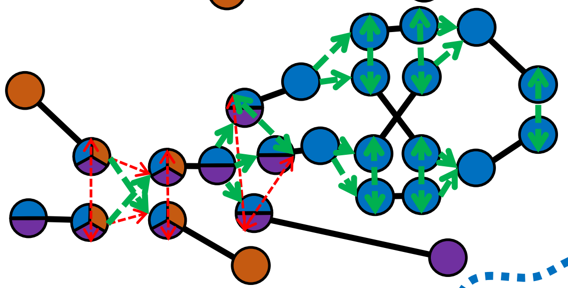
Goal



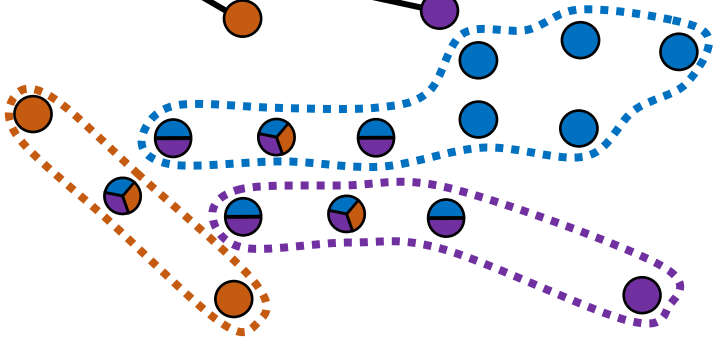
3 Build **edge-edge connections** (or **triplets**)
Use dataframes with Pandas (CPU) or cudf (GPU)



4 **Classify the triplets** with the GNN
Filter out the fake triplets



5 **Build tracks from triplets**



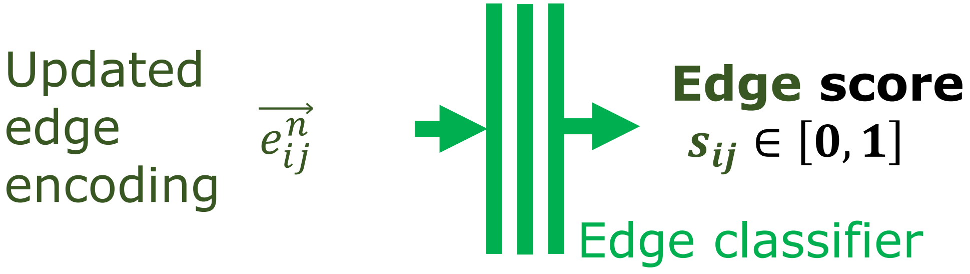
- Pipeline has become

Embedding Network → kNN ($k_{\text{max}}, d^2_{\text{max}}$) → GNN edge classifier → Filter edges ($s_{\text{edge,min}}$) → Build triplets → Triplet classifier → Filter triplets ($s_{\text{triplet,min}}$) → Build tracks

Build rough graph | Filter out fake edges | Filter out fake triplets

Embedding Network → kNN $k_{\text{max}}, d^2_{\text{max}}$ → GNN edge classifier → Filter edges $s_{\text{edge,min}}$ → Build triplets → **Triplet classifier** → Filter triplets $s_{\text{triplet,min}}$ → Build tracks

- Last step of the GNN edge classifier was



ETX4VELO

Embedding Network → kNN $k_{\max}, d^2_{\max}$ → GNN edge classifier → Filter edges $s_{\text{edge,min}}$ → Build triplets → **Triplet classifier** → Filter triplets $s_{\text{triplet,min}}$ → Build tracks

- Last step of the GNN edge classifier was

Updated edge encoding $\overrightarrow{e^n_{ij}}$ → Edge classifier → **Edge score** $s_{ij} \in [0, 1]$

- **DO NOT REPEAT THE GNN**

Concatenated edge encodings $\begin{pmatrix} \overrightarrow{e^n_{AC}} \\ \overrightarrow{e^n_{CB}} \end{pmatrix}$ → Triplet classifier → **Triplet score** $s_{A \to C \to B} \in [0, 1]$

- **Instead,** re-use directly the **very same** edge encodings!

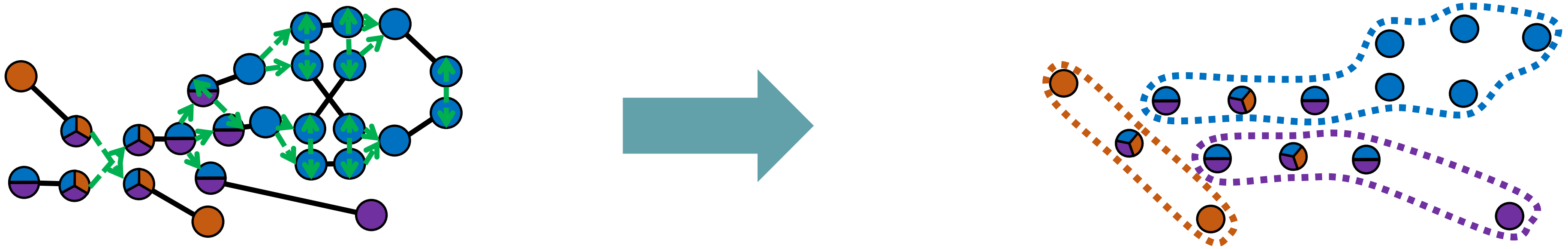
- Training with **sum of edge and triplet classification losses**:

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{edges}} + \mathcal{L}_{\text{triplets}}$$

Shared hits

Embedding Network → kNN $k_{\max}, d^2_{\max}$ → GNN edge classifier → Filter edges $s_{\mathrm{edge,min}}$ → Build triplets → Triplet classifier → Filter triplets $s_{\mathrm{triplet,min}}$ → **Build tracks**

Goal



In **annexe**.

