GNN-based pipeline for track finding in the Velo at LHCb

Anthony Correia, Fotis Giasemis, Nabil Garroum, Vava Gligorov

On behalf of the LHCb real-time analysis project

10th October 2023

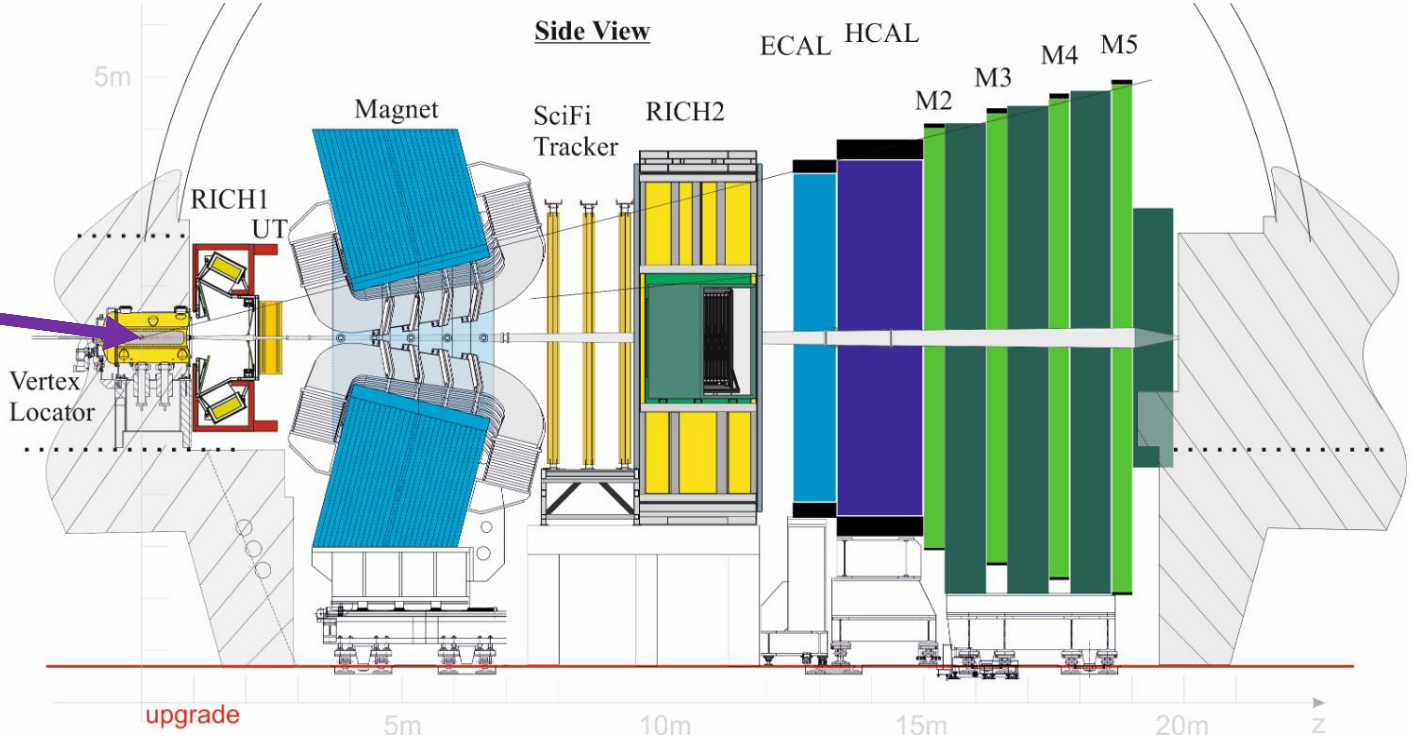
In collaboration with

LHCb Detector in Run 3

Collisions and Trigger

Collisions (Run 3)

- $20\ \mathrm{MHz}$ non-empty bunch crossing rate
- $\sim$ 5 collisions / bunch crossing
- p-p collision at $\sqrt{s} = 13.6\ \mathrm{TeV}$



J. Phys.: Conf. Ser., vol. 878, p. 012012, 2017

LHCb Detector in Run 3

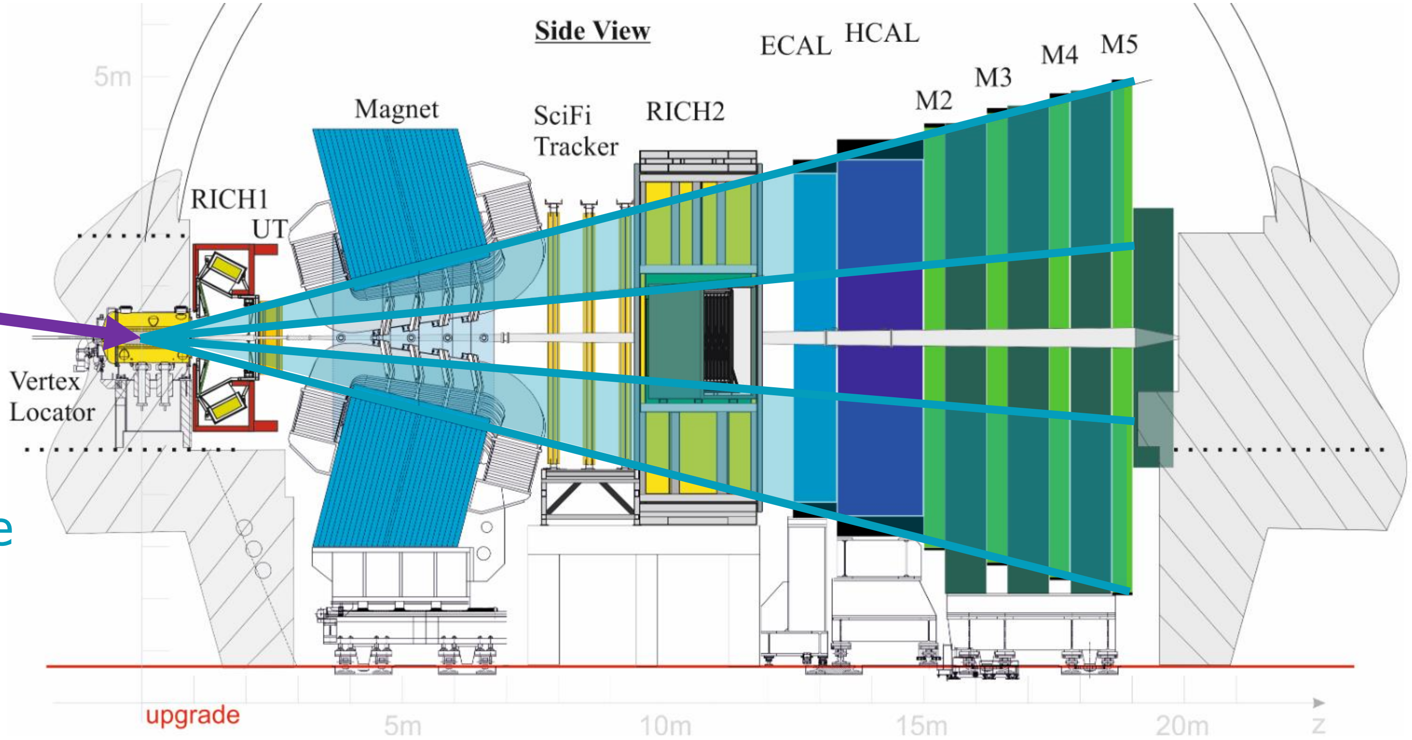
Collisions and Trigger

Collisions (Run 3)

- $20\ \mathrm{MHz}$ non-empty bunch crossing rate
- $\sim$ 5 collisions / bunch crossing
- p-p collision at $\sqrt{s} = 13.6\ \mathrm{TeV}$

LHCb Subdetectors

Acceptance
$2 < \eta < 5$



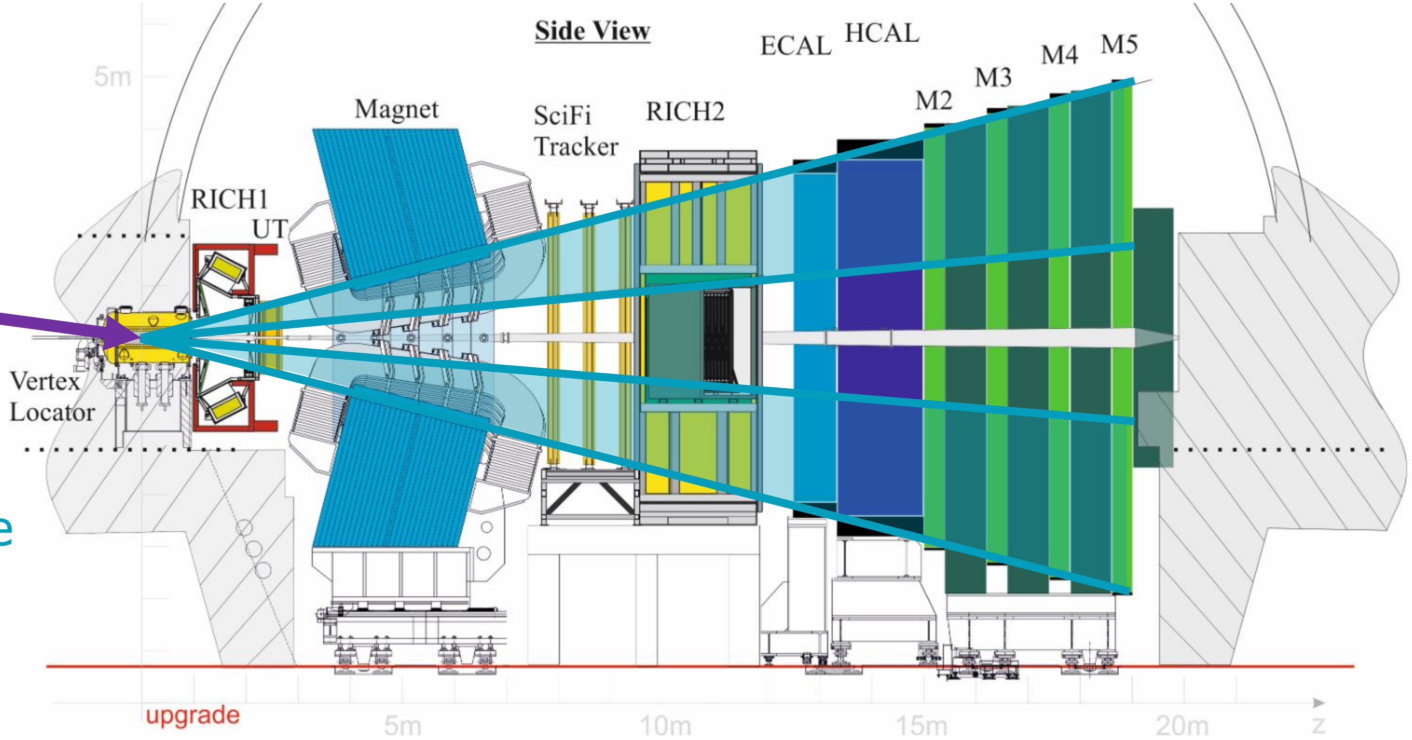
[J. Phys.: Conf. Ser., vol. 878, p. 012012, 2017](#)

LHCb Detector in Run 3

Collisions and Trigger

Collisions (Run 3)

- $20\,\mathrm{MHz}$ non-empty bunch crossing rate
- $\sim$ 5 collisions / bunch crossing
- p-p collision at $\sqrt{s} = 13.6\,\mathrm{TeV}$



J. Phys.: Conf. Ser., vol. 878, p. 012012, 2017

LHCb Subdetectors

Acceptance $2 < \eta < 5$

5 TB/s

Allen (High-Level Trigger 1)
fully GPU-based online partial reconstruction and selection

Better trigger efficiency than previous L0 FPGA-based trigger

70-200 GB/s

Storage buffer

High-Level Trigger 2
CPU-based full reconstruction and selection

10 GB/s

Numbers taken from LHCB-FIGURE-2020-016

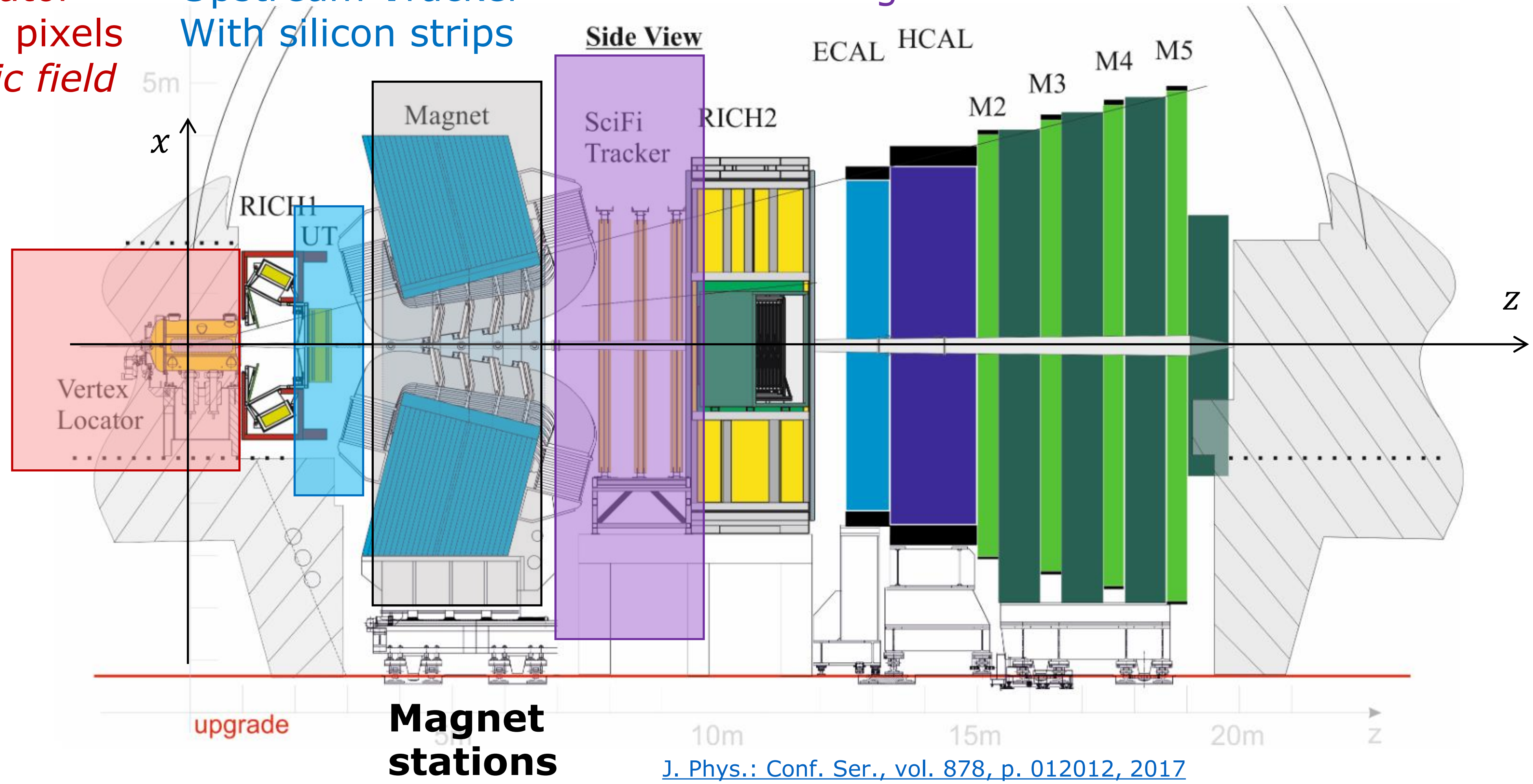
LHCb Detector in Run 3

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



Magnet stations

J. Phys.: Conf. Ser., vol. 878, p. 012012, 2017

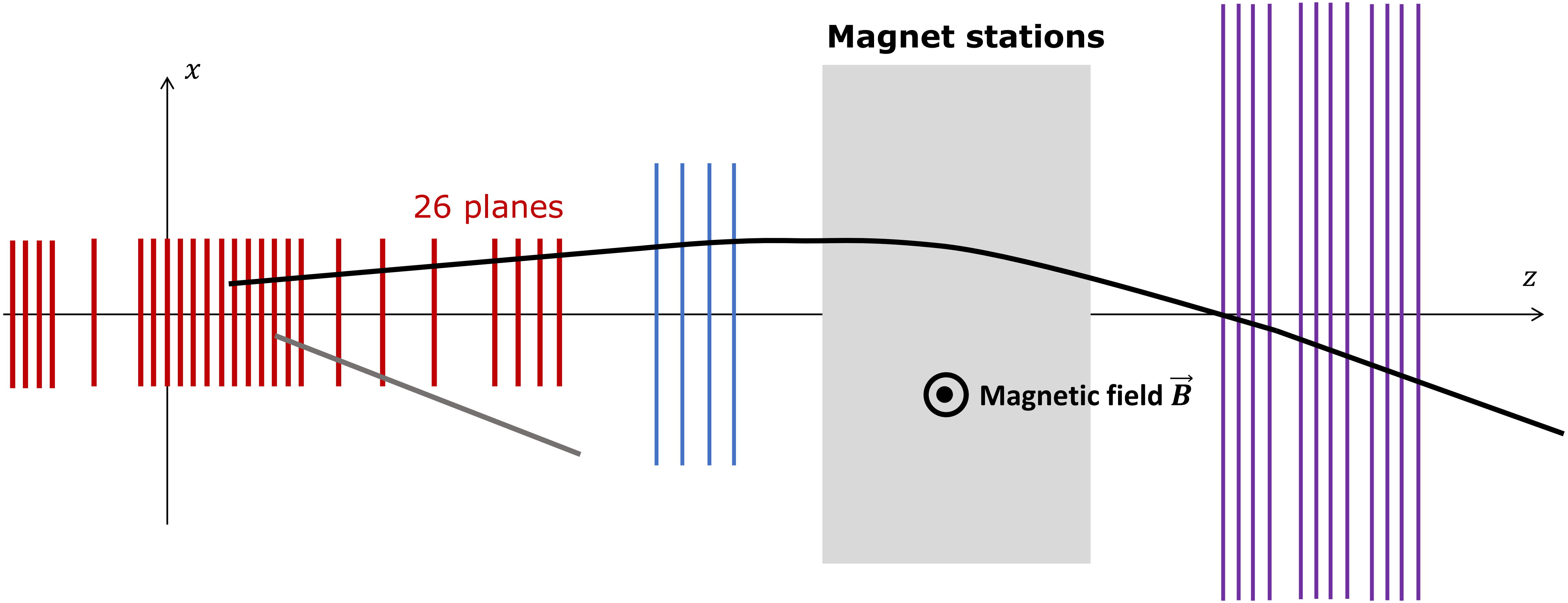
LHCb Detector in Run 3

Tracks

Velo
Vertex **Lo**cator
With silicon pixels

UT
Upstream **T**racker
With silicon strips

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



Magnet stations

26 planes

Magnetic field $\vec{B}$

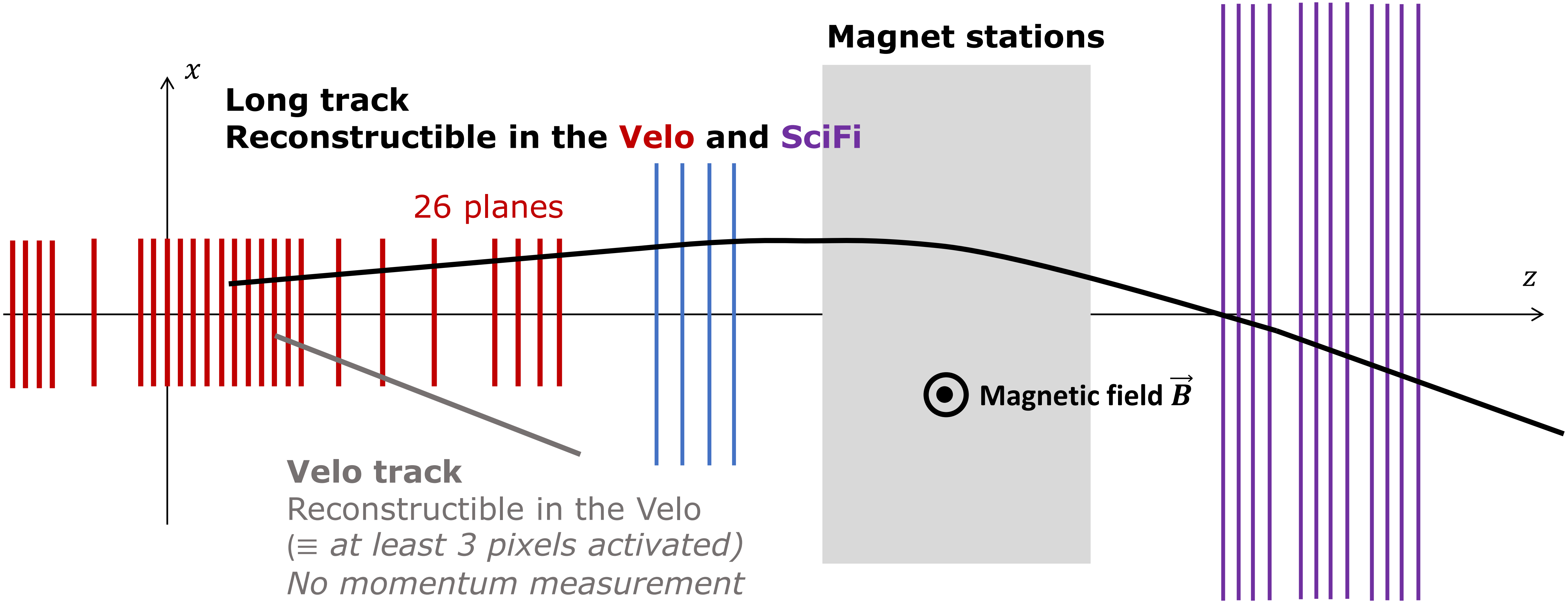
LHCb Detector in Run 3

Tracks

Velo
Vertex **Lo**cator
With silicon pixels

UT
Upstream **T**racker
With silicon strips

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



Long track
Reconstructible in the Velo and SciFi

Velo track
Reconstructible in the Velo
(≡ at least 3 pixels activated)
No momentum measurement

