New techniques in data analysis

NDIVE: a differential vertexing algorithm

Rachel Smith, Inês Ochoa, Ruben Inácio,
Jonathan Shoemaker, Michael Kagan

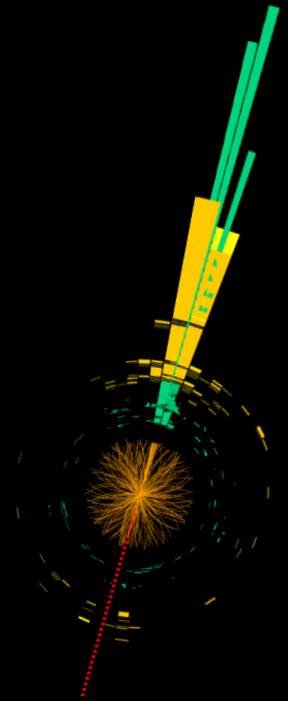
IDTM Workshop

September 12-14, 2023

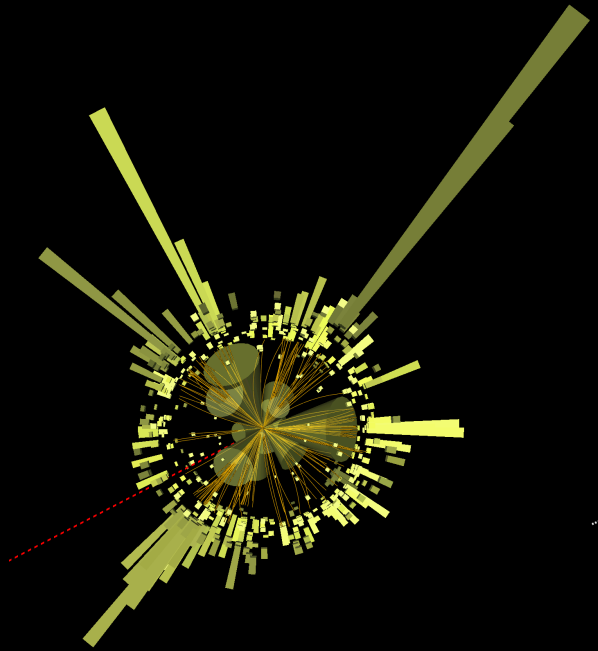
Outline

- Jets and b-tagging
  - A brief physics motivation
- Overview of b-tagging algorithms in ATLAS (current state-of-the-art)
  - Where vertexing comes in
- NDIVE: Differential vertexing and its application to b-tagging
- Future prospects

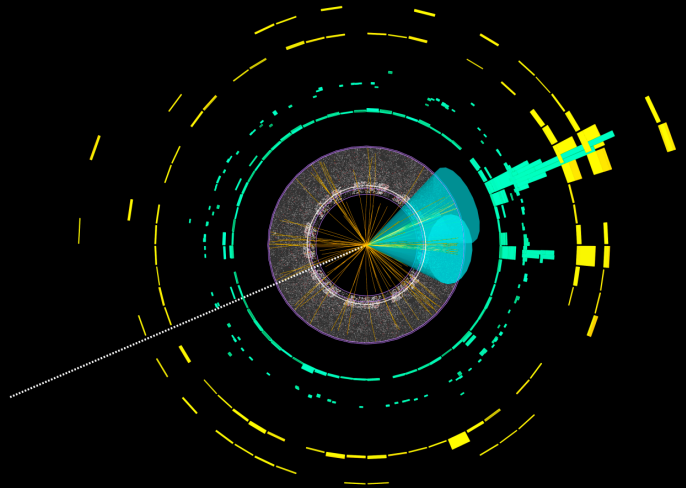
Jets in hadron colliders come in different multiplicities, sizes, flavours…