Performance

Physics performance

Embedding Network → kNN $k_{\max}, d^2_{\max}$ → GNN edge classifier → Filter edges $s_{\mathrm{edge,min}}$ → Build triplets → Triplet classifier → Filter triplets $s_{\mathrm{triplet,min}}$ → Build tracks

- Choice of $s_{\mathrm{edge,min}}$ and $s_{\mathrm{triplet,min}}$: **vary them** and **choose the one leading to best performance**.

⇒ can reach better physics performance than default algorithm at LHCb.

Proportion of…	**Default GPU algorithm**	**ETX4VELO**
Reconstructed particles	99.08%	99.33%
Duplicate tracks	2.65%	1.09%
Fake tracks	2.51%	0.71%

For particles reconstructible in the VELO and the SciFi.

However, need to consider **throughput ≡ # bunch crossings processed / s**

Performance

Throughput

- **Goal**: implement GNN-based pipeline on C++/CUDA inside Allen  .
  - **Optimization**: To **optimize throughput**
    (PyTorch slower than C++/CUDA implementation)
  - **Integration**: can be used with other reconstruction algorithms.

Performance

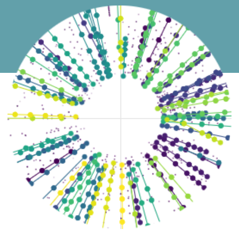
Throughput

- **Goal**: implement GNN-based pipeline on C++/CUDA inside Allen  .
  - **Optimization**: To **optimize throughput**
    (PyTorch slower than C++/CUDA implementation)
  - **Integration**: can be used with other reconstruction algorithms.
- **Pipeline**: *no triplet for the moment*

Graph Building → GNN to filter edges → Build tracks

- **Detailed view**:

Embedding Neural Network → kNN → GNN edge classifier → Filter edges → Weakly connected component algorithm



Performance

Throughput

- **Goal**: implement GNN-based pipeline on C++/CUDA inside Allen  .
  - **Optimization**: To **optimize throughput**
    (PyTorch slower than C++/CUDA implementation)
  - **Integration**: can be used with other reconstruction algorithms.
- **Pipeline**: *no triplet for the moment*

Graph Building → GNN to filter edges → Build tracks

- **Detailed view**:

Embedding Neural Network → kNN → GNN edge classifier → Filter edges → Weakly connected component algorithm

- There are **deep learning model to export** and **algorithms to implement in C++/CUDA**.
- Deep learning model inference in C++:
  1. Export model in **ONNX** open-source format.
  2. Inference in C++ using either ONNXRuntime or NVIDIA TensorRT libraries.

Performance

Throughput

- So far

	Throughput: # bunch crossings processed / s
ETX4VELO with **ONNXRuntime**	310
ETX4VELO with **TensorRT**	730
Allen (default)	540k

- But we still have ideas  to increase the throughput.

Conclusion

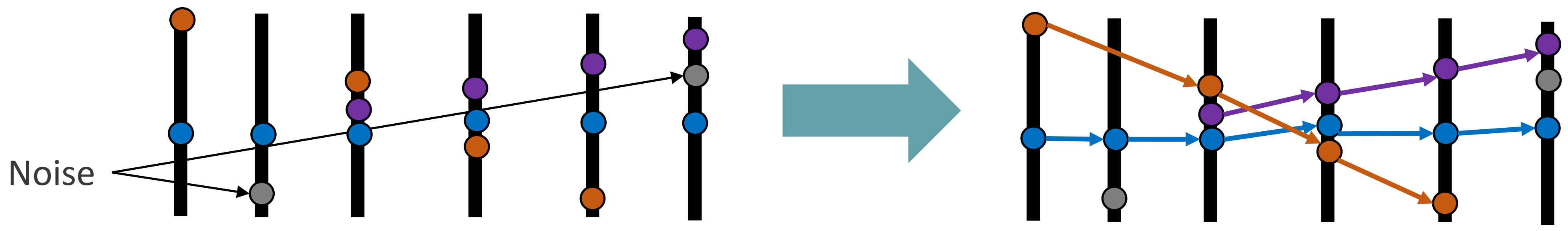
- First-level trigger at LHCb on **GPU**
- **ETX4VELO**:
  - New GNN-based pipeline for track-finding in the Velo at LHCb.
  - Repository and documentation.
- Can meet Allen  **physics** performance.
- Ongoing work:
  - Run the **full ETX4VELO pipeline in C++/CUDA inside Allen**.
  - Optimise **ETX4VELO throughput**.
  - Adapt the pipeline to other LHCb tracking detectors (e.g., SciFi detector).