1. Graph Neural Network Track Finding

Motivations

Graph Neural Network (GNN)-based track-finding pipeline based on the work of **Exa.Trkx** (*Eur. Phys. J. C* **81**, 876 (2021))

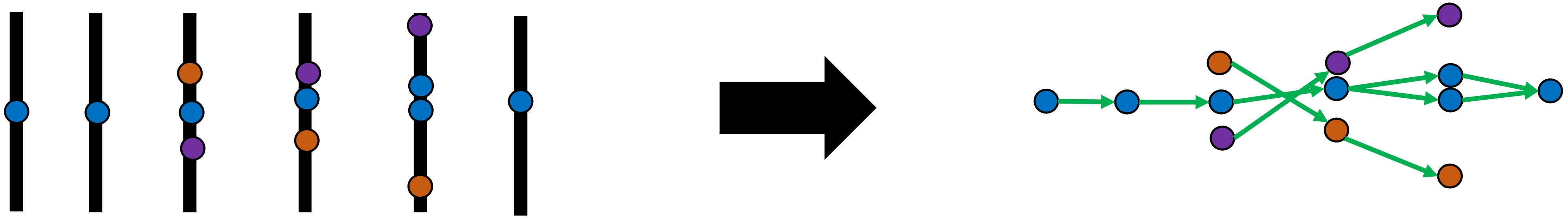
- Demonstrated **near-linear** inference time w.r.t. # hits
  - *Conventional* algorithms are worse-than-quadratic
  - In future LHCb upgrades: increase in **instaneous luminosity** and **detector granularity** → need for **even more high-throughput** track-finding algorithms

- **High-parallelisation** potential → compatible with current **GPU-based Allen** trigger

- Conventional algorithms implemented in Allen ⇒ allow **like-for-like comparison** between GNN-based algorithms and conventional algorithms **(on the very same device!)**

- Representation of tracks with a graph quite *natural*

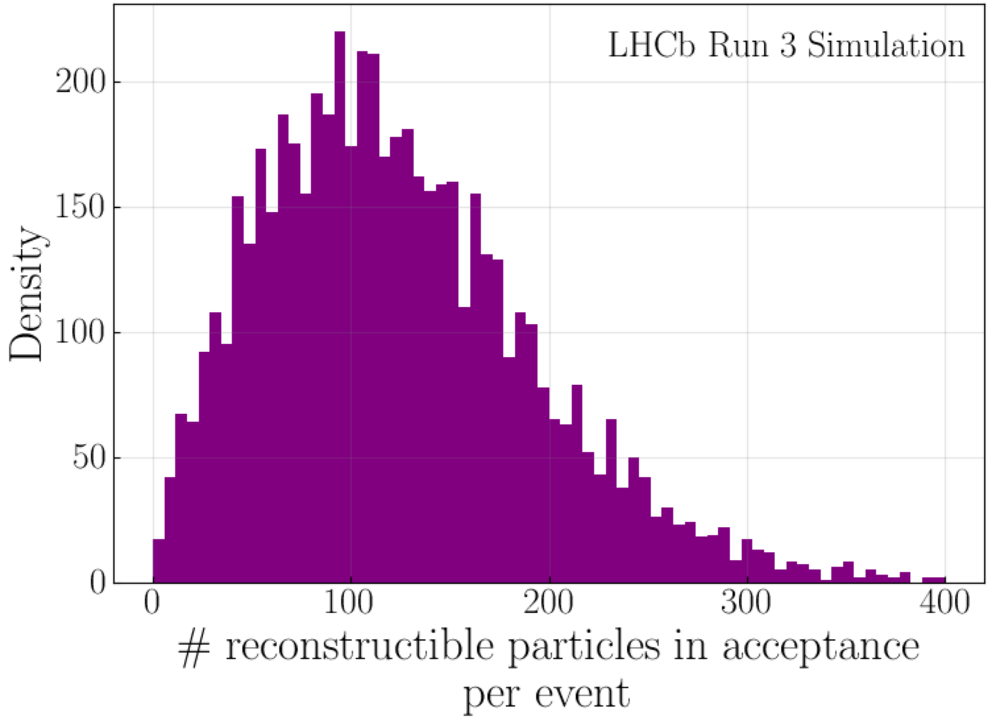
Pure graph representation



1. Graph Neural Network Track Finding

In the Velo

- Around **~ 2200 hits / event**
- Around **150 particles to reconstruct / event**

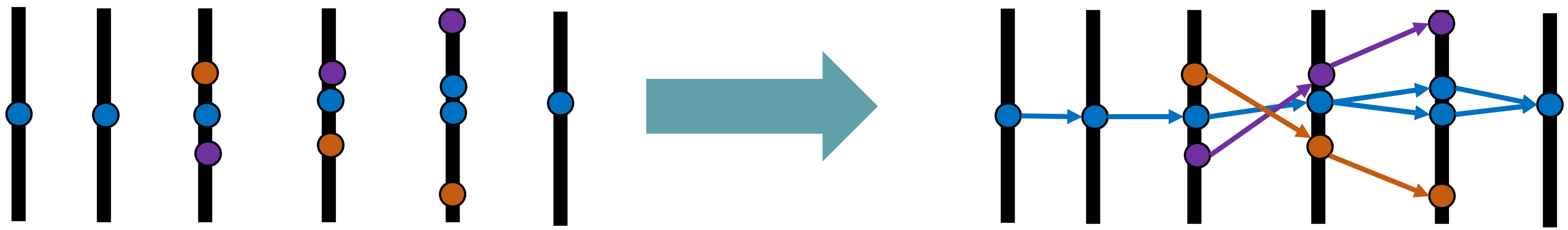


1. Graph Neural Network Track Finding

Goal

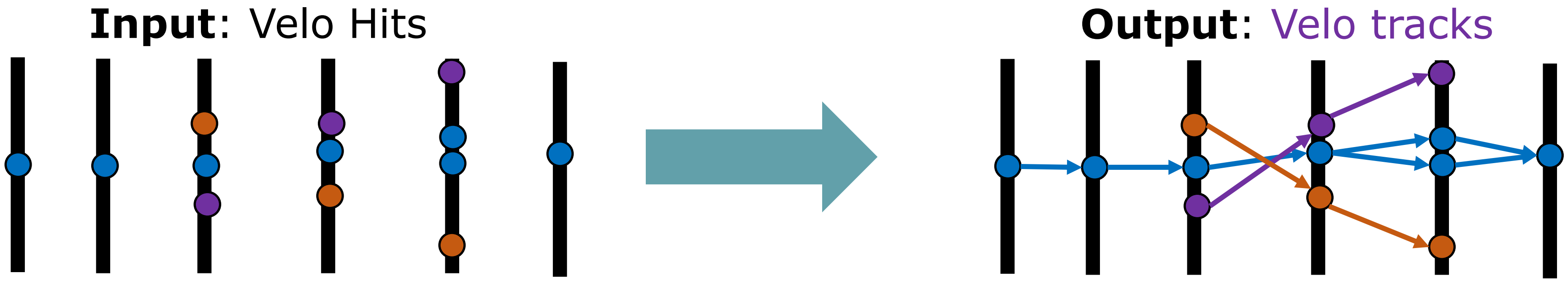
Input: Velo Hits

Output: Velo tracks



1. Graph Neural Network Track Finding

Goal

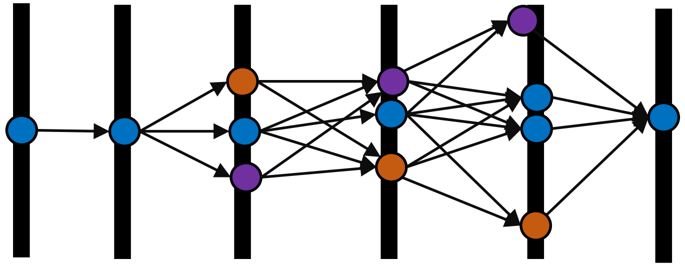

Input: Velo Hits

Output: Velo tracks

Strategy

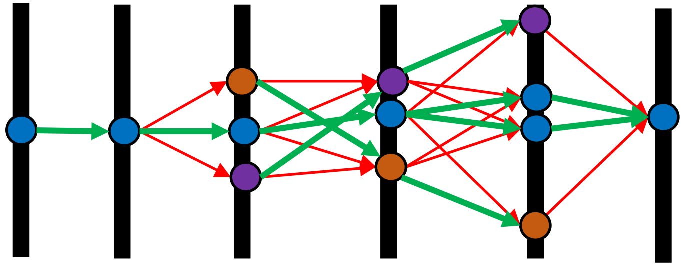
1 Build a **“rough” graph**

→ Embedding Network
+ Nearest-Neighbour Network



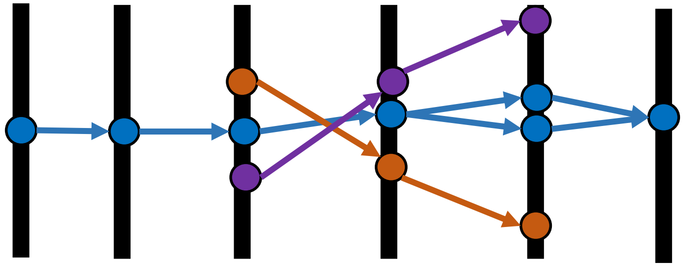
2 **Classify the edges** as genuine or fake

→ Graph Neural Network

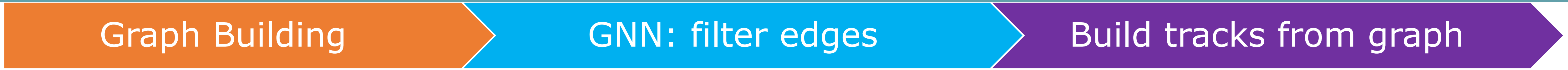


3 Keep only the genuine edges
Identify **connected hits** as tracks

→ Weakly connected component algorithm



1. Graph Neural Network Track Finding



Graph Building

GNN: filter edges

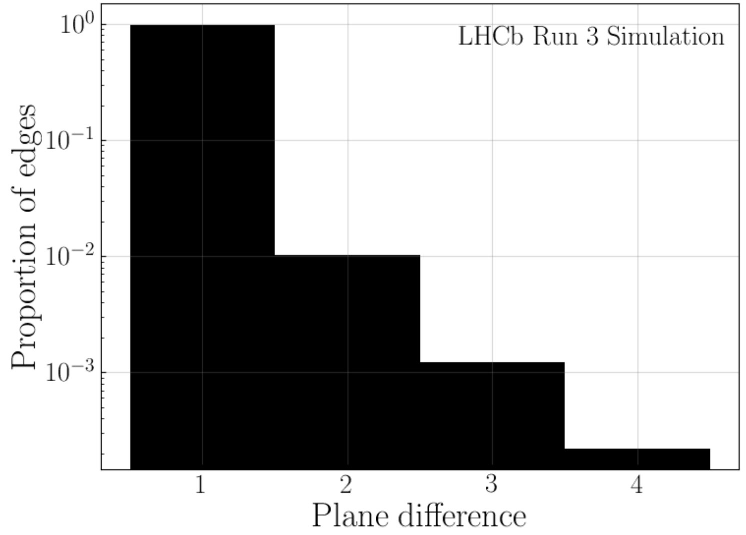
Build tracks from graph

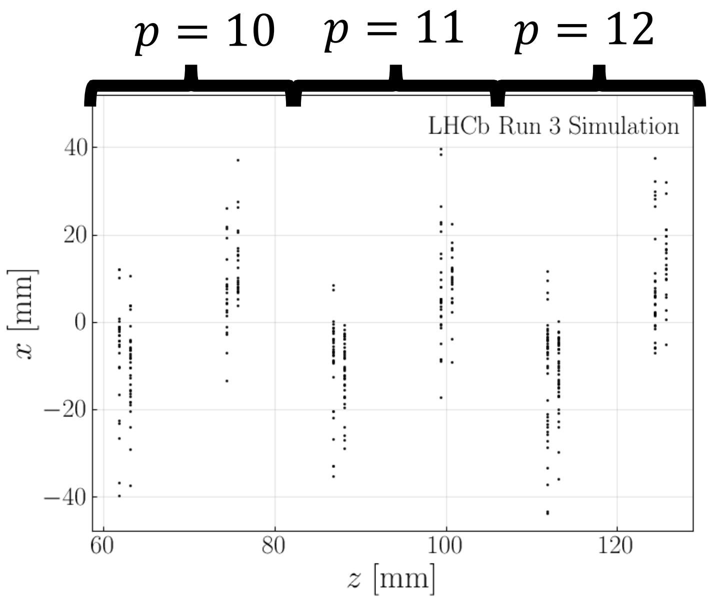
1. Graph Neural Network Track Finding

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

- **Goal**: **maximising edge efficiency** while **minimise # edges**

- **Edges**
  - 99% of genuine edges are 1-plane apart, 1% are 2-plane apart $\Rightarrow$ **allow for only 1 skipped plane** $(\sim 1\%)$
  - Only build edges **from left to right**





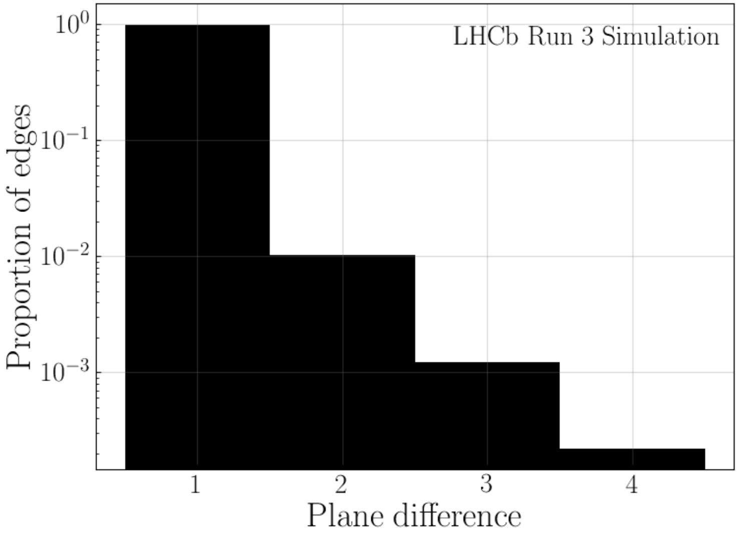
- *For every hit in plane p, how to connect it to hits belonging to the next 2 planes $p+1$ and $p+2$?*

1. Graph Neural Network Track Finding

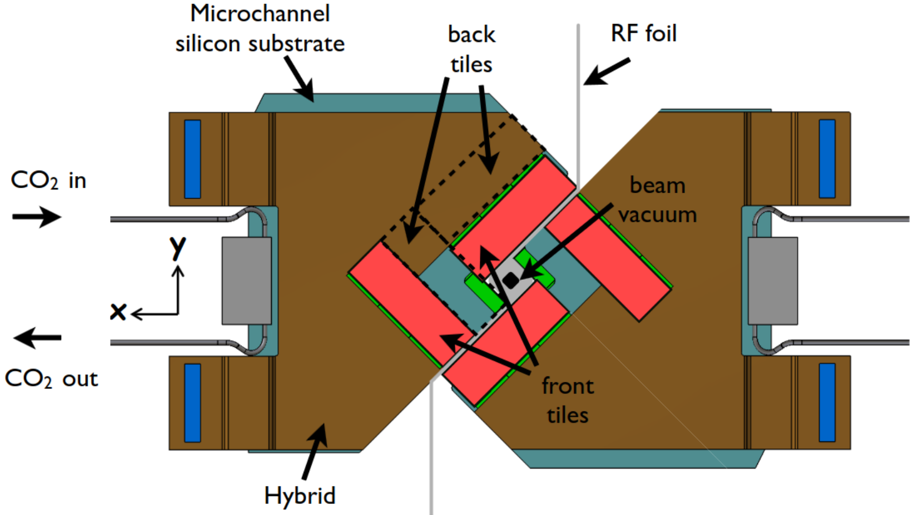
Graph Building → GNN: filter edges → Build tracks from graph

- **Goal**: **maximising edge efficiency** while **minimise # edges**

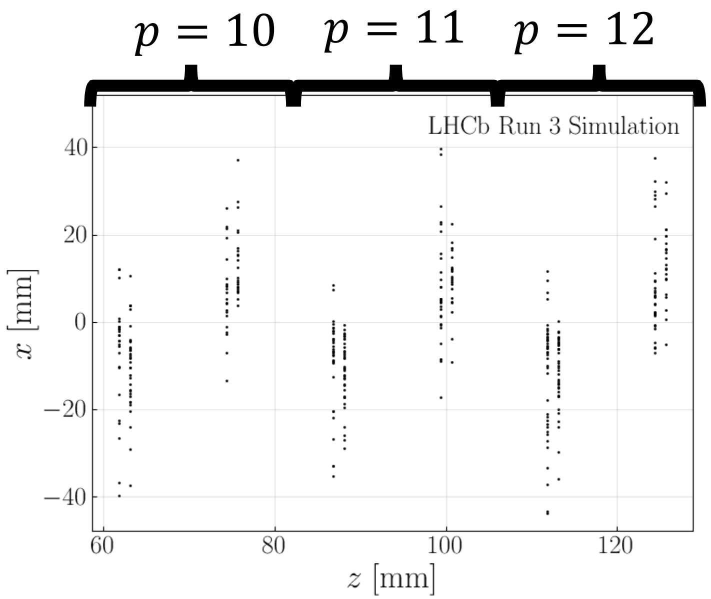
- **Edges**
  - 99% of genuine edges are 1-plane apart, 1% are 2-plane apart
    ⇒ **allow for only 1 skipped plane** (~1%)
  - Only build edges **from left to right**