Mono-jet event



High-multiplicity jet event

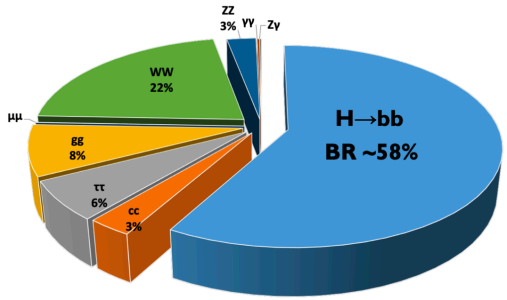


b-jets from a Higgs boson decay

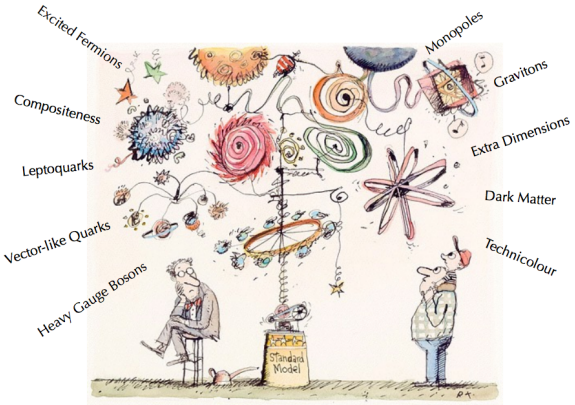
Physics motivation

- Identifying **b-jets** enable us to select interesting physics signatures from Standard Model processes and beyond…