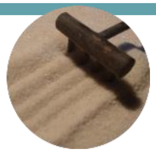
Thank you!

arXiv:2207.03936

Search by Triplet on GPU

Goal



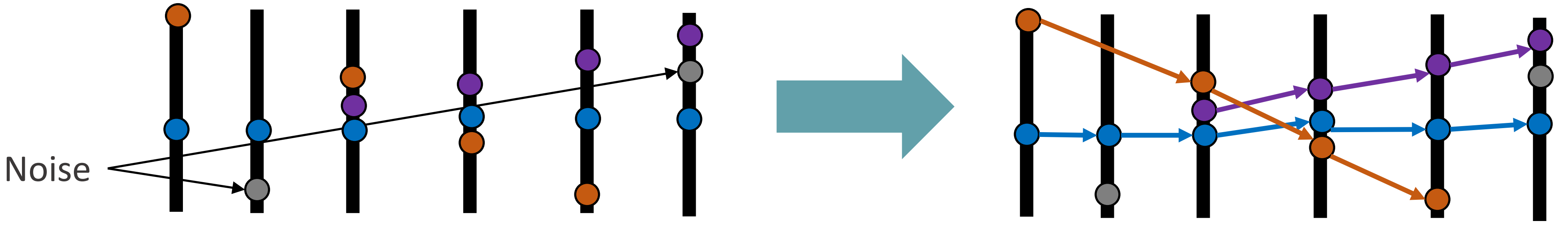
In **Allen**  : **search by triplet** algorithm

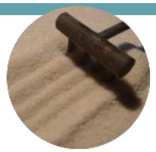
Track Finding in the Velo

Search by Triplet on GPU

arXiv:2207.03936

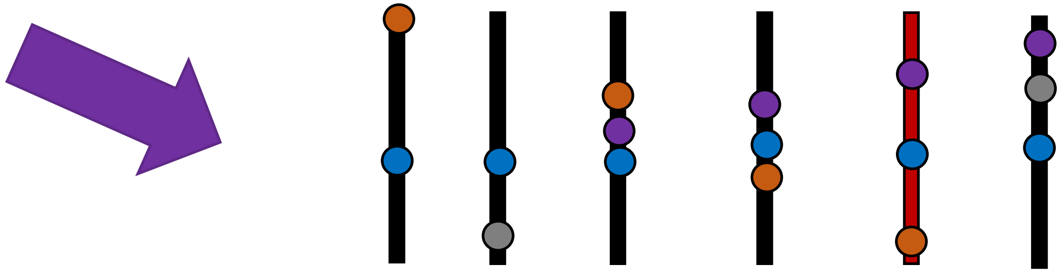
Goal



In **Allen**  : **search by triplet** algorithm

1 Iterate from the **last** to the **first** plane:

a **Seeding**:
Find compatible triplets of hits

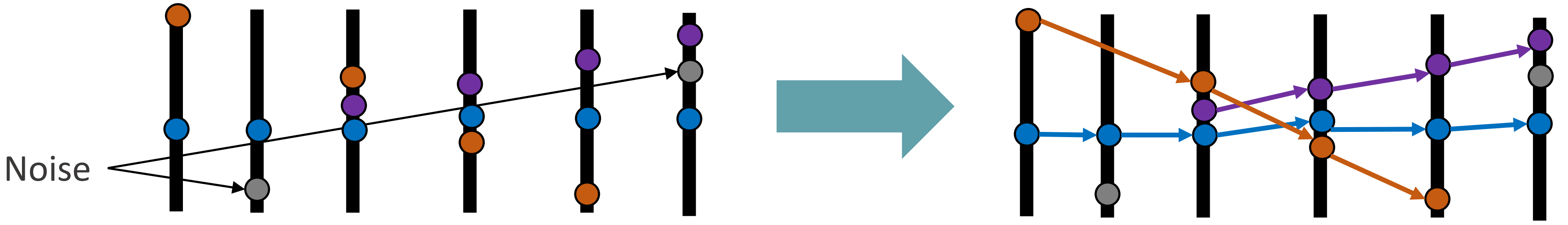


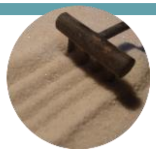
b **Following**:
a.1. Extrapolate the track
a.2. Find compatible hits

Search by Triplet on GPU

arXiv:2207.03936

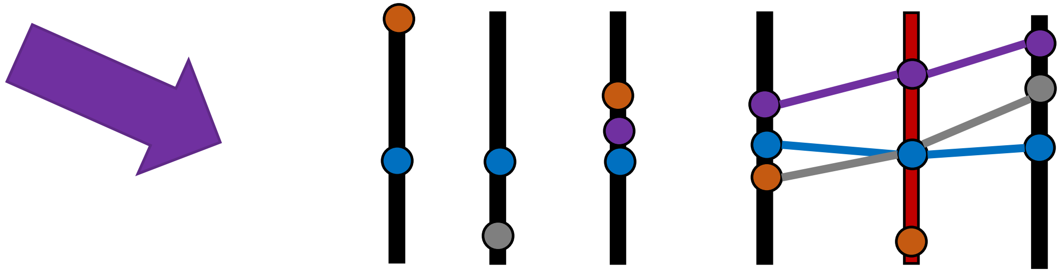
Goal



In **Allen**  : **search by triplet** algorithm

1 Iterate from the **last** to the **first** plane:

- a **Seeding**:
  Find compatible triplets of hits



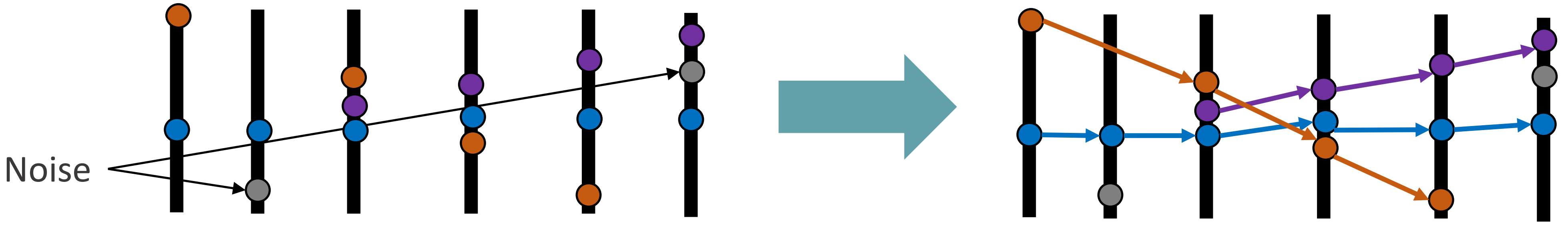
- b **Following**:
  a.1. Extrapolate the track
  a.2. Find compatible hits