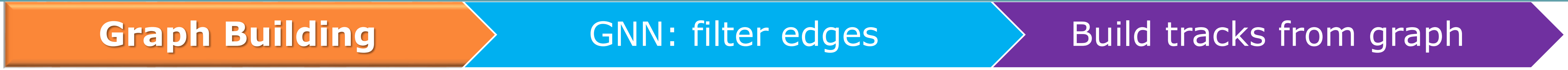
1 plane = **4 sensor planes**



P. C. Tsopelas, ‘A Silicon Pixel Detector for LHCb’, PhD Thesis, Vrije U., Amsterdam, 2016.



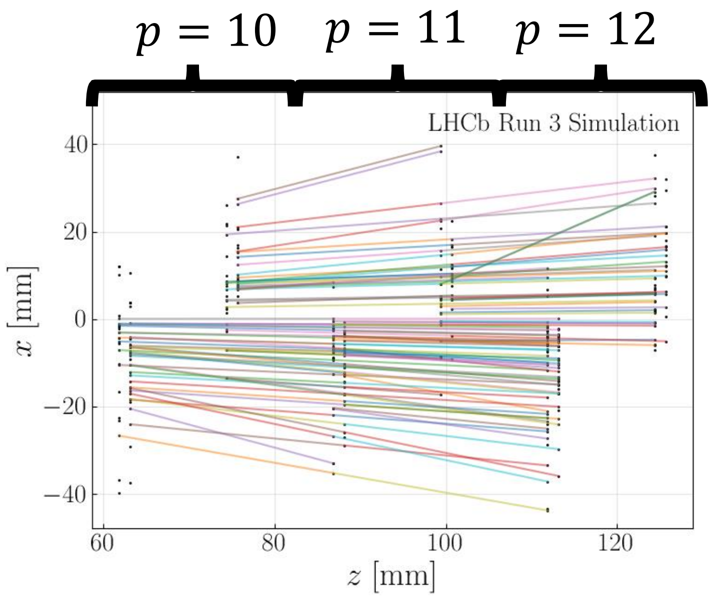
1. Graph Neural Network Track Finding



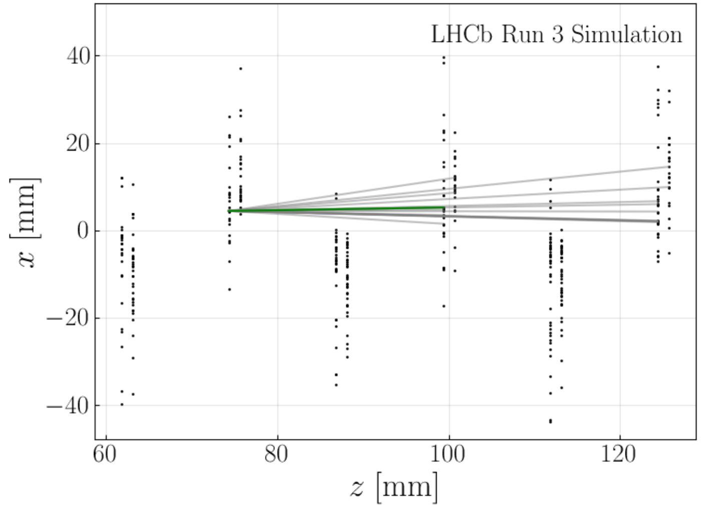
Edges are not random

- Forward
- Away from z-axis ↔ more tilted

⇒ this features could be learnt by a Neural Network



True edges



Example of edges drawn in the rough graph

1. Graph Neural Network Track Finding

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

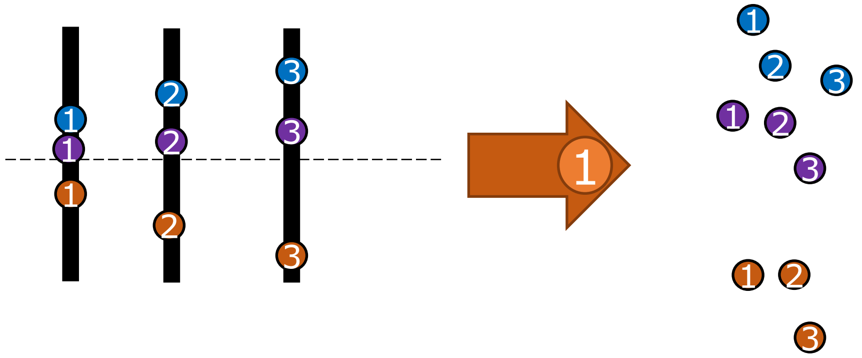
1 Embed every hit in an embedded space

Parallelise over hits

$$\left(\overbrace{r,\phi,z}^{\text{Cylindrical coordinates}},\text{plane}\right) \longrightarrow \boxed{\begin{array}{c}\text{Dense Neural Network}\\ \text{(DNN)}\\ \textbf{35K}\text{ parameters}\end{array}} \longrightarrow \vec{e} = (e_1, e_2, e_3, e_4)$$

DNN trained so that in the embedding space

- If hit A and hit B are likely to be connected by an edge $\boldsymbol{d}(\boldsymbol{A},\boldsymbol{B})^2 = \|\vec{e_A} - \vec{e_B}\|^2 < 0.010$
- Otherwise, $\boldsymbol{d}(\boldsymbol{A},\boldsymbol{B})^2 > 0.010$



1. Graph Neural Network Track Finding

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

1 **Embed every hit in an embedded space**

Parallelise over hits

$$\overbrace{(r,\phi,z}^{\text{Cylindrical coordinates}},\text{plane}) \longrightarrow \boxed{\begin{array}{c}\text{Dense Neural Network}\\\text{(DNN)}\\\textbf{35K}\text{ parameters}\end{array}} \longrightarrow \vec{e}=(e_1,e_2,e_3,e_4)$$

DNN trained so that in the embedding space

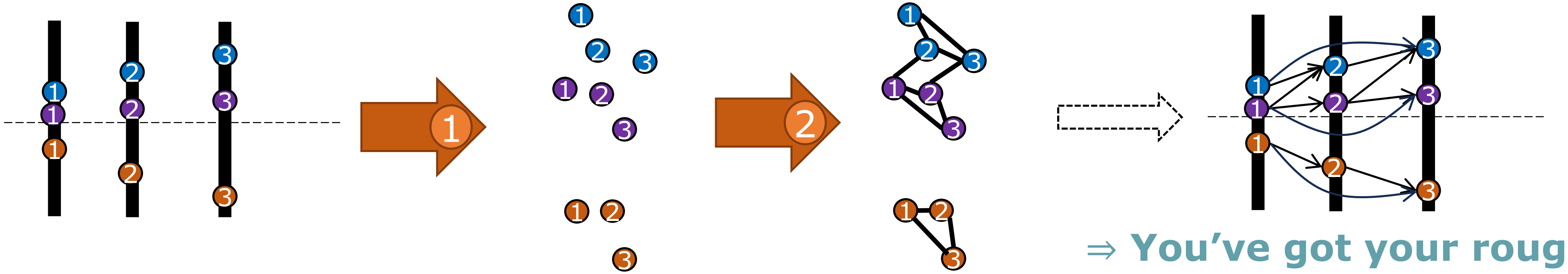
- If hit A and hit B are likely to be connected by an edge $\boldsymbol{d(A,B)}^2=\|\vec{e_A}-\vec{e_B}\|^2<0.010$
- Otherwise, $\boldsymbol{d(A,B)}^2>0.010$

2 Loop over plane $p\in\{0,\ldots,24\}$

Parallelise over hits

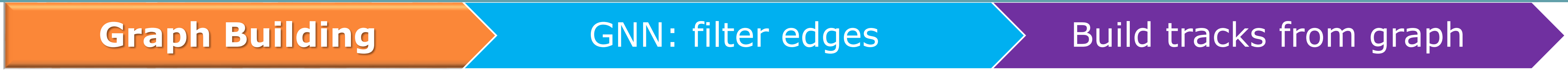
- Apply $\boldsymbol{k}_{\mathbf{max}}$-**Nearest Neighbour ($\boldsymbol{k}$NN)** algorithm between plane p and planes $\{p+1,p+2\}\Rightarrow k_{\max}$ edges / hit
- Discard edges for which $d^2>\boldsymbol{d}^{\mathbf{2}}_{\mathbf{max}}$

Parameters to optimise



⇒ You've got your rough graph

1. Graph Neural Network Track Finding



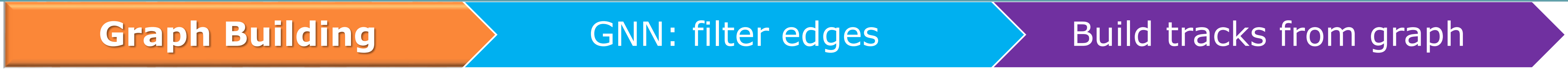
Graph Building

GNN: filter edges

Build tracks from graph

- Overall training strategy in back-up
- *After training*, we choose maximal number of neighbours $\boldsymbol{k}_{\mathbf{max}} = \mathbf{50}$ (not optimised)

1. Graph Neural Network Track Finding

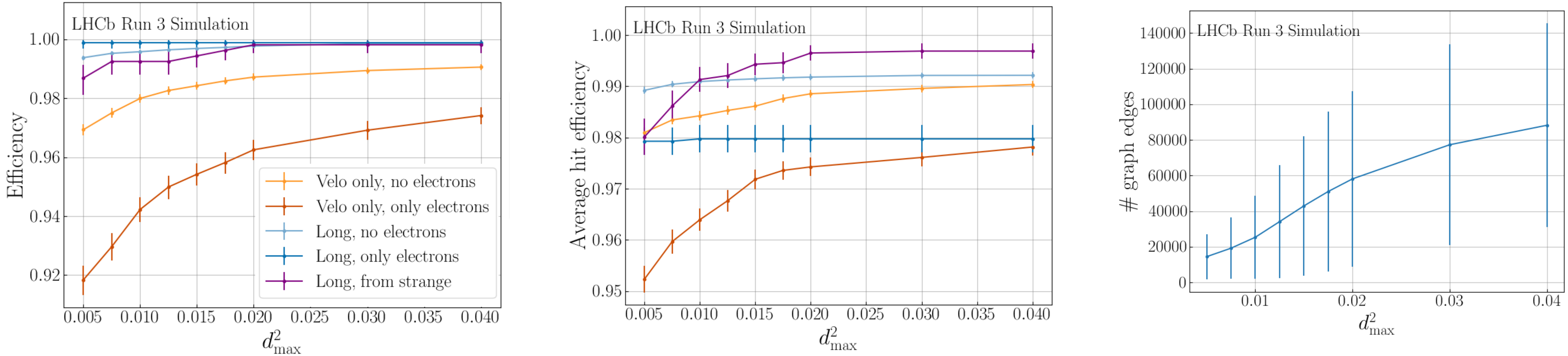


Graph Building → GNN: filter edges → Build tracks from graph

- Overall training strategy in back-up
- *After training*, we choose maximal number of neighbours $\boldsymbol{k}_{\mathbf{max}} = \mathbf{50}$ (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 $\boldsymbol{d}^2_{\mathbf{max}}$**

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



⇒ We will try $\boldsymbol{d}^2_{\mathbf{max}} = \mathbf{0.010}$ and $d^2_{\max} = 0.020$

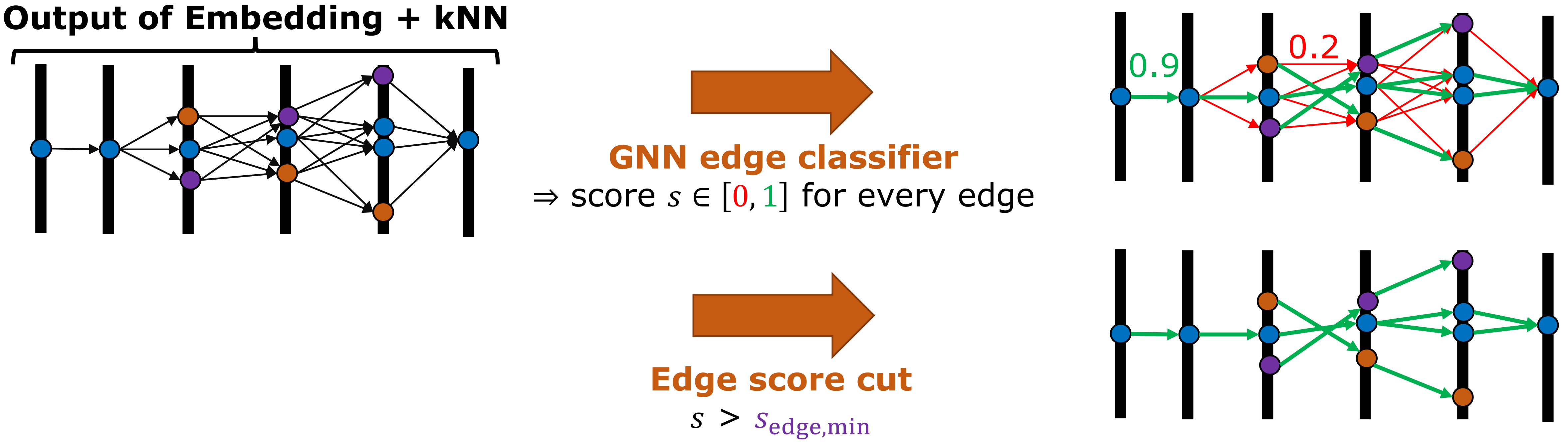
(evaluated on 200 events)

1. Graph Neural Network Track Finding

Graph Building

GNN: filter edges

Build tracks from graph



Output of Embedding + kNN

GNN edge classifier

⇒ score $s \in [0, 1]$ for every edge

Edge score cut

$s > s_{\text{edge,min}}$

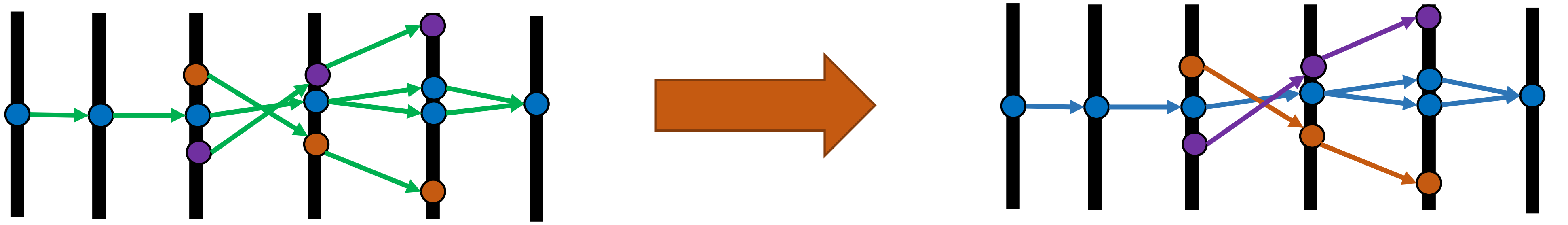
Change: Incoming and outgoing neighbours are **aggregated separately**, which increased overall GNN performance

1. Graph Neural Network Track Finding

Graph Building

GNN: filter edges

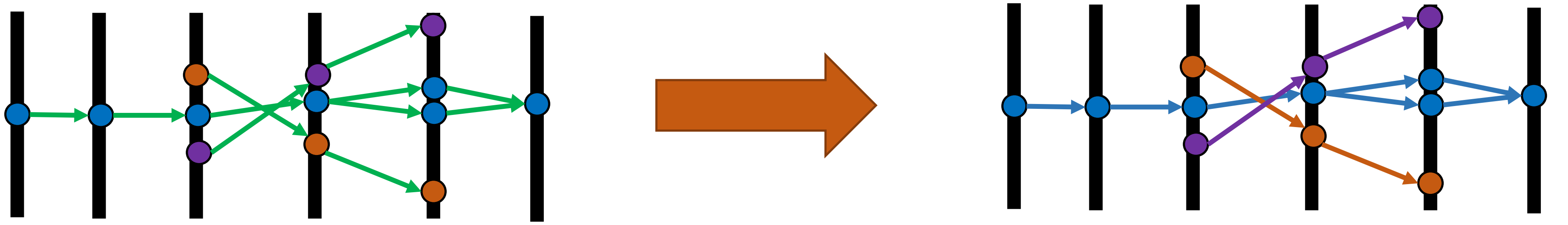
Build tracks from graph



Tracks obtained by identifying **connected hits**

1. Graph Neural Network Track Finding

Graph Building → GNN: filter edges → **Build tracks from graph**



Tracks obtained by identifying **connected hits**

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

Metric	Allen	etx4velo
Efficiency	**98.17%**	**46.23%**
Clone rate	3.07%	0.47%
Hit efficiency	95.35%	98.89%
Hit purity	99,67%	93.89%

🤮

(evaluated on 1000 events)

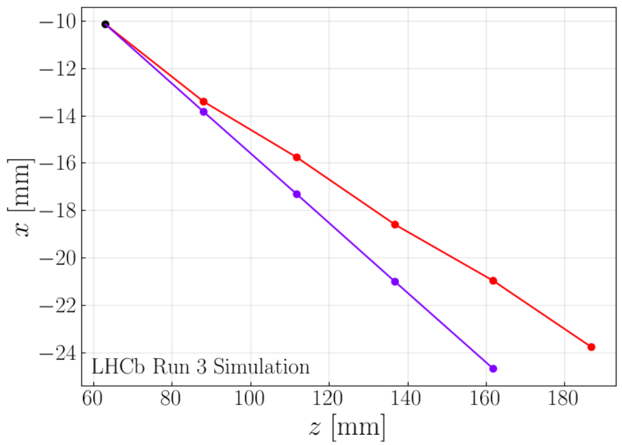
2. Issue of Shared Hits

The Case of Electrons

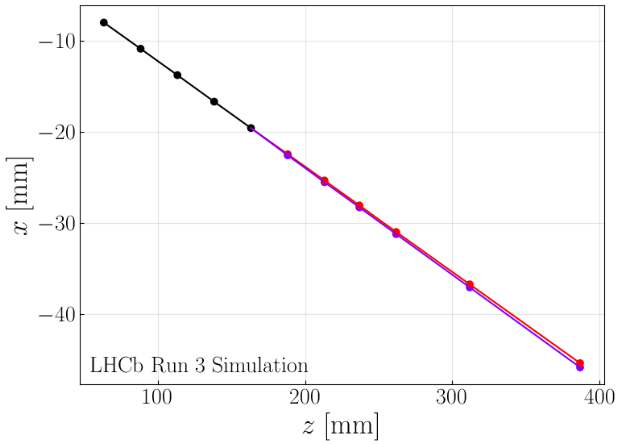
Observations

- $\sim$ 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

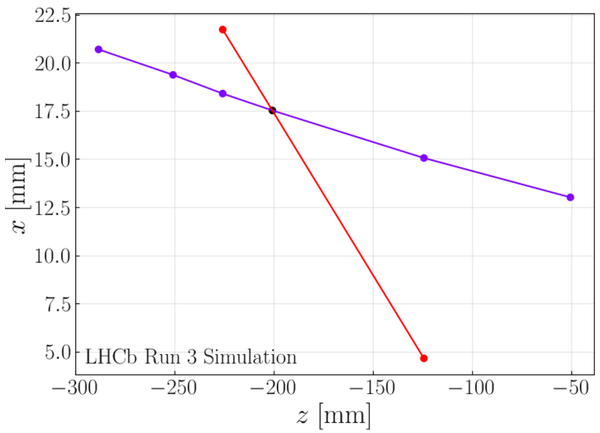


⇒ the **connected component algorithm** consider the **2** electron tracks as a **single** track

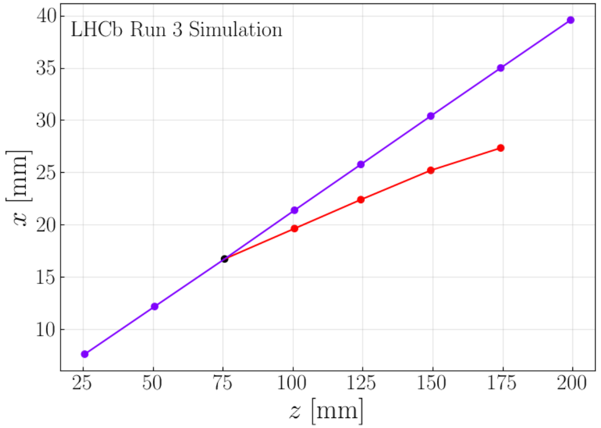
2. Issue of Shared Hits

Other Tracks With Shared Hits

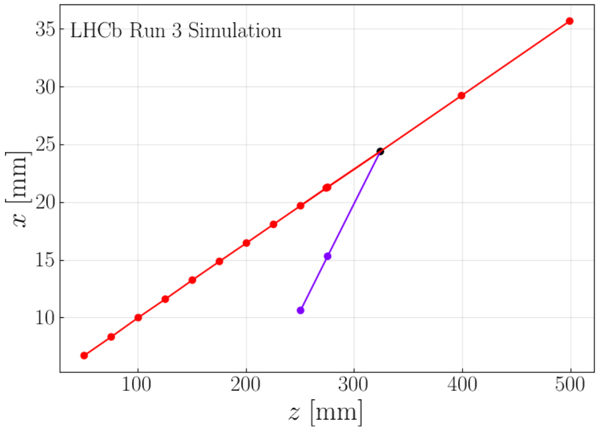
- **Tracks crossing** (> 524 in 1000 events)



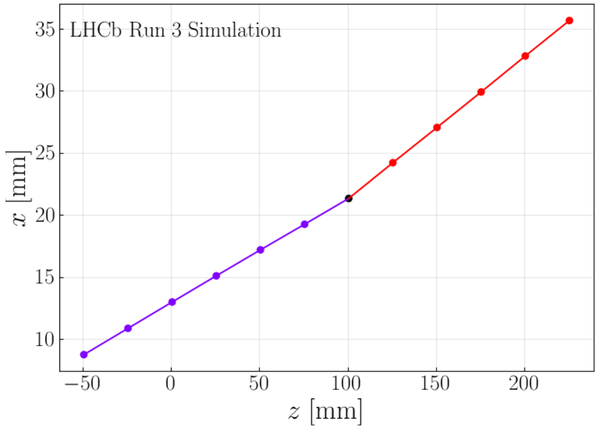
- **Track starts on a shared hit**



- **Track ends on a shared hit**

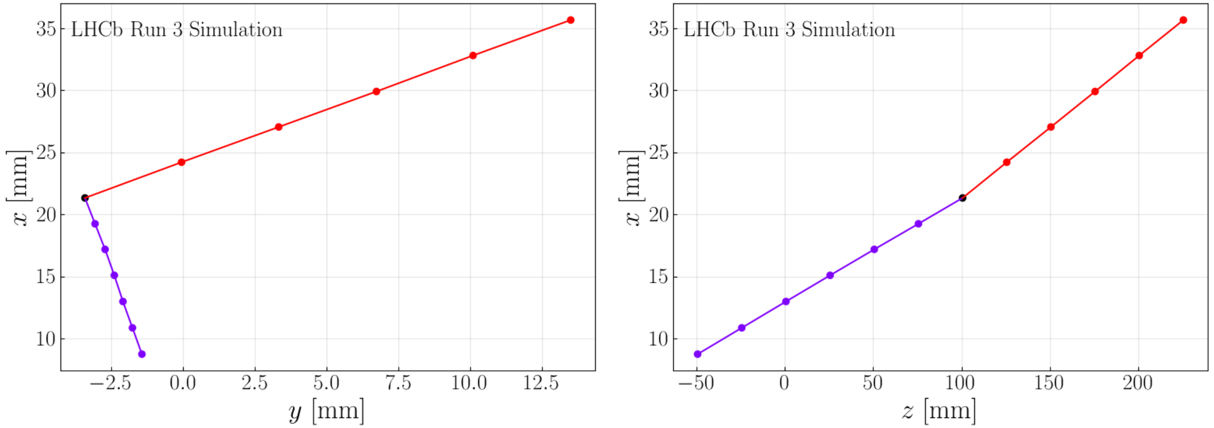


- **The last hit of a track is the first hit of another track** (>141 in 1000 events)



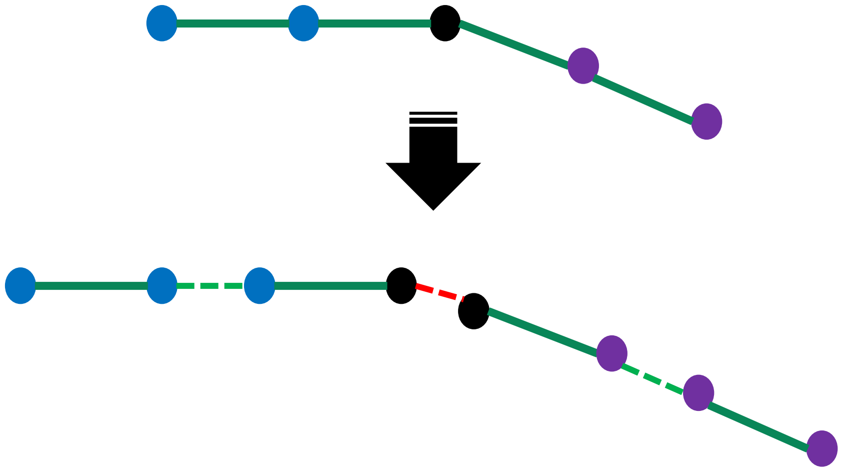
2. Issue of Shared Hits

Edge-Edge Connections



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

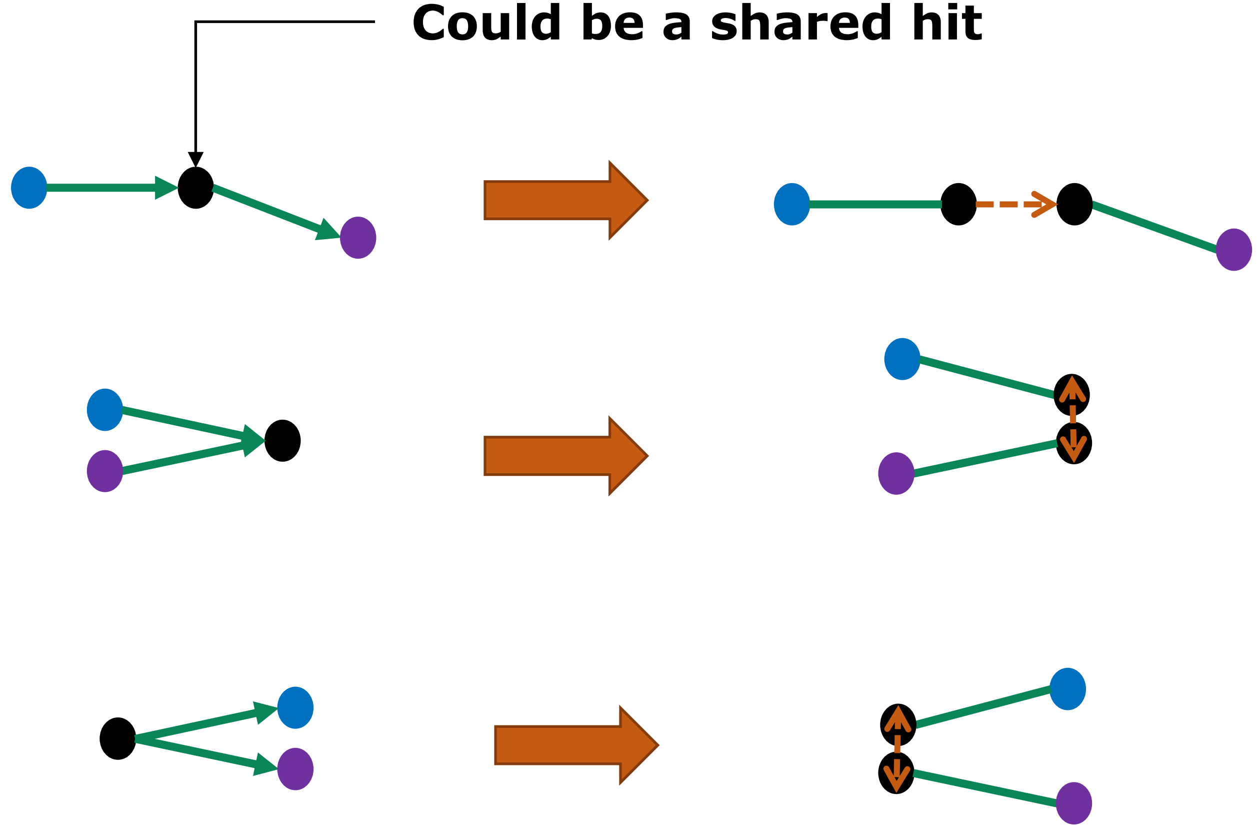
Hit-hit connection is not enough
⇒ need **edge-edge connections**



2. Issue of Shared Hits

Edge-Edge Connections

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



Could be a shared hit

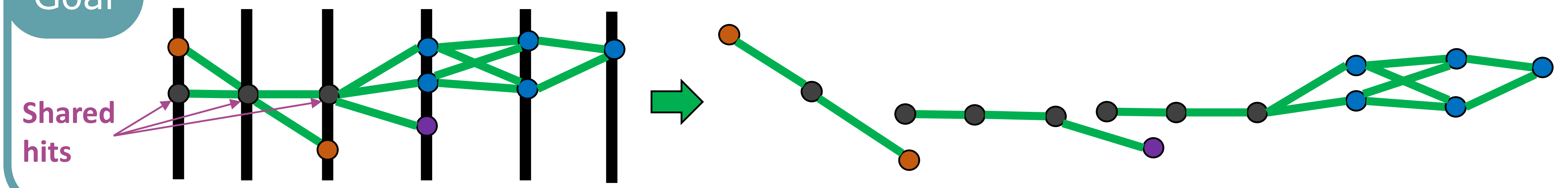
Articulation

Left elbow

Right elbow

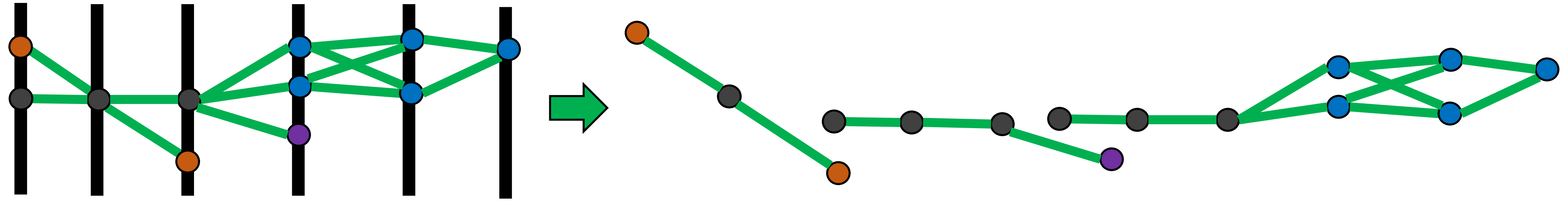
2. Issue of Shared Hits

Goal



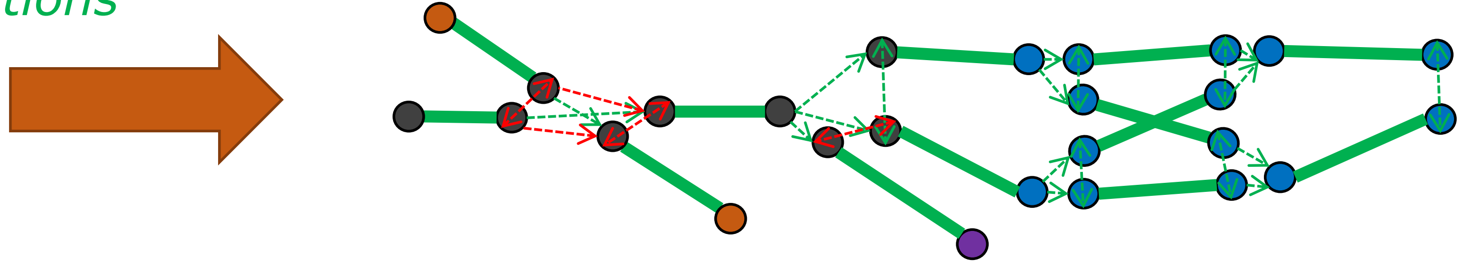
2. Issue of Shared Hits

Goal



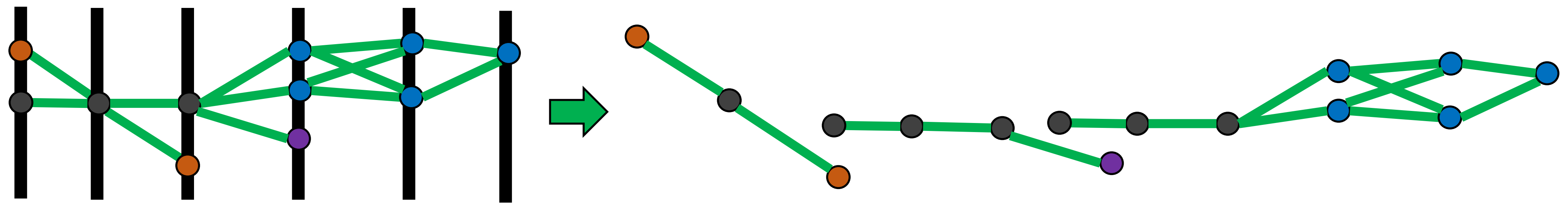
From the purified graph of hit-hit connections

1 Build edge-edge connections (or triplets)



2. Issue of Shared Hits

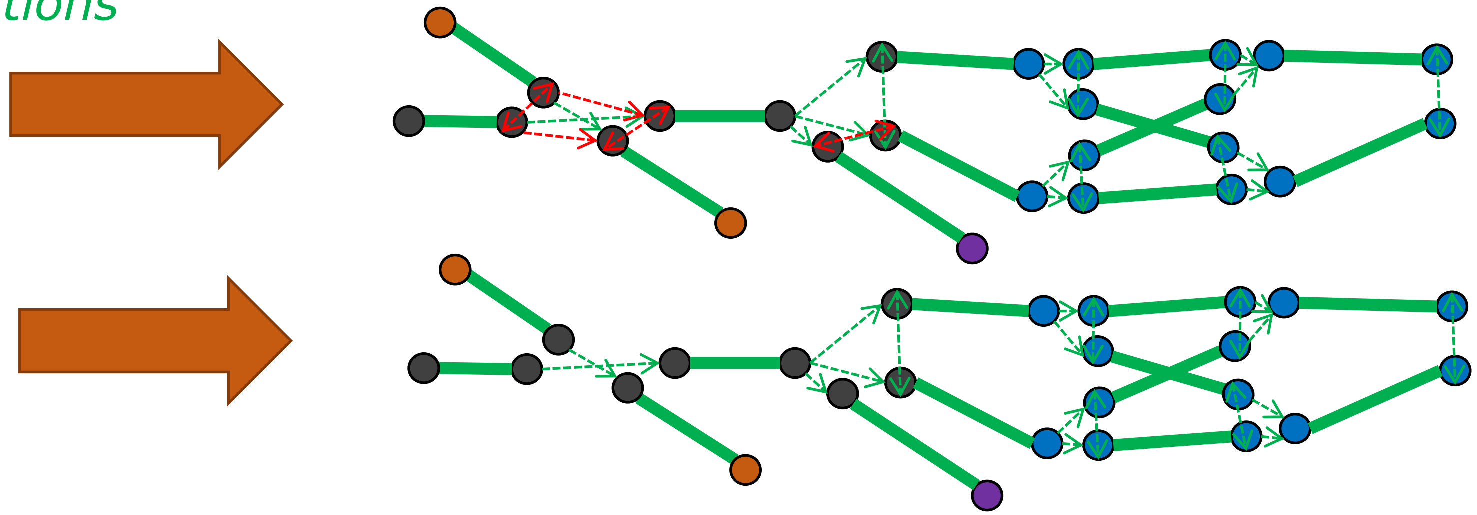
Goal



From the purified graph of hit-hit connections

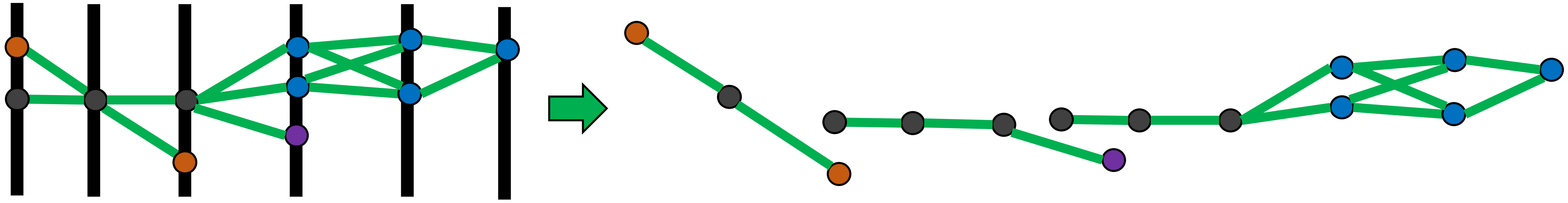
1 Build edge-edge connections
(or triplets)

2 Classify the triplets with the GNN
Filter out the fake triplets



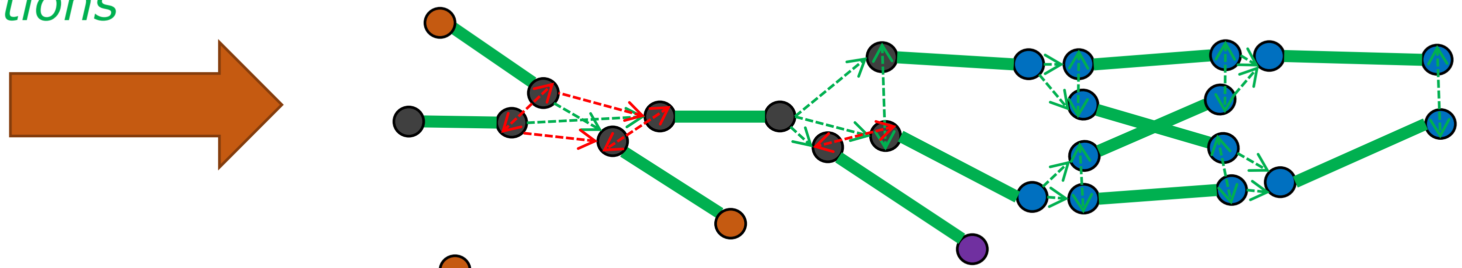
2. Issue of Shared Hits

Goal

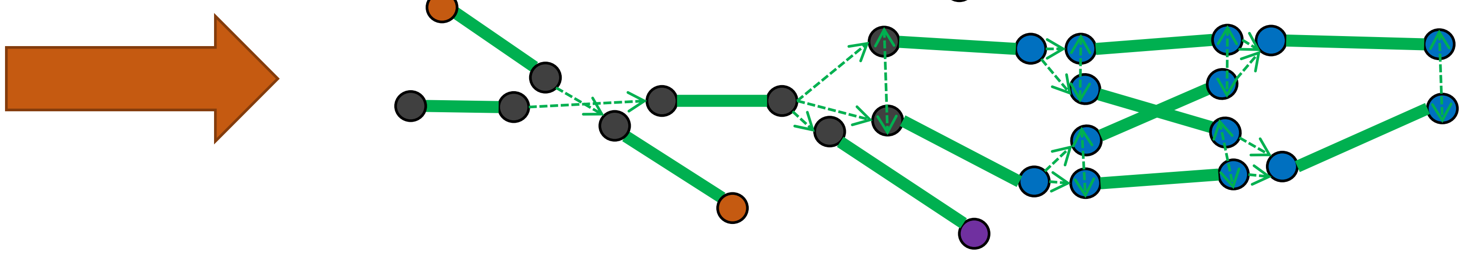


From the purified graph of hit-hit connections

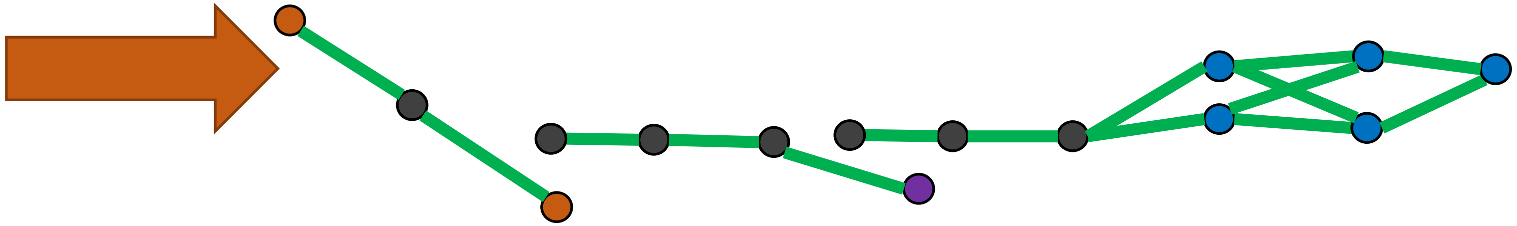
1 Build edge-edge connections (or triplets)



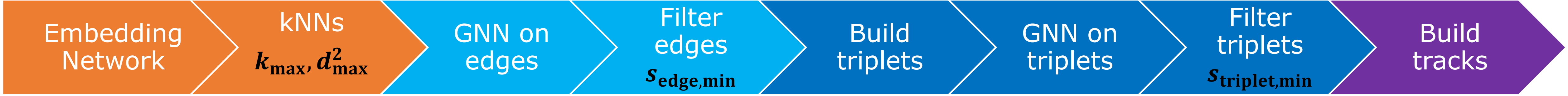
2 Classify the triplets with the GNN
Filter out the fake triplets