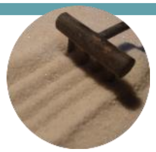
Track Finding in the Velo

Search by Triplet on GPU

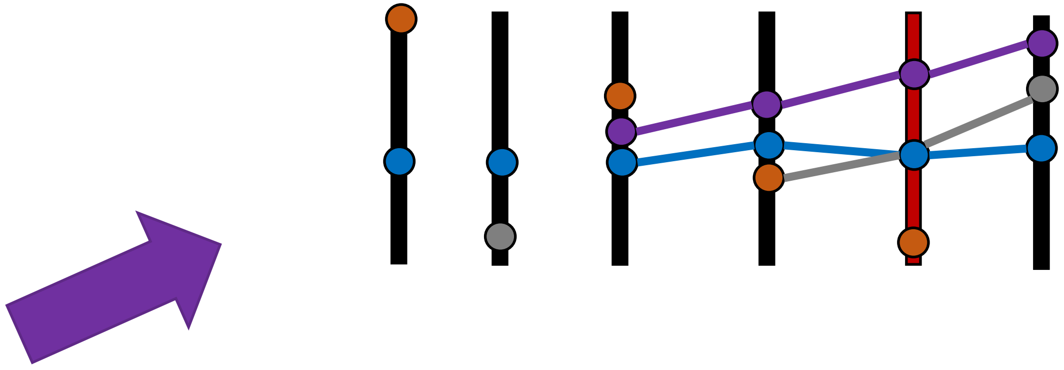
arXiv:2207.03936

Goal



In **Allen**  : **search by triplet** algorithm

1. Iterate from the **last** to the **first** plane:
   - a. **Seeding**: Find compatible triplets of hits
   - b. **Following**:
     a.1. Extrapolate the track
     a.2. Find compatible hits

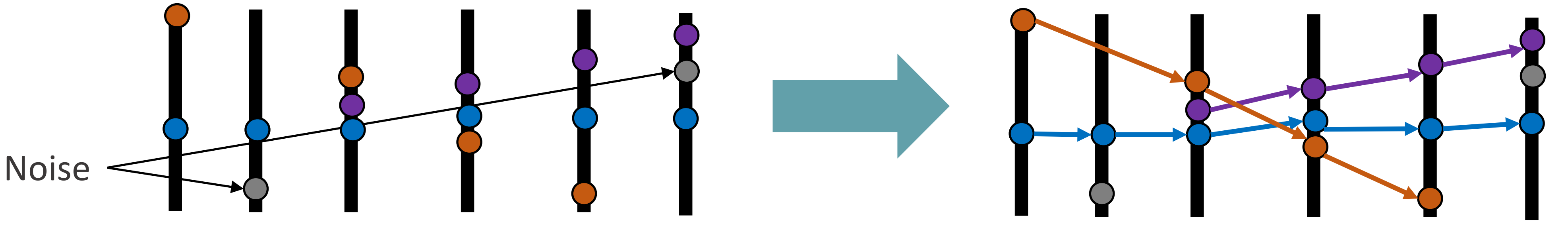


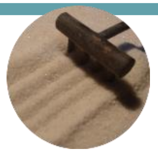
Track Finding in the Velo

arXiv:2207.03936

Search by Triplet on GPU

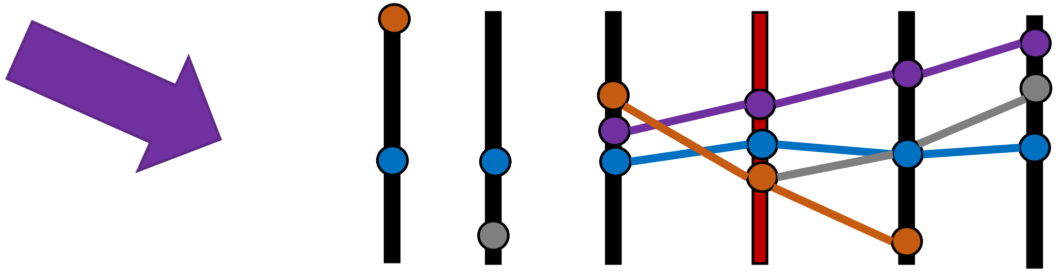
Goal



In **Allen**  : **search by triplet** algorithm

1 Iterate from the **last** to the **first** plane:

a **Seeding**:
Find compatible triplets of hits

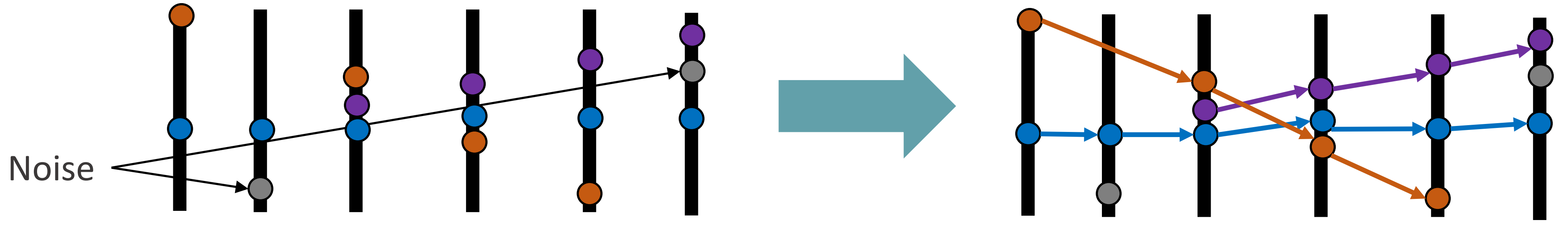


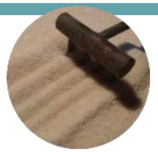
b **Following**:
a.1. Extrapolate the track
a.2. Find compatible hits