3 Algorithm to build tracks from triplets



2. Issue of Shared Hits



Embedding Network

kNNs $k_{\mathrm{max}}, d^2_{\mathrm{max}}$

GNN on edges

Filter edges $s_{\mathrm{edge,min}}$

Build triplets

GNN on triplets

Filter triplets $s_{\mathrm{triplet,min}}$

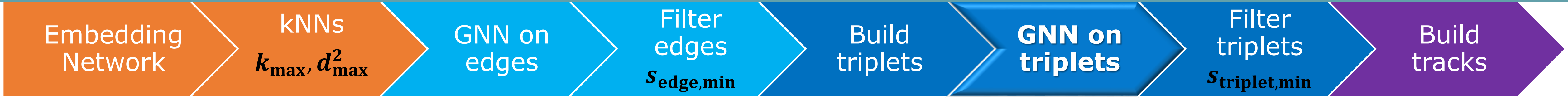
Build tracks

Build rough graph

Filter out fake edges

Filter out fake triplets

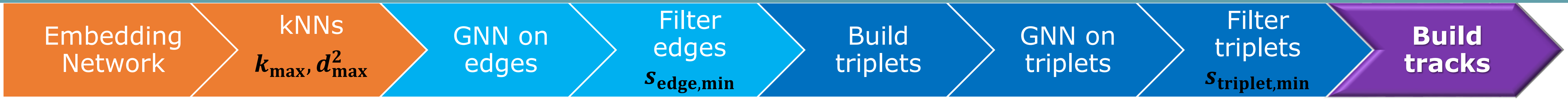
2. Issue of Shared Hits



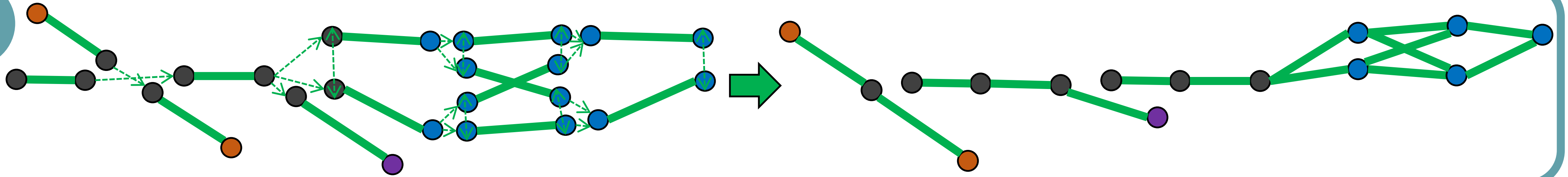
Don’t repeat the overall GNN inference: start from the previous GNN

- Compute **triplet score** from **node and edge encodings of the GNN**
- **Train GNN with overall loss** $\mathcal{L} = \mathcal{L}_{\mathrm{edges}} + \mathcal{L}_{\mathrm{triplets}}$

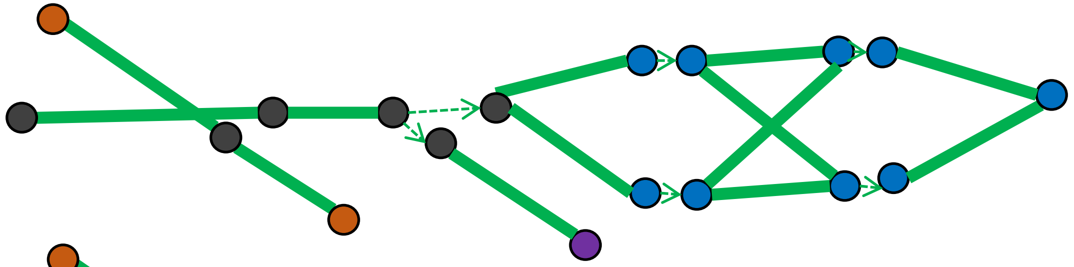
2. Issue of Shared Hits


Embedding Network
kNNs $k_{\max}, d^2_{\max}$
GNN on edges
Filter edges $s_{\text{edge,min}}$
Build triplets
GNN on triplets
Filter triplets $s_{\text{triplet,min}}$
Build tracks

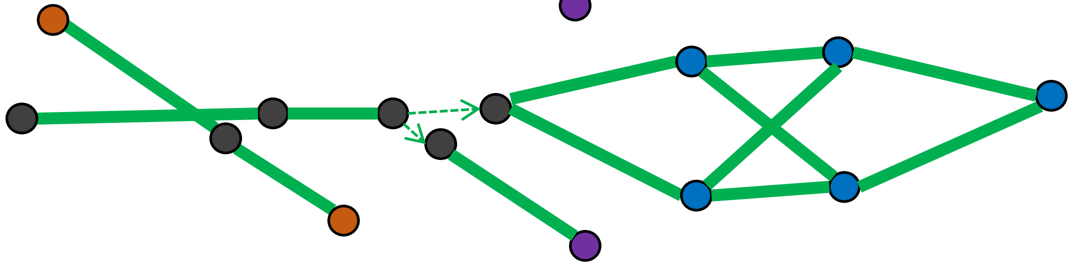
Goal



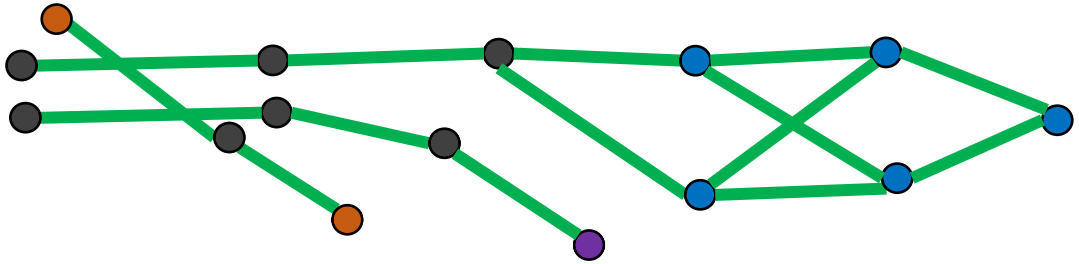
1 **Connect left and right elbows** and remove duplicate edge-edge connections



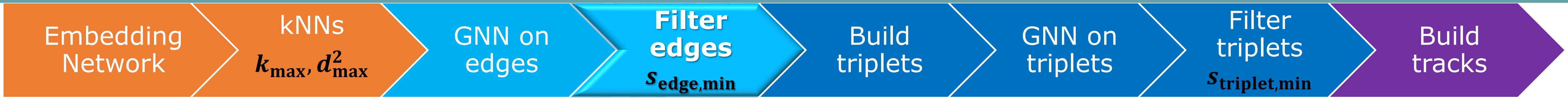
2 **Apply connected components**, excluding splitting edge-edge connections



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



3. Track-Finding Performance

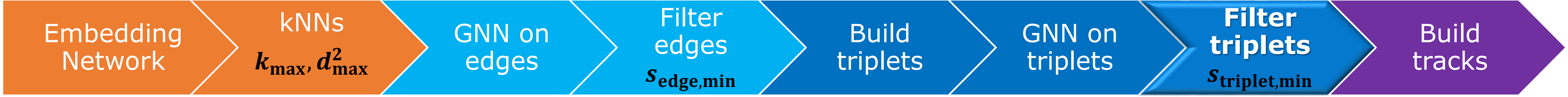


Embedding Network → kNNs $k_{\mathrm{max}}, d^2_{\mathrm{max}}$ → GNN on edges → **Filter edges** $s_{\mathrm{edge,min}}$ → Build triplets → GNN on triplets → Filter triplets $s_{\mathrm{triplet,min}}$ → Build tracks

Before building the tracks from the graph of triplets…

- Choose $s_{\mathrm{edge,min}} = 0.4$ to optimise performance *(could be increased to optimise throughput)*

3. Track-Finding Performance



Embedding Network

kNNs $k_{\max}, d^2_{\max}$

GNN on edges

Filter edges $s_{\text{edge,min}}$

Build triplets

GNN on triplets

Filter triplets $s_{\text{triplet,min}}$

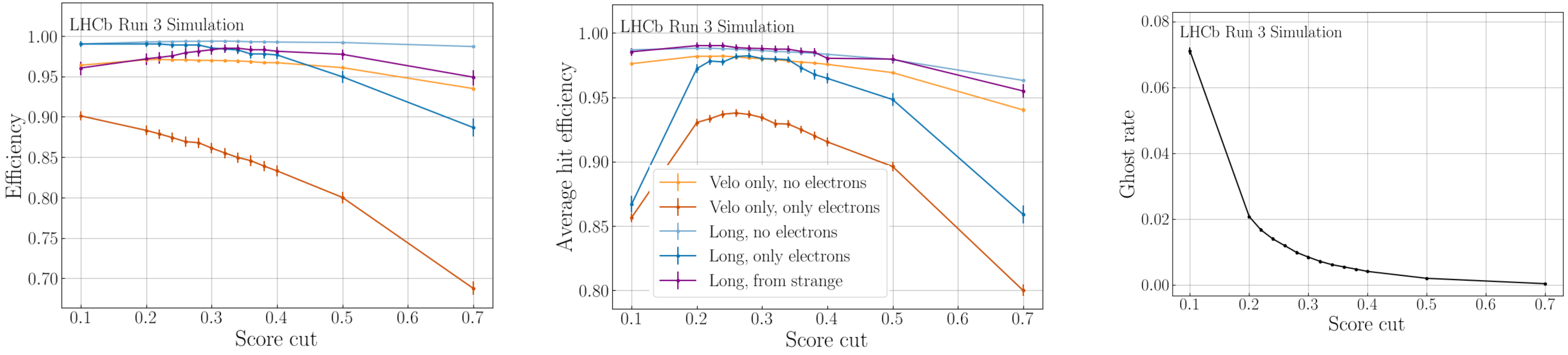
Build tracks

Before building the tracks from the graph of triplets

- Choose $s_{\text{edge,min}} = 0.4$ to optimise performance *(could be increased to optimise throughput)*
- Choose $s_{\text{triplet,min}}$ by evaluating track-finding performance as a function of $s_{\text{triplet,min}}$
  - High efficiency
  - Ghost rate $< 1\%$

Performance as a function of the triplet score cut $s_{\text{triplet,min}}$

(evaluated on 200 events)



⇒ choose $\boldsymbol{s_{\text{triplet,min}} = 0.32}$

3. Track-Finding Performance

Category	**Metric**
Long, no electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Not an electron	Efficiency
	Clone rate
	Hit efficiency
	Hit Purity
Long electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Electron	Efficiency
	Clone rate
	Hit efficiency
	Hit purity
Long, from strange ✓ In acceptance ✓ Reconstructible in the velo ✓ Decays from a strange *Good proxy for displaced tracks*	Efficiency
	Clone rate
	Hit efficiency
	Hit purity
X	Ghost rate

- Evaluation with 5,000 events
- **Track matched to a particle** if at least 70% of its hits belong to this particle

Long categories

3. Track-Finding Performance

Category	Metric	Allen
Long, no electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Not an electron	Efficiency	99.26%
	Clone rate	2.54%
	Hit efficiency	96.46%
	Hit Purity	99.78%
Long electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Electron	Efficiency	97.11%
	Clone rate	4,25%
	Hit efficiency	95.24%
	Hit purity	97.11%
Long, from strange ✓ In acceptance ✓ Reconstructible in the velo ✓ Decays from a strange *Good proxy for displaced tracks*	Efficiency	97.69%
	Clone rate	2.50%
	Hit efficiency	97.69%
	Hit purity	99.34%
X	Ghost rate	2.18%

- Evaluation with 5,000 events

- **Track matched to a particle** if at least 70% of its hits belong to this particle

- Allen algorithm described in arXiv:2207.03936v2

Long categories

🟥 🟥 🟧 🟩 🟩 🟩

Worse **Better**

3. Track-Finding Performance

Category	Metric	Allen	$s_{\text{triplet}} > 0.32$ Etx4velo $d^2_{\text{max}} = 0.010$
Long, no electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Not an electron	Efficiency	99.26%	99.28%
	Clone rate	2.54%	0.96%
	Hit efficiency	96.46%	98.73%
	Hit Purity	99.78%	99.94%
Long electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Electron	Efficiency	97.11%	98.80%
	Clone rate	4,25%	7.42%
	Hit efficiency	95.24%	96.54%
	Hit purity	97.11%	98.46%
Long, from strange ✓ In acceptance ✓ Reconstructible in the velo ✓ Decays from a strange *Good proxy for displaced tracks*	Efficiency	97.69%	97.50%
	Clone rate	2.50%	0.92%
	Hit efficiency	97.69%	98.22%
	Hit purity	99.34%	99.68%
X	Ghost rate	2.18%	0.76%

- Evaluation with 5,000 events
- **Track matched to a particle** if at least 70% of its hits belong to this particle
- Allen algorithm described in arXiv:2207.03936v2

Long categories

🟥🟥🟧🟩🟩🟩

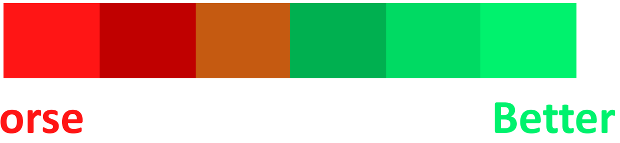
Worse Better

3. Track-Finding Performance

			$s_{\text{triplet}} > 0.32$	$s_{\text{triplet}} > 0.36$
Category	**Metric**	**Allen**	Etx4velo $d^2_{\text{max}} = 0.010$	Etx4velo $d^2_{\text{max}} = 0.020$
Long, no electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Not an electron	Efficiency	99.26%	99.28%	99.51%
	Clone rate	2.54%	0.96%	0.89%
	Hit efficiency	96.46%	98.73%	98.90%
	Hit Purity	99.78%	99.94%	99.94%
Long electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Reconstructible in the SciFi ✓ Electron	Efficiency	97.11%	98.80%	99.22%
	Clone rate	4,25%	7.42%	7.31%
	Hit efficiency	95.24%	96.54%	96.79%
	Hit purity	97.11%	98.46%	98.46%
Long, from strange ✓ In acceptance ✓ Reconstructible in the velo ✓ Decays from a strange *Good proxy for displaced tracks*	Efficiency	97.69%	97.50%	98.06%
	Clone rate	2.50%	0.92%	0.81%
	Hit efficiency	97.69%	98.22%	98.77%
	Hit purity	99.34%	99.68%	99.68%
X	Ghost rate	2.18%	0.76%	0.81%

- Evaluation with 5,000 events
- **Track matched to a particle** if at least 70% of its hits belong to this particle
- Allen algorithm described in arXiv:2207.03936v2
- 2 different GNN trainings for $d^2_{\text{max}} = 0.010$ and $d^2_{\text{max}} = 0.020$

Long categories



3. Track-Finding Performance

			$s_{\text{triplet}} > 0.32$	$s_{\text{triplet}} > 0.36$
Category	**Metric**	**Allen**	Etx4velo $d^2_{\text{max}} = 0.010$	Etx4velo $d^2_{\text{max}} = 0.020$
Velo-only, no electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Not reconstructible in the SciFi ✓ Not an electron	Efficiency	96.84%	97.03%	97.86%
	Clone rate	3.84%	1.08%	1.02%
	Hit efficiency	93.89%	97.93%	98.32%
	Hit Purity	99.50%	99.84%	99.82%
Velo-only electrons ✓ In acceptance ✓ Reconstructible in the velo ✓ Not reconstructible in the SciFi ✓ Electron	Efficiency	67.81%	85.10%	86.69%
	Clone rate	10.27%	5.02%	4.97%
	Hit efficiency	79.21%	93.33%	93.88%
	Hit purity	97.35%	99.07%	98.99%
Velo-only, from strange ✓ In acceptance ✓ Not reconstructible in the velo ✓ Decays from a strange *Good proxy for displaced tracks*	Efficiency	93.53%	93.07%	96.05%
	Clone rate	5.60%	1.97%	1.77%
	Hit efficiency	90.05%	93.92%	96.05%
	Hit purity	99.36%	99.67%	99.64%

Velo-only categories

🟥🟥🟧🟩🟩🟩

Worse Better

Conclusion

Track-Finding Physics Performance of GNN-based pipeline

- **Comparable or superior performance** to Allen’s velo track-finding algorithm
- **Excellent electron reconstruction**
- **Low ghost rate**

Ongoing Work

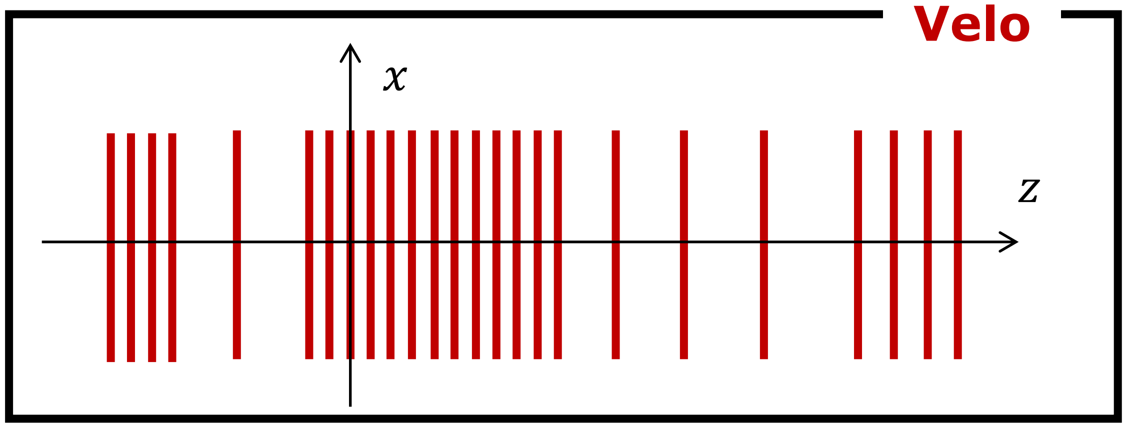
- Implementation in Allen to
  - properly **optimise the throughput** of the GNN-based pipeline
  - **Compare the optimal throughput to conventional algorithm**

- Extension to other LHCb tracking detectors, starting from the SciFi

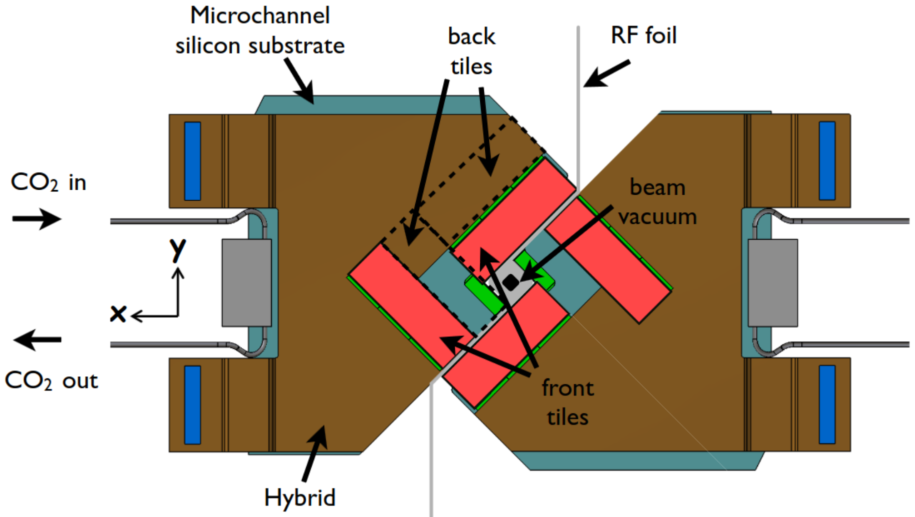
Thank You For Your Attention!