Search by Triplet on GPU

arXiv:2207.03936

Goal

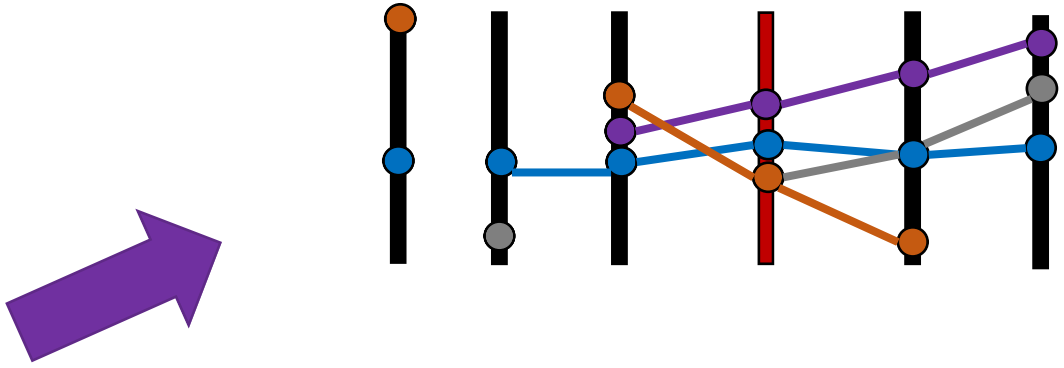


In **Allen**  : **search by triplet** algorithm

1 Iterate from the **last** to the **first** plane:

a **Seeding**:
Find compatible triplets of hits

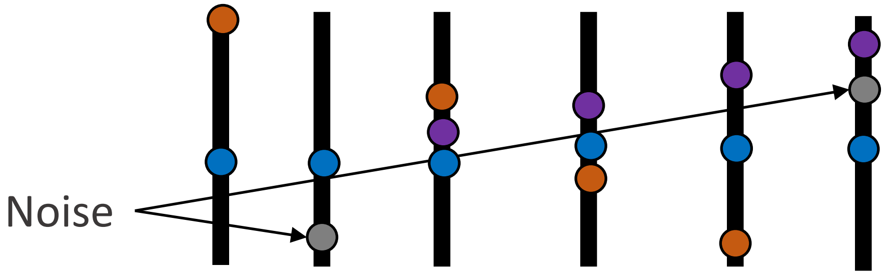
b **Following**:
a.1. Extrapolate the track
a.2. Find compatible hits

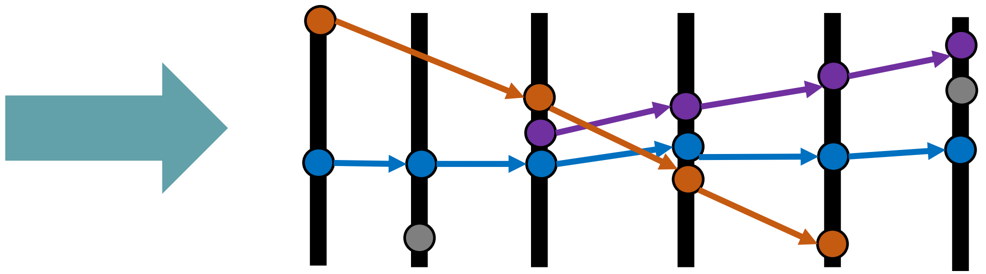


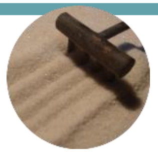
arXiv:2207.03936

Search by Triplet on GPU

Goal

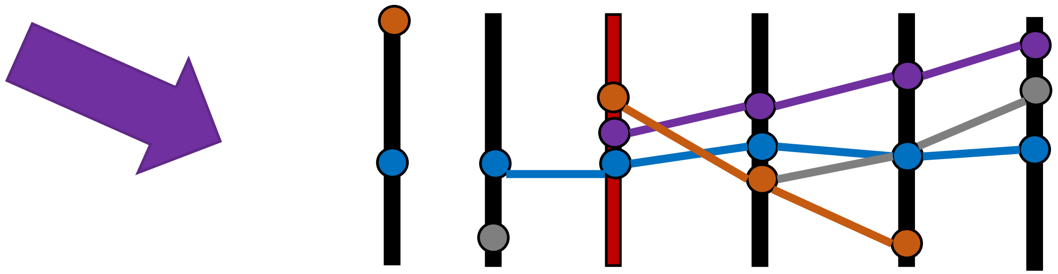




In **Allen**  : **search by triplet** algorithm

1 Iterate from the **last** to the **first** plane:

a **Seeding**:
Find compatible triplets of hits

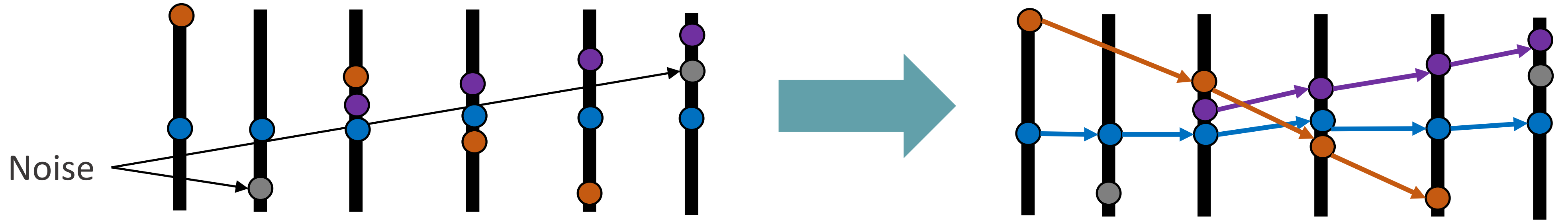


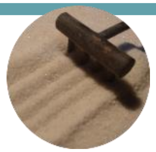
b **Following**:
a.1. Extrapolate the track
a.2. Find compatible hits

arXiv:2207.03936

Search by Triplet on GPU

Goal

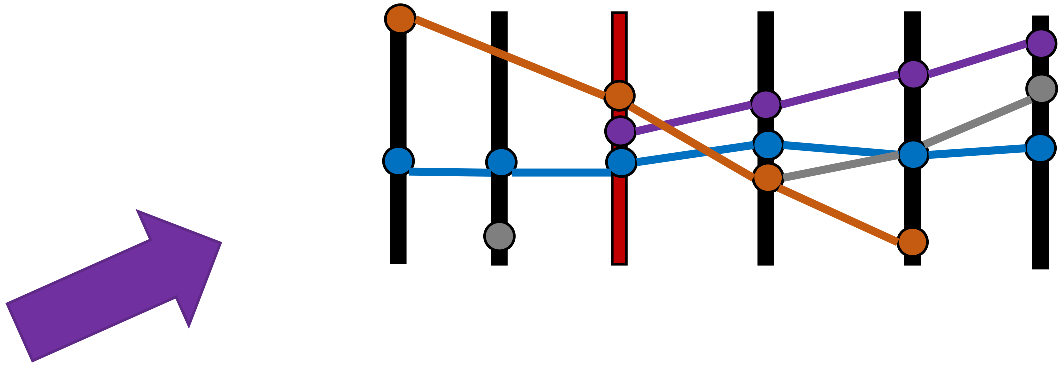


In **Allen**  : **search by triplet** algorithm

1 Iterate from the **last** to the **first** plane:

a **Seeding**:
Find compatible triplets of hits

b **Following**:
a.1. Extrapolate the track
a.2. Find compatible hits

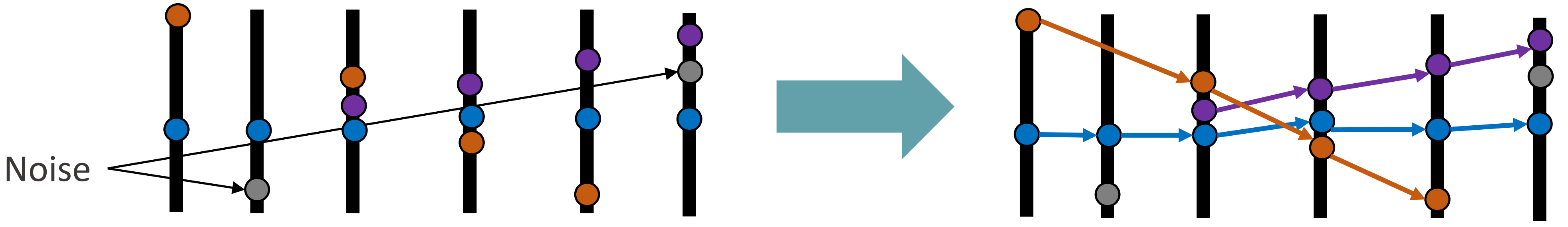


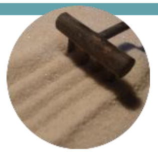
Track Finding in the Velo

arXiv:2207.03936

Search by Triplet on GPU

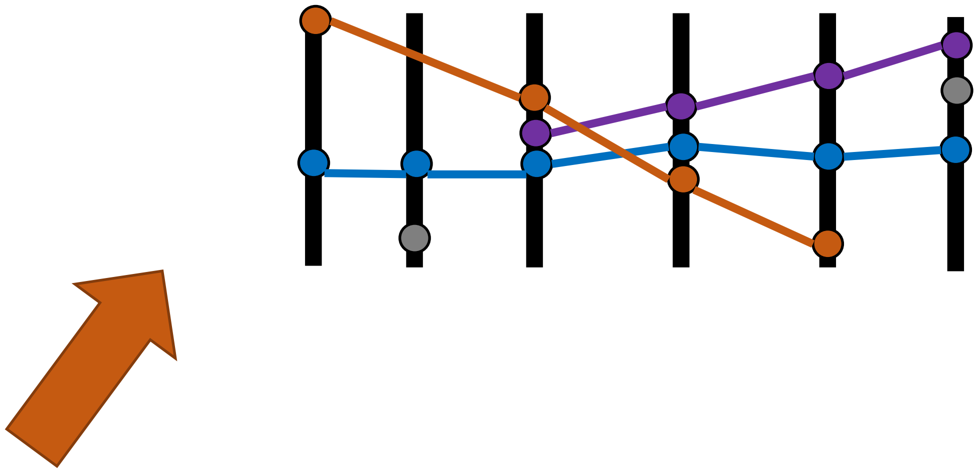
Goal



In **Allen**  : **search by triplet** algorithm

1. Iterate from the **last** to the **first** plane:
   - a. **Seeding**: Find compatible triplets of hits
   - b. **Following**: a.1. Extrapolate the track a.2. Find compatible hits