Backup Slides

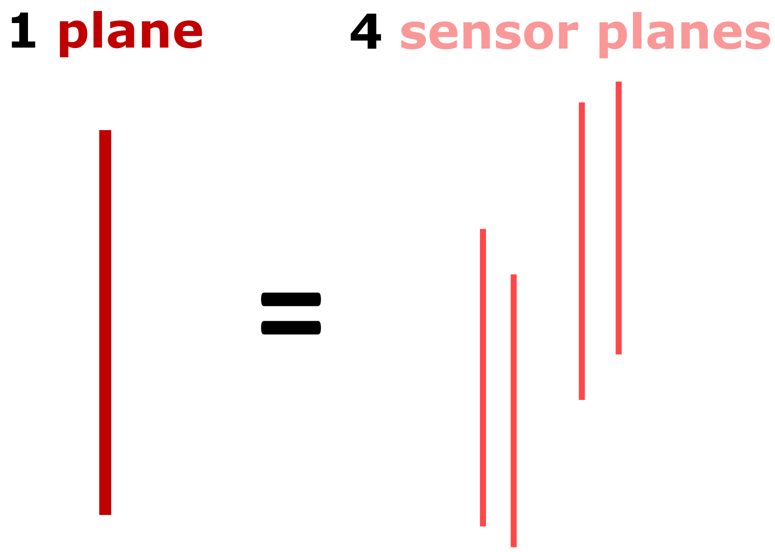
Velo geometry



1 plane = 4 sensor planes



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


1 plane = 4 sensor planes

1. Graph Neural Network Track Finding

Use **700,000 events** for training, with the following selection

- **Particles** are **straight enough**
- **Particles** leave ≥ **3 hits** in the Velo
- **Event** has ≥ 500 genuine hits

1. GNN-based Track Finding Approach

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

Training step

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

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:

   $\{d^2_{\text{genuine},i}, \forall i\}$ and $\{d^2_{\text{fake},j}, \forall j\}$

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

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

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

$$\mathcal{L}_{\text{fake}} = \frac{1}{n_{\text{fake}}}\sum_j \max\left(\overset{\text{hyperparameter}}{0.01} - d^2_{\text{fake},j}, 0\right)$$

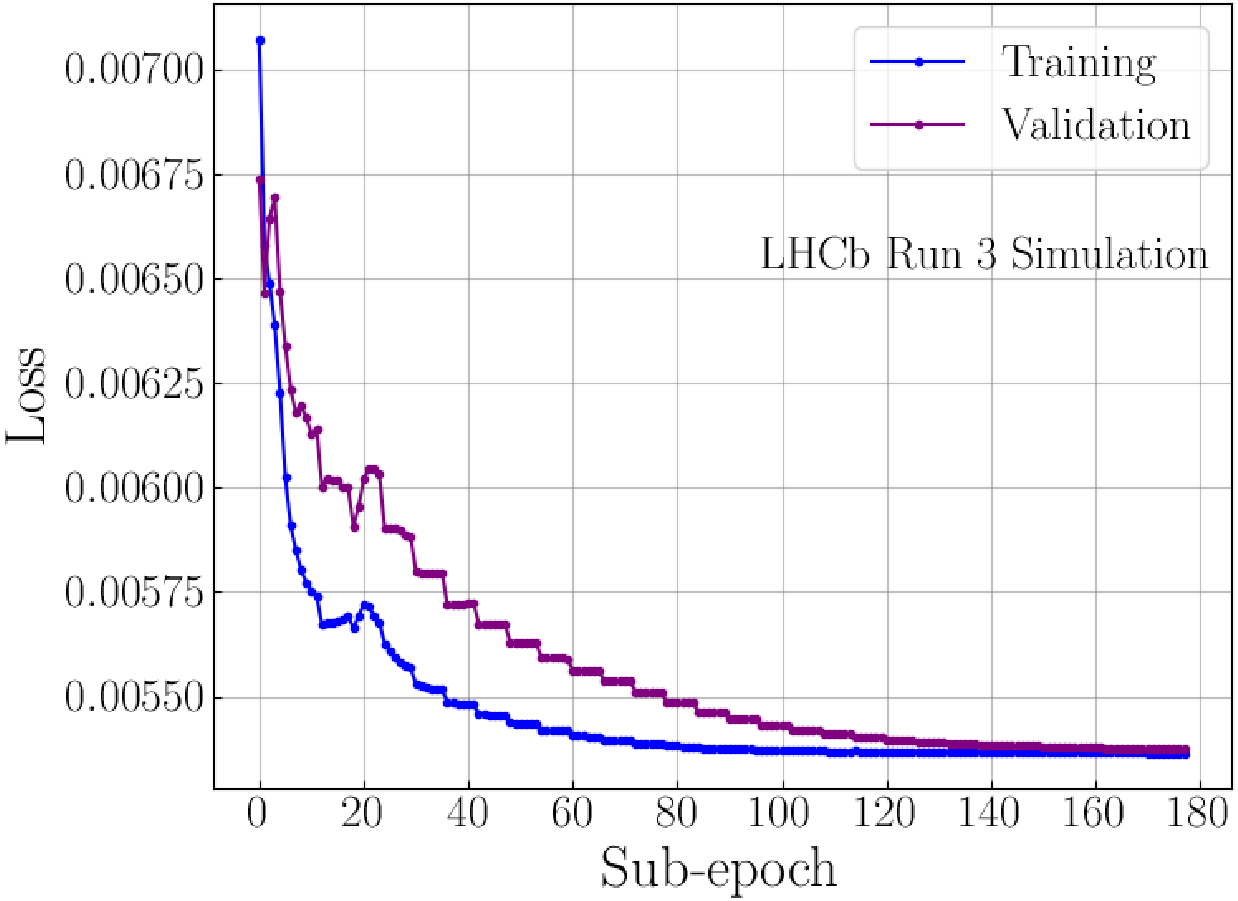
Maximise $d_{\text{fake},j}$

Training dataset

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

1. GNN-based Track Finding Approach

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

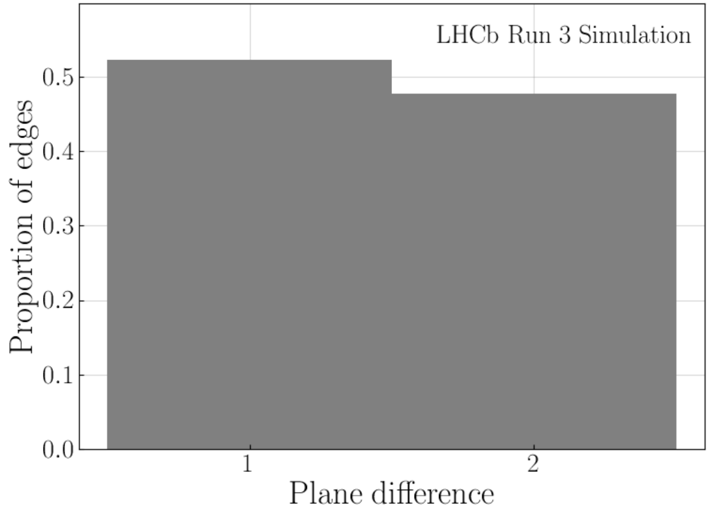


Training set of 700,000 events divided into sub-epochs of 7,000 events

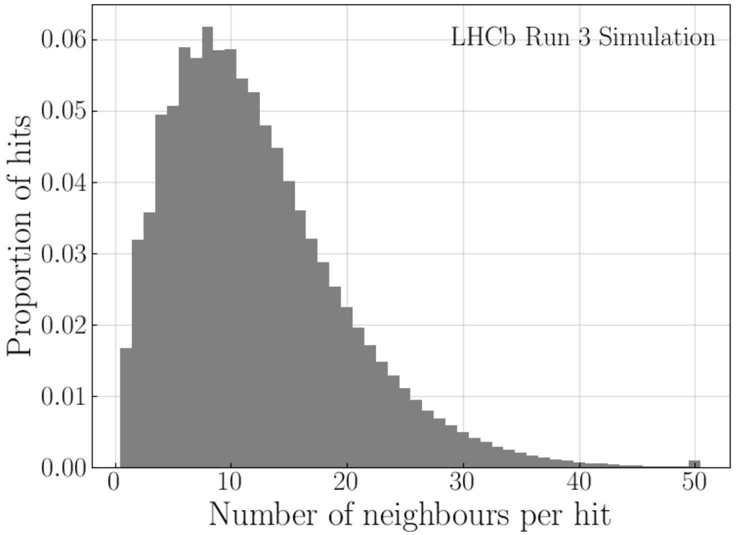
1. GNN-based Track Finding Approach

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

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**

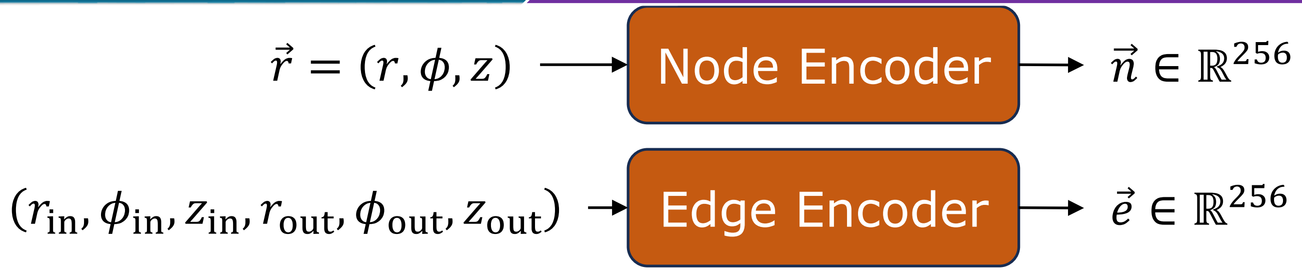


$\Rightarrow$ $\boldsymbol{k_{\max}}$ **could probably be reduced** to increase throughput

1. Graph Neural Network Track Finding

Graph Building → **GNN: filter edges** → Build tracks from graph

1 **Encode every hit and edge** in a high-dimensional space



$\vec{r} = (r, \phi, z) \rightarrow$ Node Encoder $\rightarrow \vec{n} \in \mathbb{R}^{256}$

$(r_{\text{in}}, \phi_{\text{in}}, z_{\text{in}}, r_{\text{out}}, \phi_{\text{out}}, z_{\text{out}}) \rightarrow$ Edge Encoder $\rightarrow \vec{e} \in \mathbb{R}^{256}$

1. Graph Neural Network Track Finding

Graph Building → **GNN: filter edges** → Build tracks from graph

1 **Encode every hit and edge** in a high-dimensional space

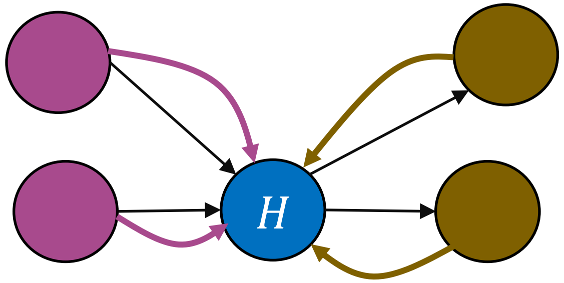
$$\vec{r} = (r, \phi, z) \longrightarrow \boxed{\text{Node Encoder}} \longrightarrow \vec{n} \in \mathbb{R}^{256}$$

$$(r_{\text{in}}, \phi_{\text{in}}, z_{\text{in}}, r_{\text{out}}, \phi_{\text{out}}, z_{\text{out}}) \longrightarrow \boxed{\text{Edge Encoder}} \longrightarrow \vec{e} \in \mathbb{R}^{256}$$

hyperparameter

2 **Message passing**: repeat 6 times

a Build "**message**" by aggregating neighbour hit encodings



Message = [$\overrightarrow{\textbf{max}}(\{\vec{n}_{\text{input}}\})$, $\overrightarrow{\textbf{sum}}(\{\vec{n}_{\text{input}}\})$, $\overrightarrow{\textbf{max}}(\{\vec{n}_{\text{output}}\})$, $\overrightarrow{\textbf{sum}}(\{\vec{n}_{\text{output}}\})$]

b Update edge and node encodings

$$[\vec{n}, \textbf{message}] \longrightarrow \boxed{\text{Node Network}} \longrightarrow \oplus \longrightarrow \overrightarrow{n_{\text{updated}}}$$

$$[\vec{\text{e}}, \vec{n}^{in}_{\text{updated}}, \vec{n}^{\text{out}}_{\text{updated}}] \longrightarrow \boxed{\text{Edge Network}} \longrightarrow \oplus \longrightarrow \overrightarrow{e_{\text{updated}}}$$

1. Graph Neural Network Track Finding

Graph Building | **GNN: filter edges** | Build tracks from graph

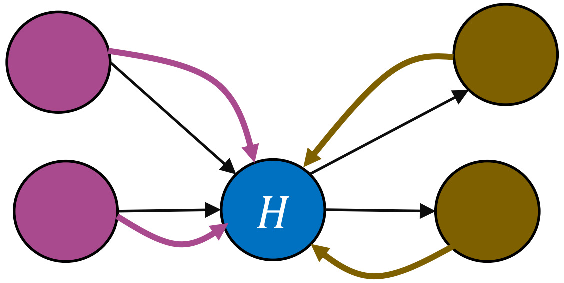
1 **Encode every hit and edge** in a high-dimensional space

$\vec{r} = (r, \phi, z) \rightarrow$ Node Encoder $\rightarrow \vec{n} \in \mathbb{R}^{256}$

$(r_{\text{in}}, \phi_{\text{in}}, z_{\text{in}}, r_{\text{out}}, \phi_{\text{out}}, z_{\text{out}}) \rightarrow$ Edge Encoder $\rightarrow \vec{e} \in \mathbb{R}^{256}$

2 **Message passing**: repeat 6 times ← hyperparameter

a Build "**message**" by aggregating neighbour hit encodings



Message = [$\overrightarrow{\textbf{max}}(\{\vec{n}_{\text{input}}\})$, $\overrightarrow{\textbf{sum}}(\{\vec{n}_{\text{input}}\})$, $\overrightarrow{\textbf{max}}(\{\vec{n}_{\text{output}}\})$, $\overrightarrow{\textbf{sum}}(\{\vec{n}_{\text{output}}\})$ **]**

b Update edge and node encodings

$[\vec{n}$, **message**$] \rightarrow$ Node Network $\rightarrow \oplus \rightarrow \overrightarrow{n_{\text{updated}}}$

$[\vec{e}, \vec{n}^{in}_{\text{updated}}, \vec{n}^{\text{out}}_{\text{updated}}] \rightarrow$ Edge Network $\rightarrow \oplus \rightarrow \overrightarrow{e_{\text{updated}}}$

3 **Compute edge scores**

$[\vec{n}_{\text{in}}, \vec{n}_{\text{out}}, \vec{e}] \rightarrow$ Edge Classifier $\rightarrow$ Edge score $s \in [0, 1]$

Trained with a sigmoid focal loss

1. Graph Neural Network Track Finding

Graph Building → **GNN: filter edges** → Build tracks from graph

1 **Encode every hit and edge** in a high-dimensional space

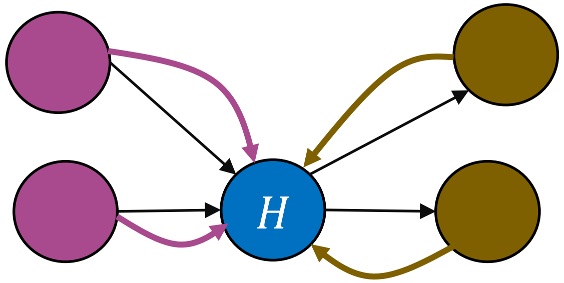
$\vec{r} = (r, \phi, z) \rightarrow$ Node Encoder $\rightarrow \vec{n} \in \mathbb{R}^{256}$

$(r_{\text{in}}, \phi_{\text{in}}, z_{\text{in}}, r_{\text{out}}, \phi_{\text{out}}, z_{\text{out}}) \rightarrow$ Edge Encoder $\rightarrow \vec{e} \in \mathbb{R}^{256}$

hyperparameter

2 **Message passing**: repeat 6 times

a Build "**message**" by aggregating neighbour hit encodings



Change w.r.t. Exa.Trkx

Incoming and outgoing neighbours are **aggregated separately**

Message = [$\overrightarrow{\max}(\{\vec{n}_{\text{input}}\})$, $\overrightarrow{\text{sum}}(\{\vec{n}_{\text{input}}\})$, $\overrightarrow{\max}(\{\vec{n}_{\text{output}}\})$, $\overrightarrow{\text{sum}}(\{\vec{n}_{\text{output}}\})$]

b Update edge and node encodings

$[\vec{n}$, **message**$] \rightarrow$ Node Network $\rightarrow \oplus \rightarrow \overrightarrow{n_{\text{updated}}}$

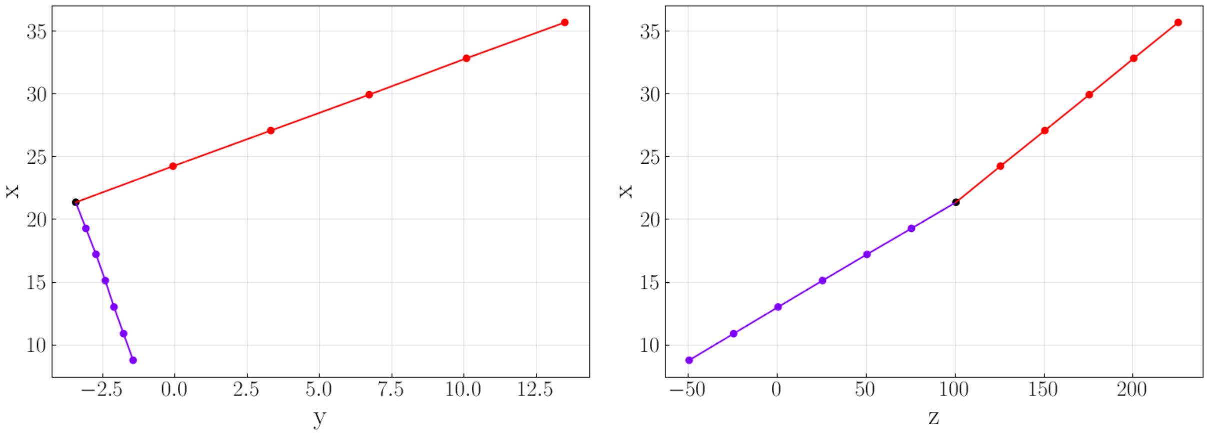
$[\vec{e}, \vec{n}^{in}_{\text{updated}}, \vec{n}^{out}_{\text{updated}}] \rightarrow$ Edge Network $\rightarrow \oplus \rightarrow \overrightarrow{e_{\text{updated}}}$

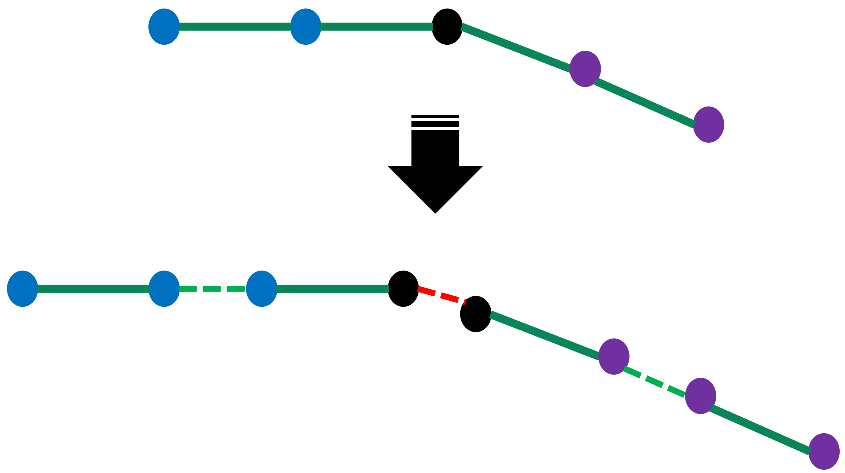
3 **Compute edge scores**

$[\vec{n}_{\text{in}}, \vec{n}_{\text{out}}, \vec{e}] \rightarrow$ Edge Classifier $\rightarrow$ Edge score $s \in [0, 1]$

Trained with a sigmoid focal loss

2. Issue of Shared Hits

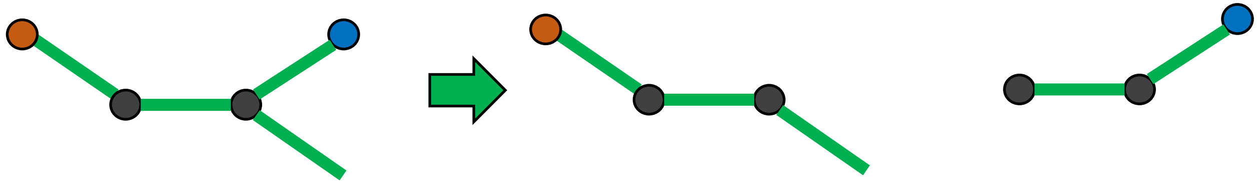




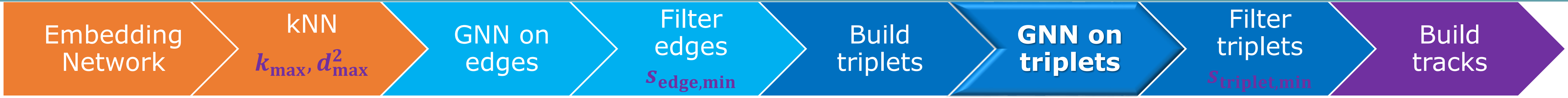
Hit-hit connection is not enough
⇒ need **edge-edge connections**

- Solve the ambiguity of shared hits under the following hypothesis:
  "All hits that precede a splitting point can be attributed to all the newly identified tracks"

- ⇒ Assume that this does not happen



2. Issue of Shared Hits



Embedding Network

kNN $k_{\max}, d^2_{\max}$

GNN on edges

Filter edges $s_{\text{edge,min}}$

Build triplets

GNN on triplets

Filter triplets $s_{\text{triplet,min}}$

Build tracks

Don’t repeat the 6-step message passing: **start from the previous GNN**

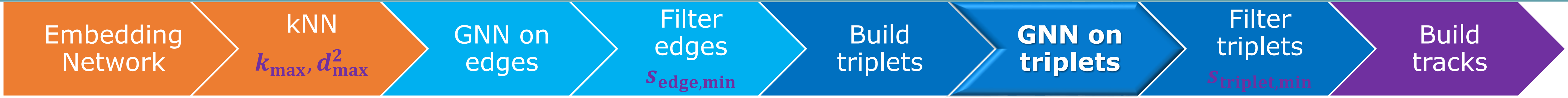
3 **Compute edge scores**

$[\vec{n}_{\text{in}}, \vec{n}_{\text{out}}, \vec{e}]$ → Edge Classifier → Edge score $s_{\text{edge}} \in [0, 1]$