2. Filter triplets



Stage 1: Graph Building

Graph Building | GNN to filter edges | Build tracks from graph

Training step

1. Embed all the hits using the network $(r, \phi, z, \text{plane}) \longrightarrow$ DNN $\longrightarrow \vec{e} = (e_1, e_2, e_3, e_4)$

2. For a random given set of hits, build a **dataset of genuine edges and fake edges**. Compute the distances between their hits in the embedding space:
$$\{d^2_{\text{genuine},i}, \forall i\} \text{ and } \{d^2_{\text{fake},j}, \forall j\}$$

3. Minimise hinge loss $\mathcal{L}_{\text{total}} = \underbrace{8}_{\text{hyperparameter}}\mathcal{L}_{\text{genuine}} + \mathcal{L}_{\text{fake}}$ where

$$\mathcal{L}_{\text{genuine}} = \frac{1}{n_{\text{genuine}}} \sum_i d^2_{\text{genuine},i}$$

Minimise $d_{\text{genuine},i}$

$$\mathcal{L}_{\text{fake}} = \frac{1}{n_{\text{fake}}} \sum_j \max(\underbrace{1}_{\text{hyperparameter}} - d^2_{\text{fake},j}, 0)$$

Maximise $d_{\text{fake},j}$

Training dataset

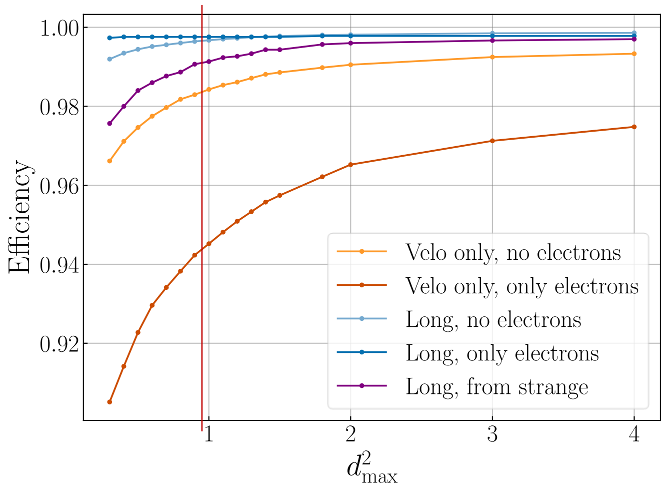
- **Hard Negative Mining**: edges built by a kNN *(→ "hard" negatives)*
- **True** edges
- **Random** edges

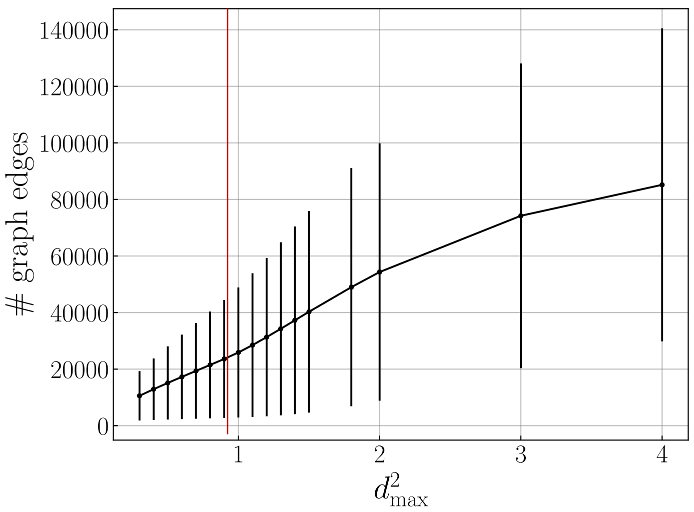
Stage 1: Graph Building

Graph Building | GNN to filter edges | Build tracks from graph

- *After training*, we choose maximal number of neighbours $\boldsymbol{k_{\max} = 50}$ (not optimised)
- To choose maximal squared distance $d^2_{\max}$, for various values for $d^2_{\max}$:
  1. Build the rough graph using $d^2_{\max}$
  2. **Remove all fake edges** in the rough graph and build the tracks from this purified graph
  3. Compute track-finding performance ⇒ correspond to the **best performance given $\boldsymbol{d^2_{\max}}$**

Performance if all the fake edges are discarded(≡ best performance)