4 Filter out the fake edges by requiring $s_{\text{edge}} > s_{\text{edge,min}}$ to reduce # edge-edge connections

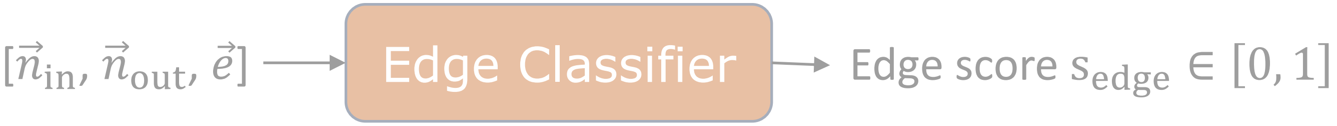
5 Build **triplets**

2. Issue of Shared Hits



Don’t repeat the 6-step message passing: **start from the previous GNN**

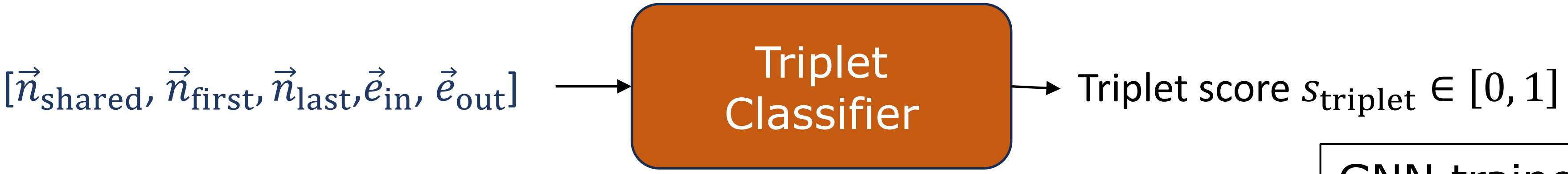
3 **Compute edge scores**


$[\vec{n}_{\text{in}}, \vec{n}_{\text{out}}, \vec{e}]$
Edge Classifier
Edge score $s_{\text{edge}} \in [0, 1]$

4 Filter out the fake edges by requiring $s_{\text{edge}} > s_{\text{edge,min}}$ to reduce # edge-edge connections

5 Build **triplets**

6 **Directly compute triplet scores from the edge and node encodings of the triplet**

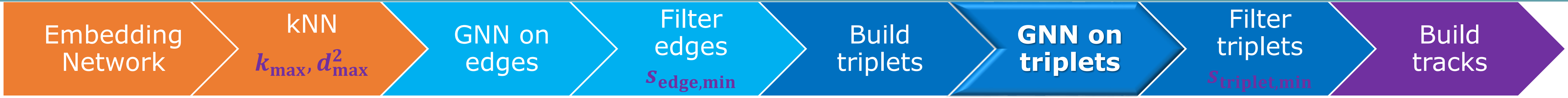

$[\vec{n}_{\text{shared}}, \vec{n}_{\text{first}}, \vec{n}_{\text{last}}, \vec{e}_{\text{in}}, \vec{e}_{\text{out}}]$
Triplet Classifier
Triplet score $s_{\text{triplet}} \in [0, 1]$

Filter out the fake triplets by requiring $s_{\text{triplet}} > s_{\text{triplet,min}}$

GNN trained with the overall loss

$$\mathcal{L}_{\text{tot}} = \mathcal{L}_{\text{edge}} + \mathcal{L}_{\text{triplet}}$$

2. Issue of Shared Hits



Embedding Network

kNN $k_{\text{max}}, d^2_{\text{max}}$

GNN on edges

Filter edges $s_{\text{edge,min}}$

Build triplets

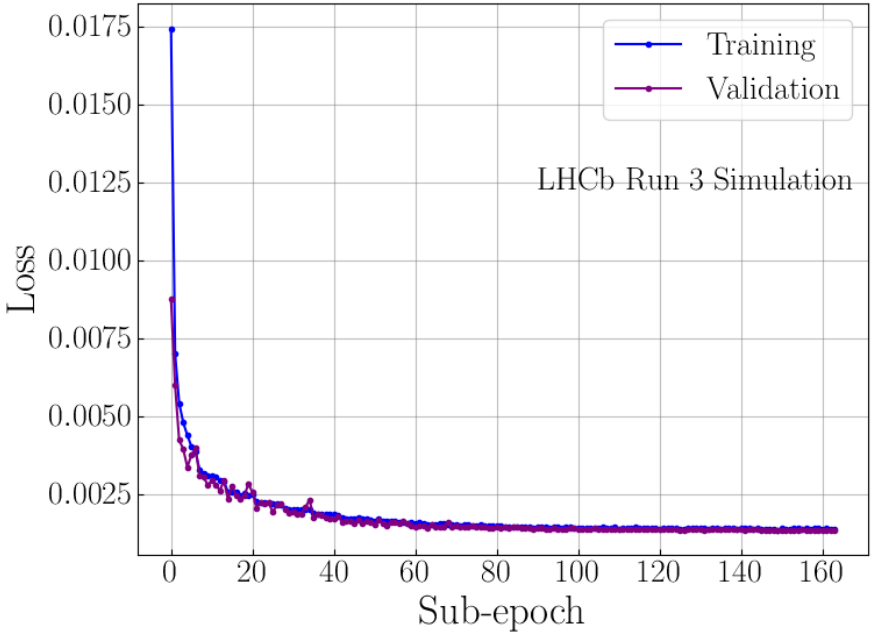
GNN on triplets

Filter triplets $s_{\text{triplet,min}}$

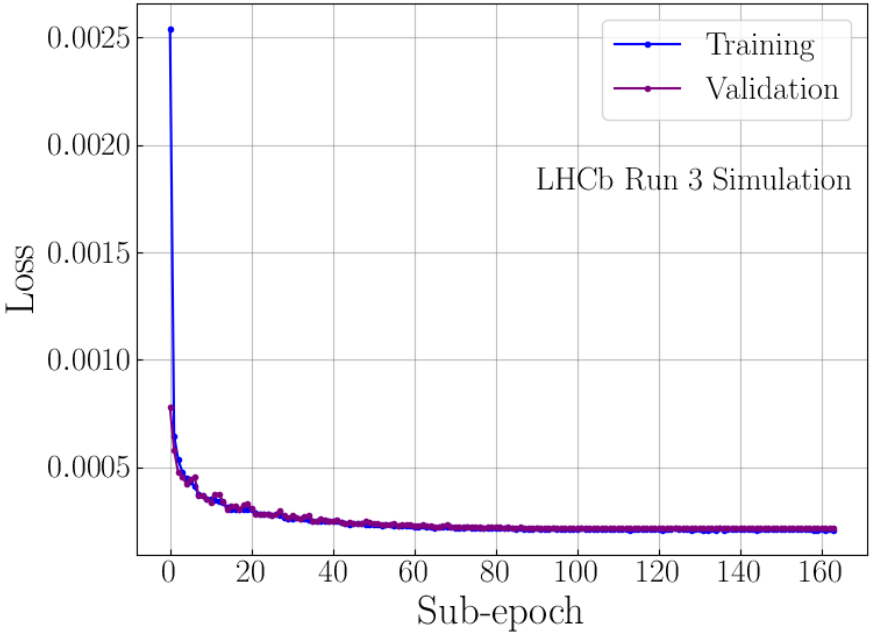
Build tracks

Overall GNN loss = GNN loss on edges + GNN loss on triplets

GNN loss on edges

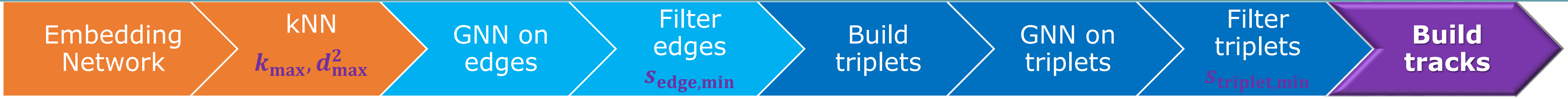


GNN loss on triplets



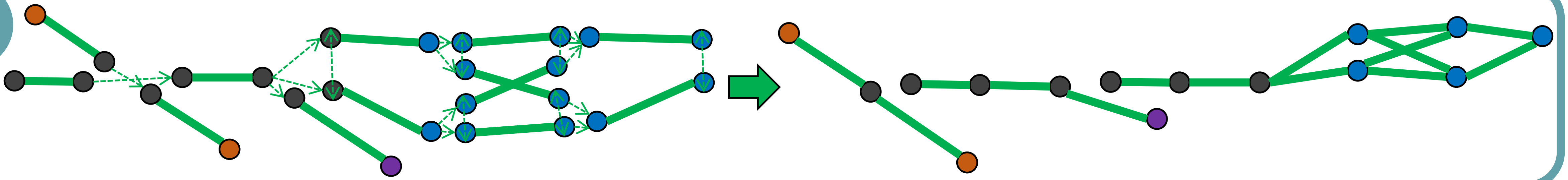
Training set of 700,000 events divided into sub-epochs of 7,000 events

2. Issue of Shared Hits

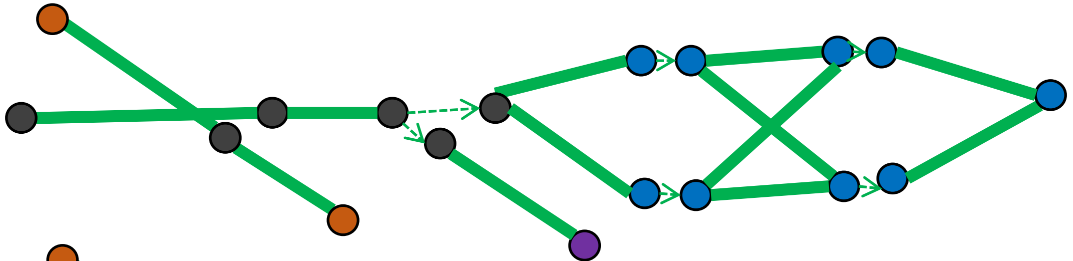


Embedding Network → kNN $k_{\max}, d^2_{\max}$ → GNN on edges → Filter edges $s_{\text{edge,min}}$ → Build triplets → GNN on triplets → Filter triplets $s_{\text{triplet,min}}$ → **Build tracks**

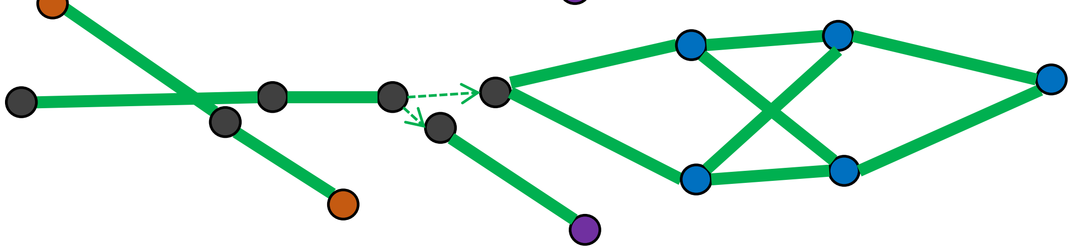
Goal



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



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



New Hypothesis: a track may split into 2 tracks **only one time**
→ Allow to keep *locality*

3. Track-Finding Performance

Matching candidate

(track, particle) couple
for which **70% of the hits of track belong to the particle**

Quality of overall track-finding

$$\textbf{Efficiency} = \frac{\#\,\text{matched particles}}{\#\,\text{particles}}$$

Proportion of matched particles

$$\textbf{Clone rate} = \frac{\#\,\text{candidates} - \#\,\text{matched particles}}{\#\,\text{candidates}} = \frac{\#\,\text{clones}}{\#\,\text{candidates}}$$

Proportion of redundant candidates

$$\textbf{Ghost rate} = \frac{\#\,\text{unmatched tracks}}{\#\,\text{tracks}}$$

Proportion of unmatched tracks

Quality of individual tracks

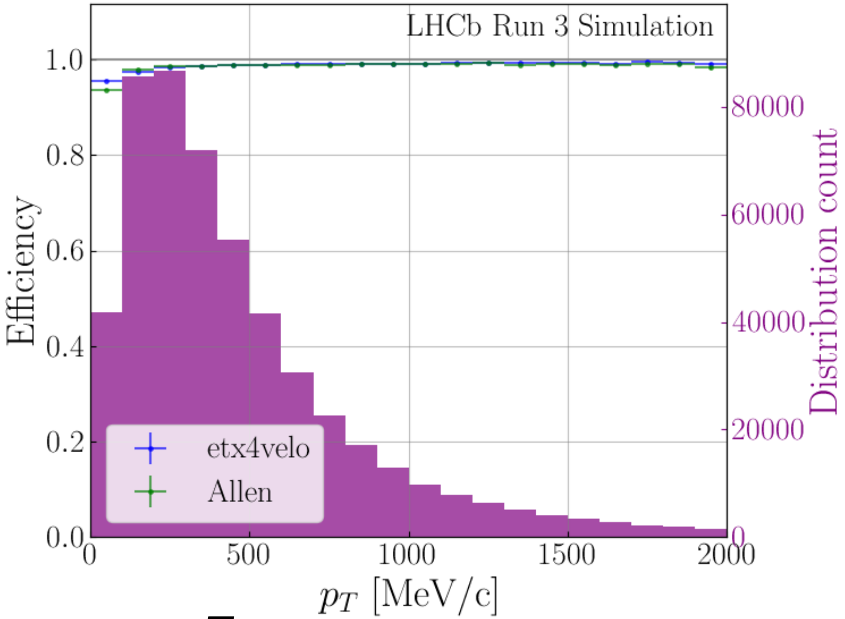
$$\textbf{Hit Efficiency} = \left\langle \frac{\#\,\text{matched hits on track}}{\#\,\text{hits on particle}} \right\rangle_{\text{candidates}}$$

Average proportion of matched hits on particle

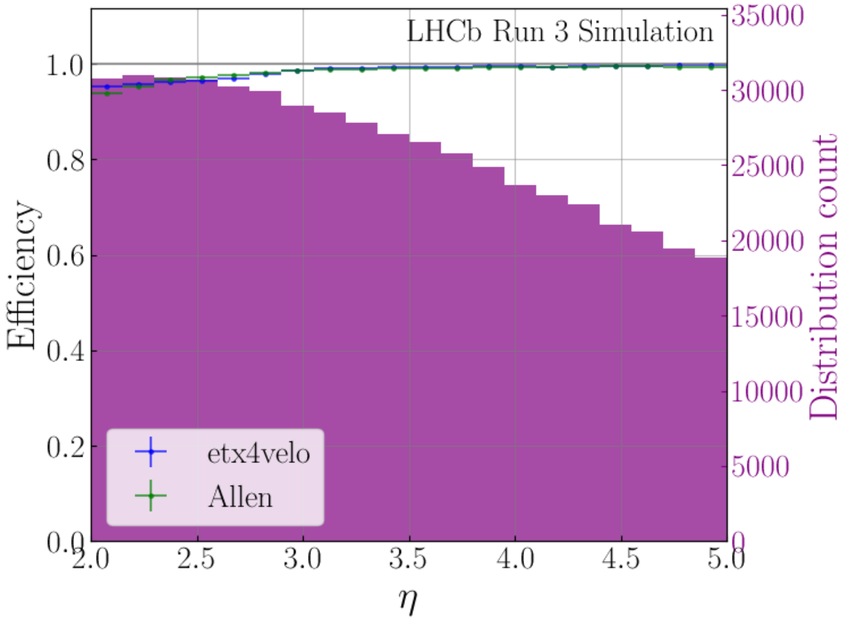
$$\textbf{Hit Purity} = \left\langle \frac{\#\,\text{matched hits on track}}{\#\,\text{hits on track}} \right\rangle_{\text{candidates}}$$

Average proportion of matched hits on track

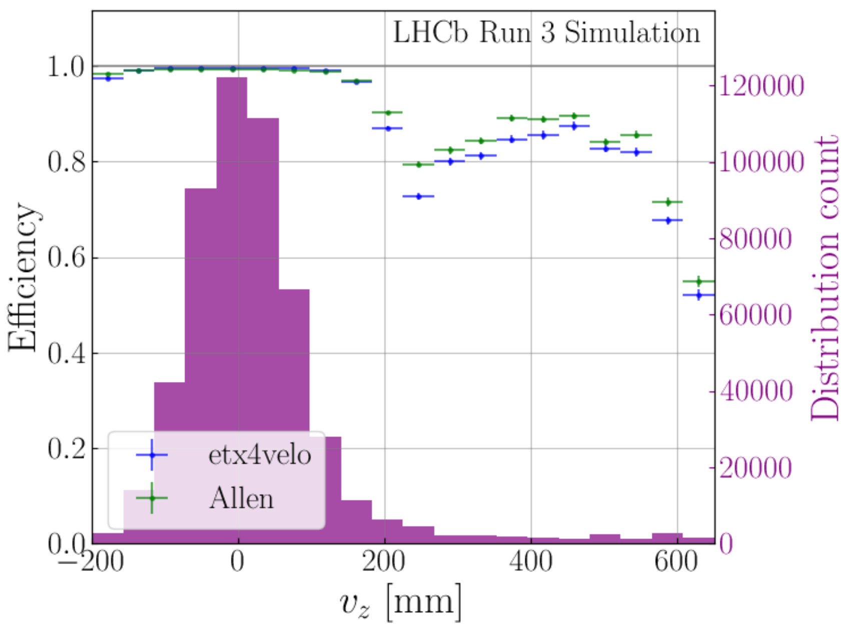
3. Track-Finding Performance



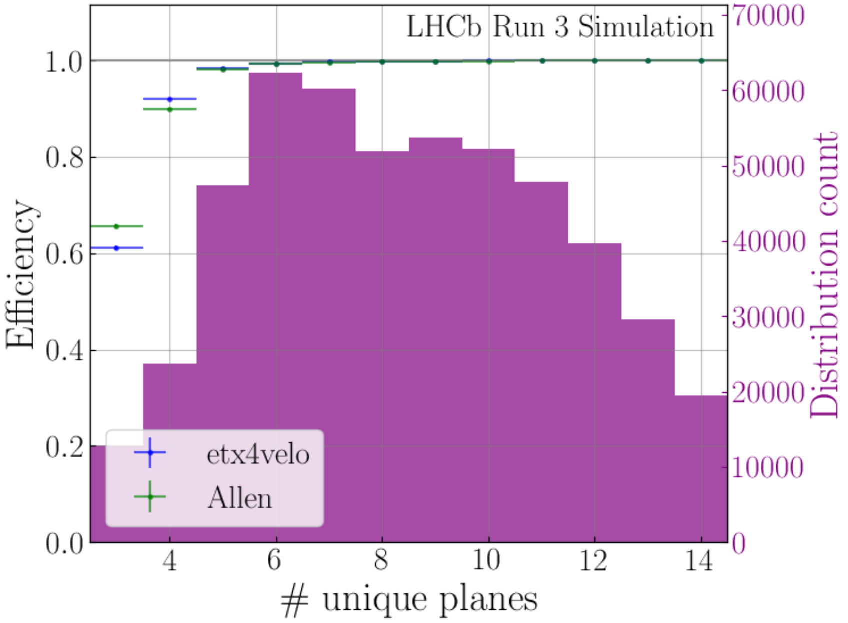
Transverse momentum



Pseudo-rapidity



Vertex origin position along the z-axis



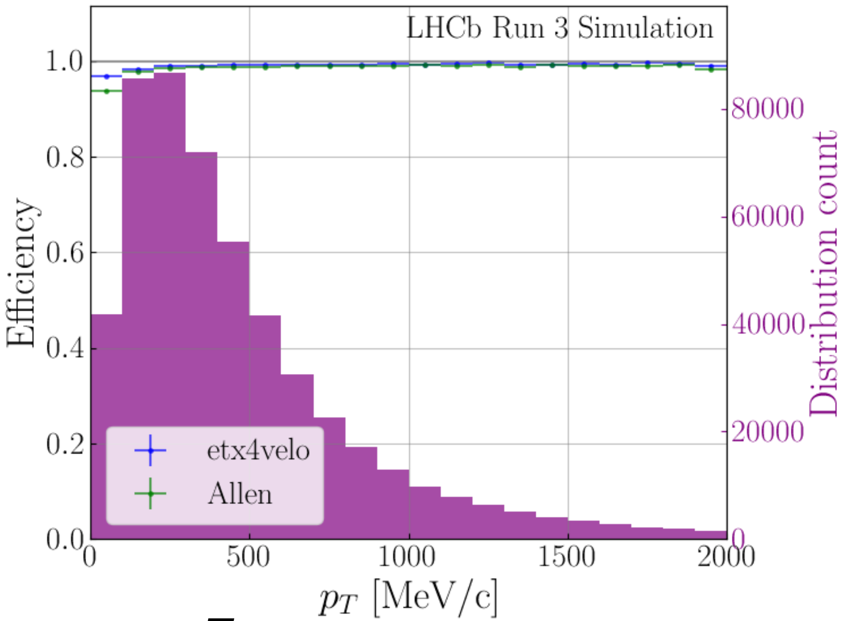
Velo,
no electrons

$\boldsymbol{d^2_{\max} = 0.010}$

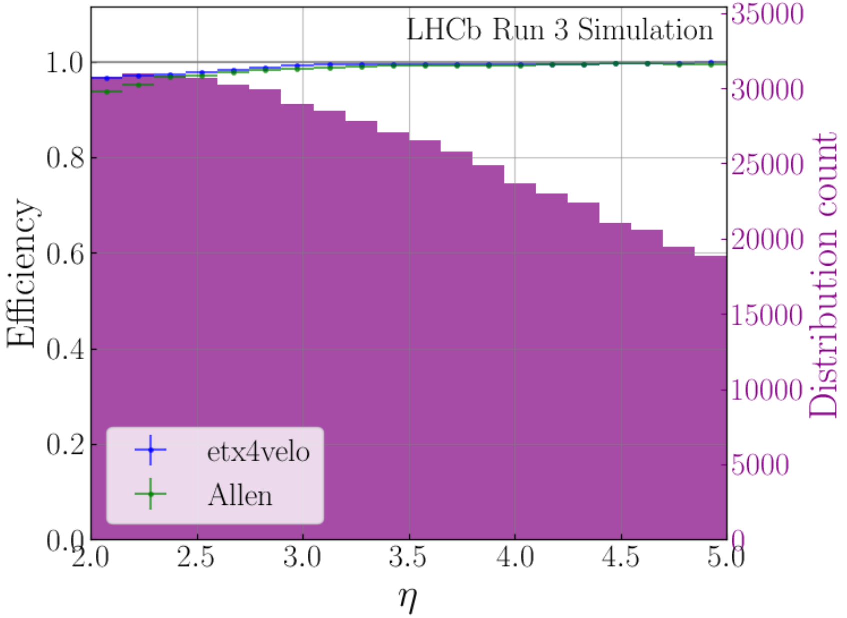
Lower efficiency at

- Larger v_z
- Smaller # unique planes

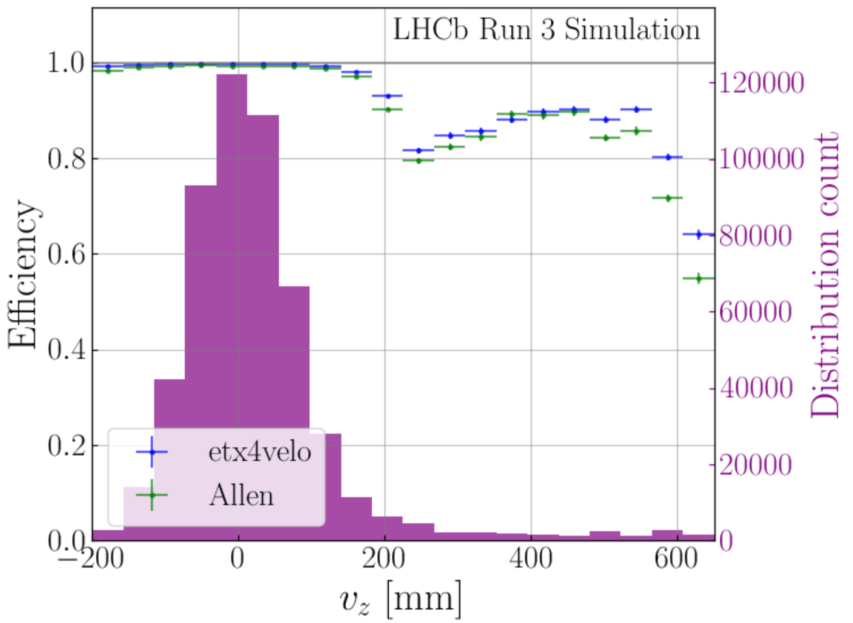
3. Track-Finding Performance



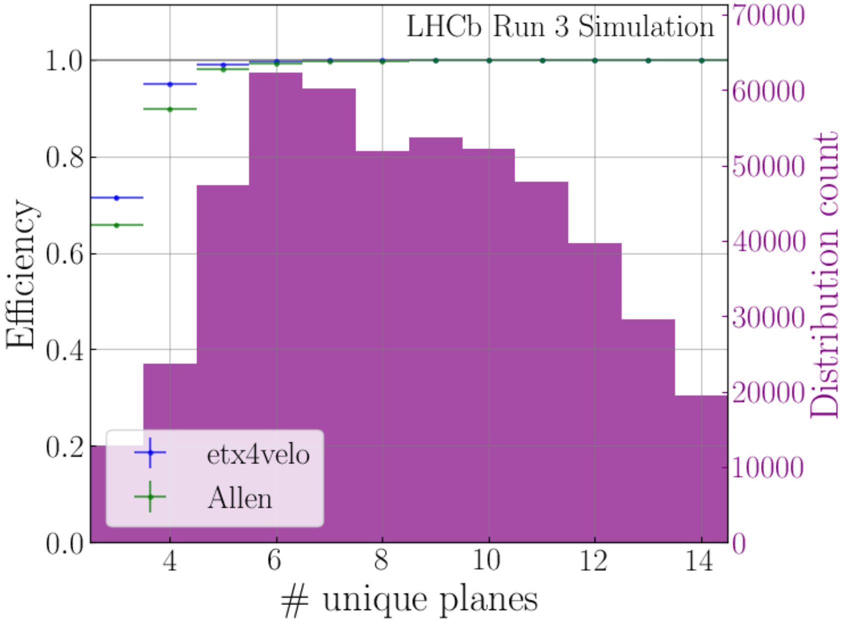
Transverse momentum



Pseudo-rapidity



Vertex origin position along the z-axis

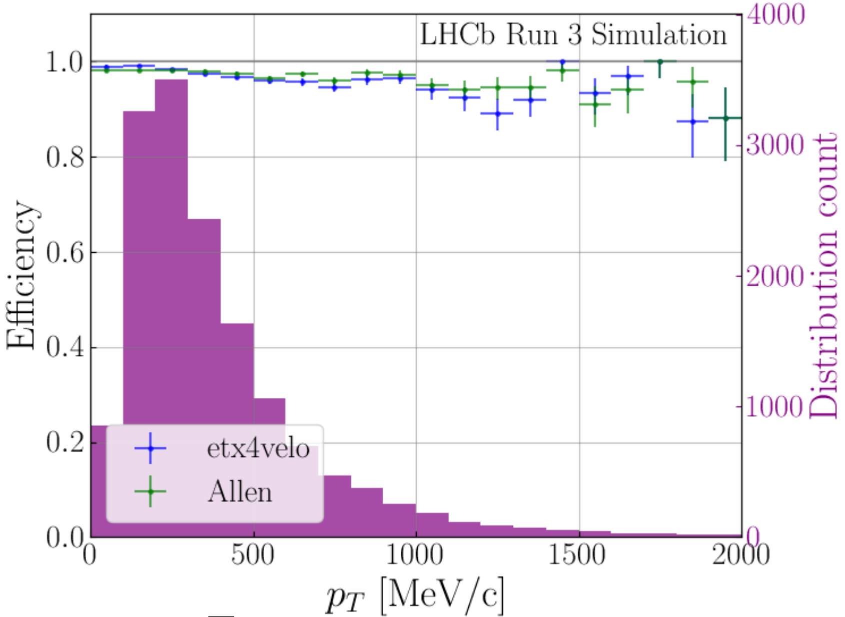


Velo,
no electrons

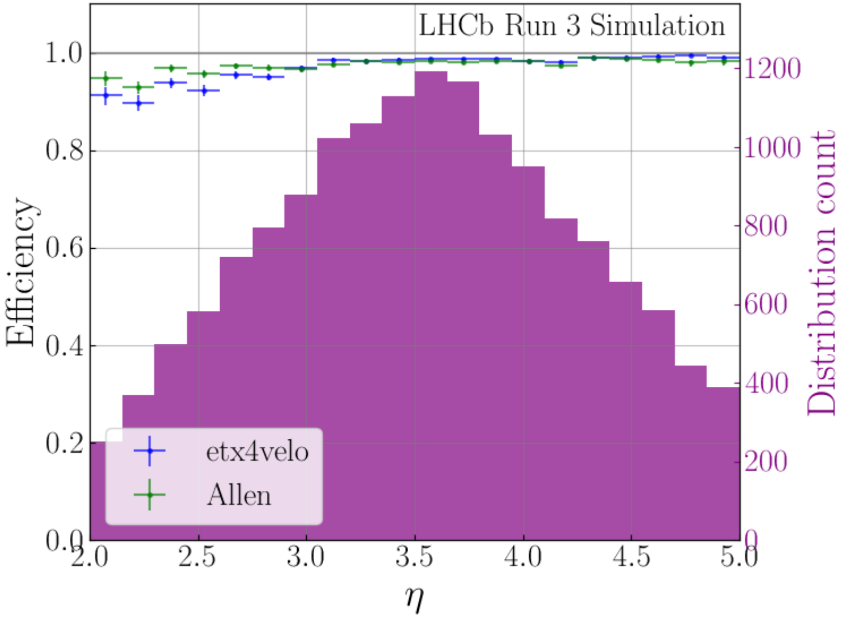
$d^2_{\max} = 0.020$

Better efficiencies everywhere

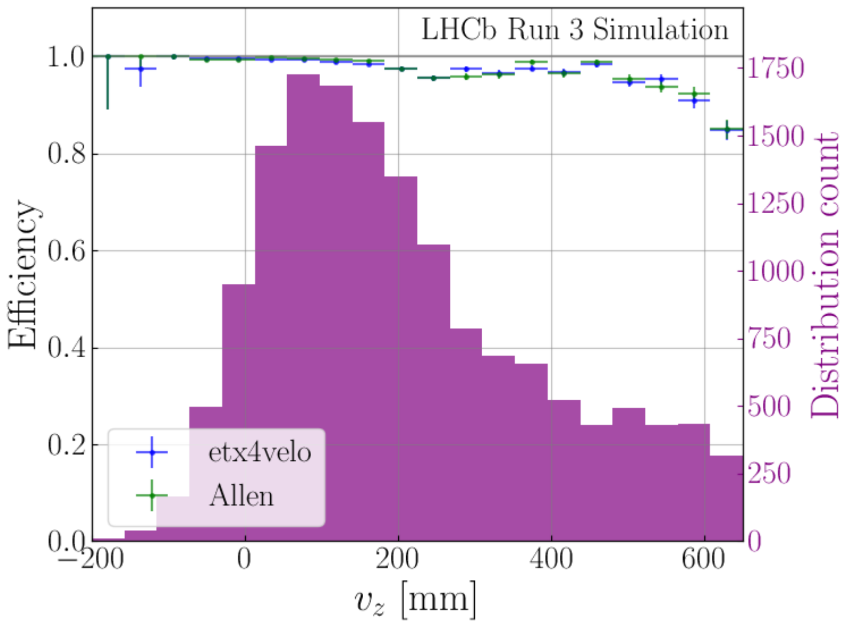
3. Track-Finding Performance



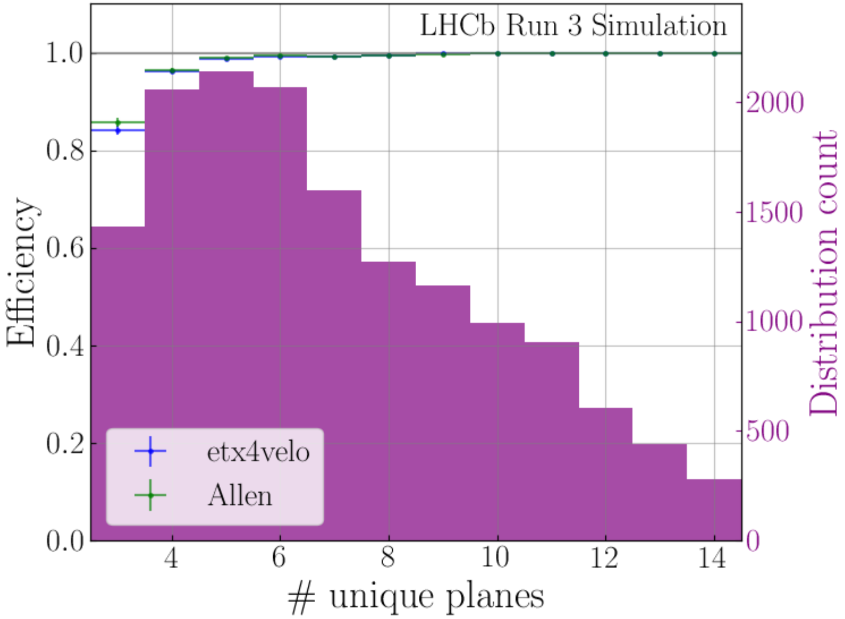
Transverse momentum



Pseudo-rapidity



Vertex origin position along the z-axis



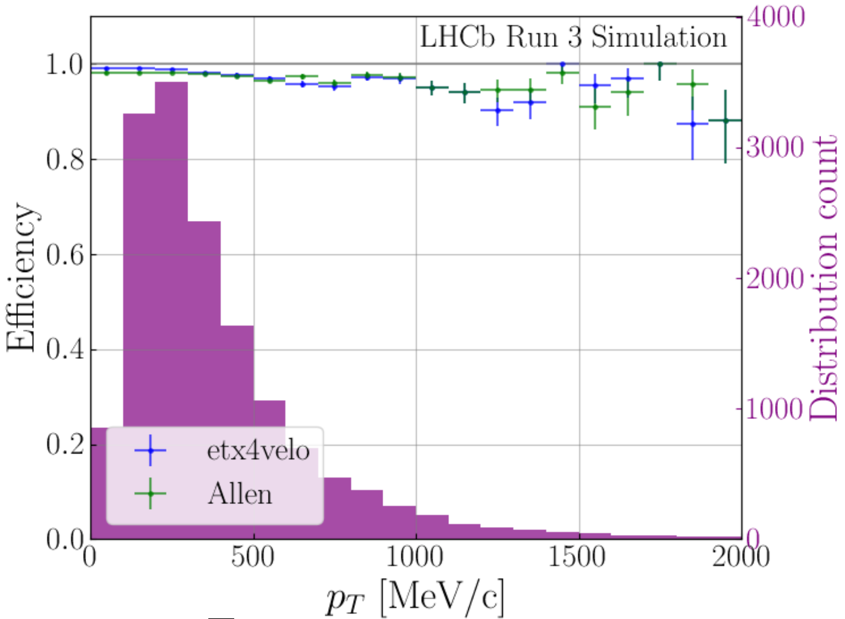
Long, from strange

$d^2_{\max} = 0.010$

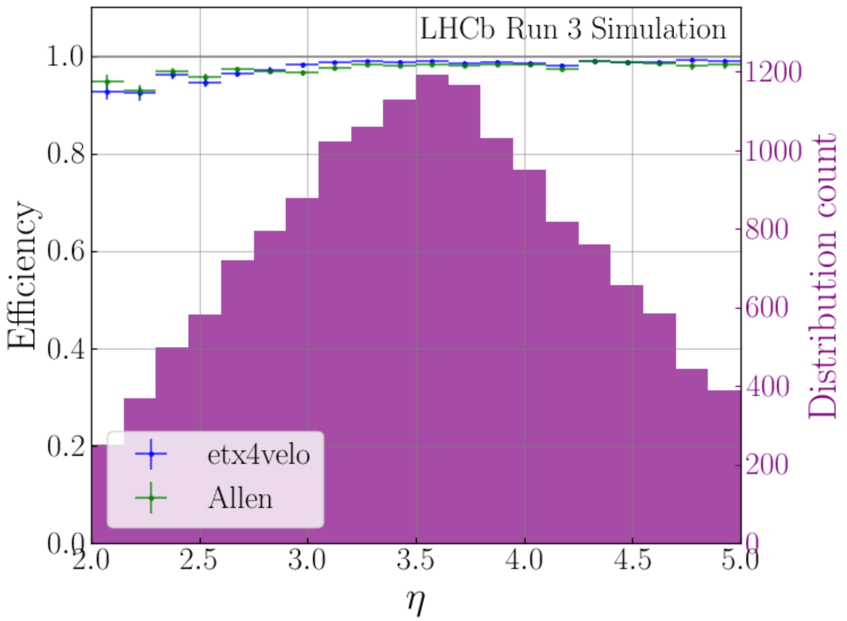
Lower efficiency at

- smaller η
- Smaller # unique planes

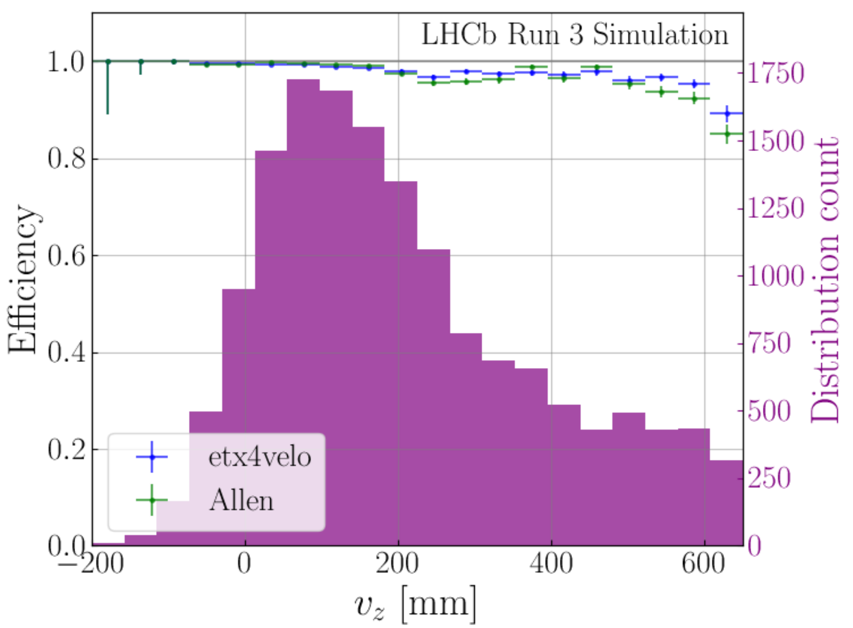
3. Track-Finding Performance



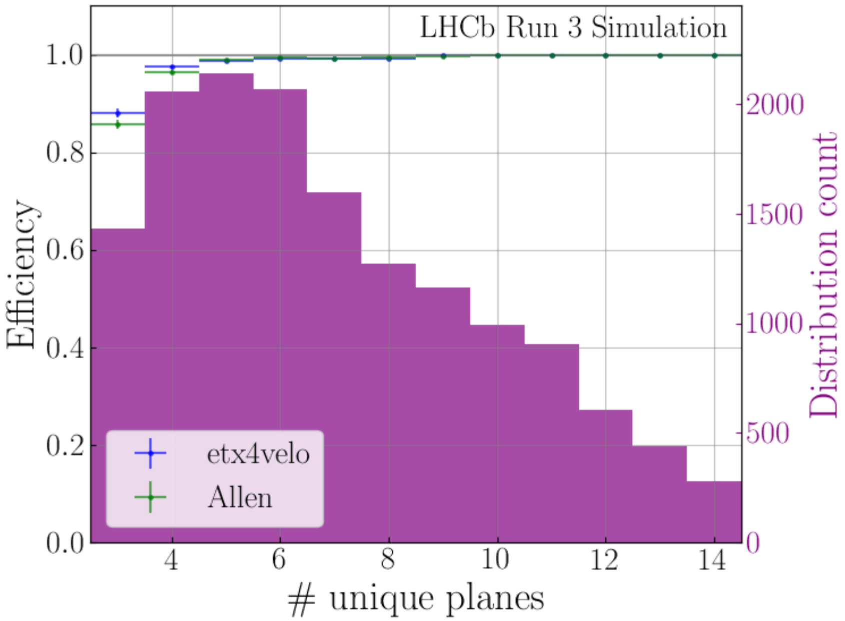
Transverse momentum



Pseudo-rapidity



Vertex origin position along the z-axis



Long, from strange

$d^2_{\max} = 0.020$

Better efficiencies everywhere