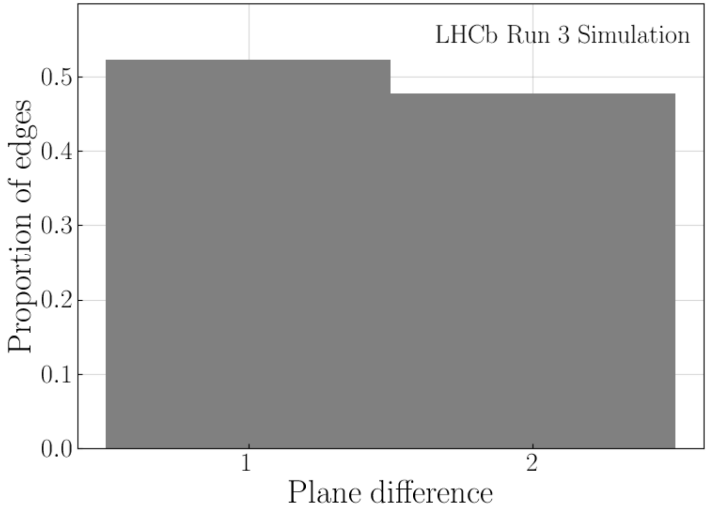




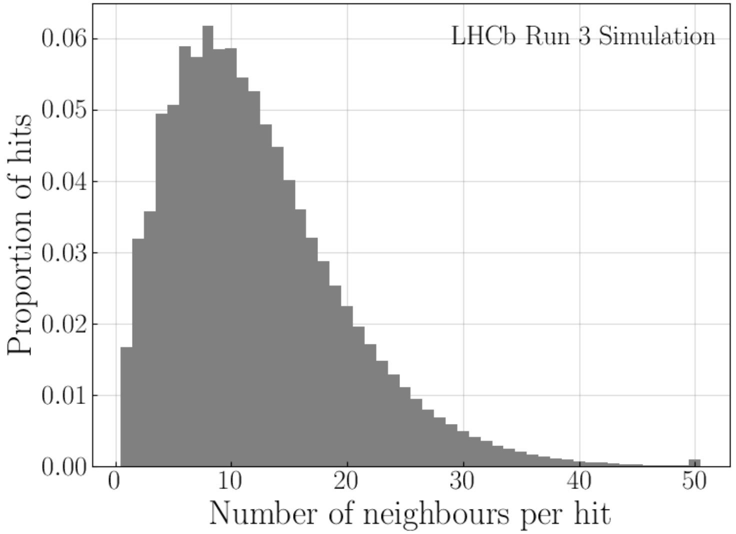
⇒ Choose $\boldsymbol{d^2_{\max} = 0.9}$

(evaluated on 200 events)

Rough graph with $k_{\max} = 50$ and $d^2_{\max} = 0.010$



Even though **1% of genuine edges are 2-plane apart**, the rough graph needs to contain **almost 50% of such edges**



$\Rightarrow$ $\boldsymbol{k_{\max}}$ **could probably be reduced** to increase throughput

Graph Neural Networks

- **Graph Neural Network** (**GNN**): Neural Network architecture that operates on graphs 

  → Problem needs **to be formulated with** a graph.

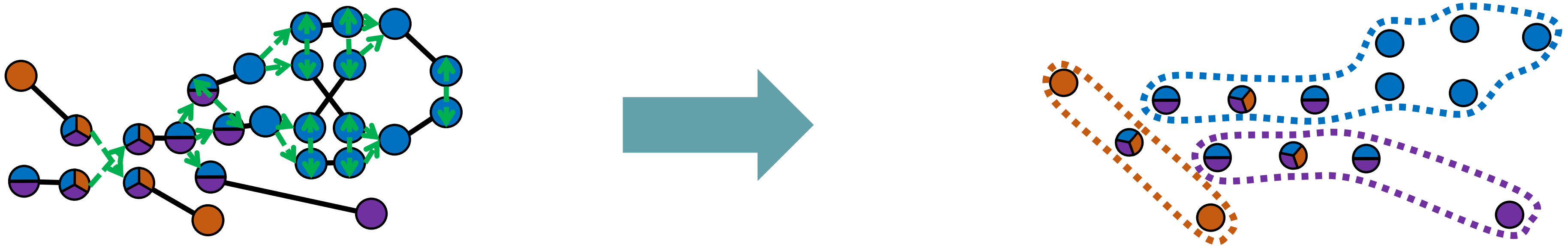
Resource	Description	Opinion
Stanford online videos	Series of lectures recorded in Youtube.	• Clear explanations • Quite complete • Notebooks without solution
Introduction to Graph Neural Networks Zhiyuan Liu, Lie Zhou	Book	• Clear, quite complete • Succinct
PyTorch Geometric Tutorials	Videos & notebooks	• Notebooks • Use of PyTorch Geometric • Not very clear • Self-advertisement
Graph Neural Networks: Foundations, Frontiers, and Applications Lingfei Wu, Peng Cui, Jian Pei, Liang Zhao	Book	• Extremely complete • Not so good for just learning

Probably other learning resources out there!

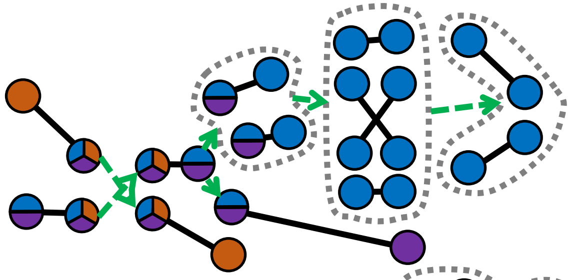
Shared hits

Embedding Network → kNN $k_{\mathrm{max}}, d^2_{\mathrm{max}}$ → GNN edge classifier → Filter edges $s_{\mathrm{edge,min}}$ → Build triplets → Triplet classifier → Filter triplets $s_{\mathrm{triplet,min}}$ → **Build tracks**

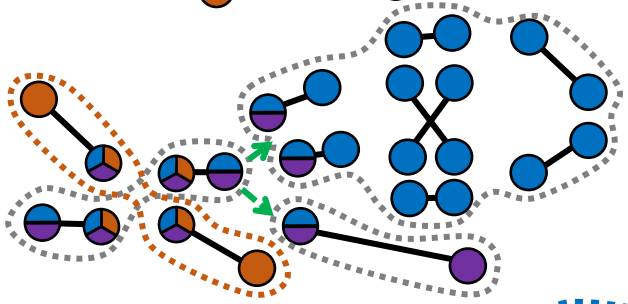
Goal



1 **Connect left and right elbows** and remove duplicate edge-edge connections



2 **Apply connected components**, excluding splitting edge-edge connections



3 **Each remaining link** correspond to a **new track**