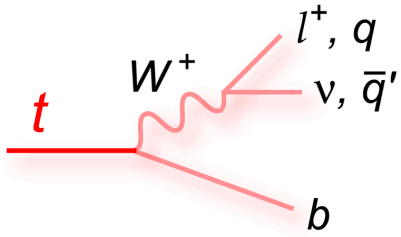
Higgs boson decay modes



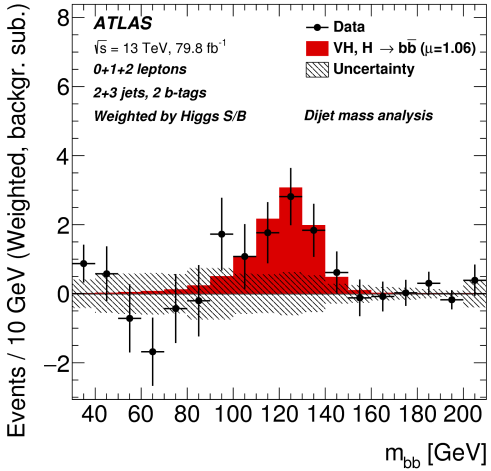
SUSY, exotic particles, …



Top quark decay



VH, H→bb

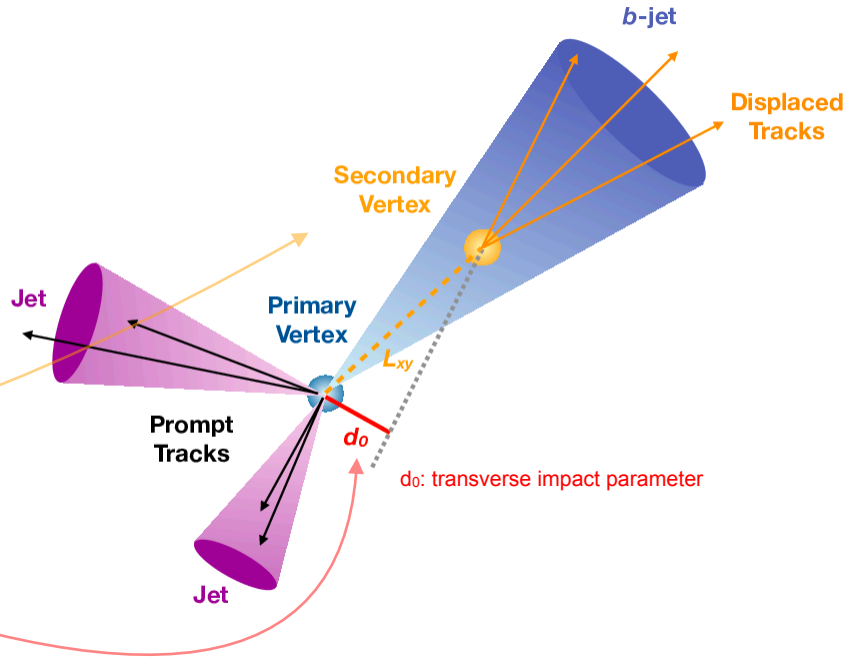


b-quarks → b-hadrons → b-jets

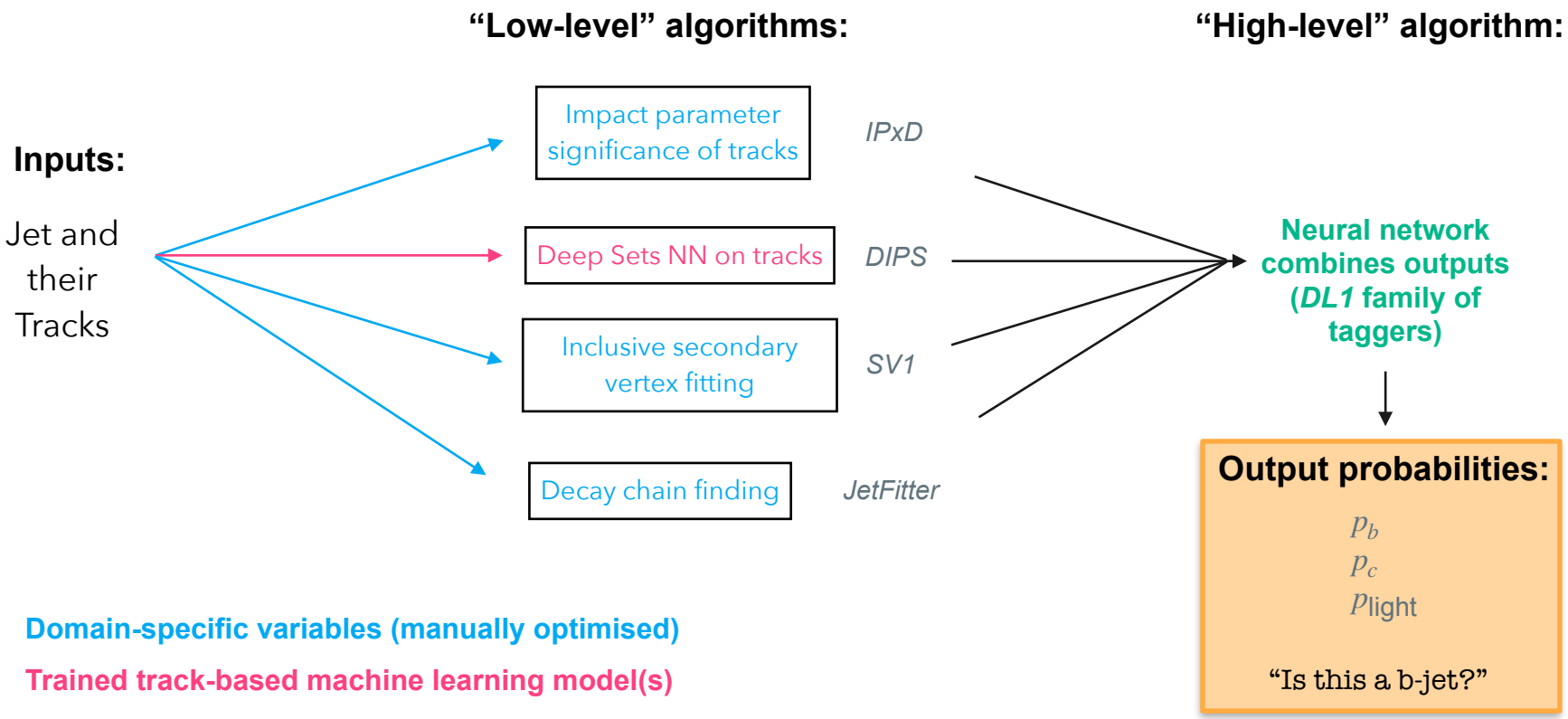
fragment into which decay and create

- **b-jets** contain the decay particles of ***long-lived*** b-hadrons and some additional particles.

- This leads to **unique characteristics** that distinguish them from **light** (u,d,s,g) and to a lesser extent charm (c) jets:
  - A secondary vertex
  - Tracks with large impact parameters
  - Leptons from the b-hadron decay



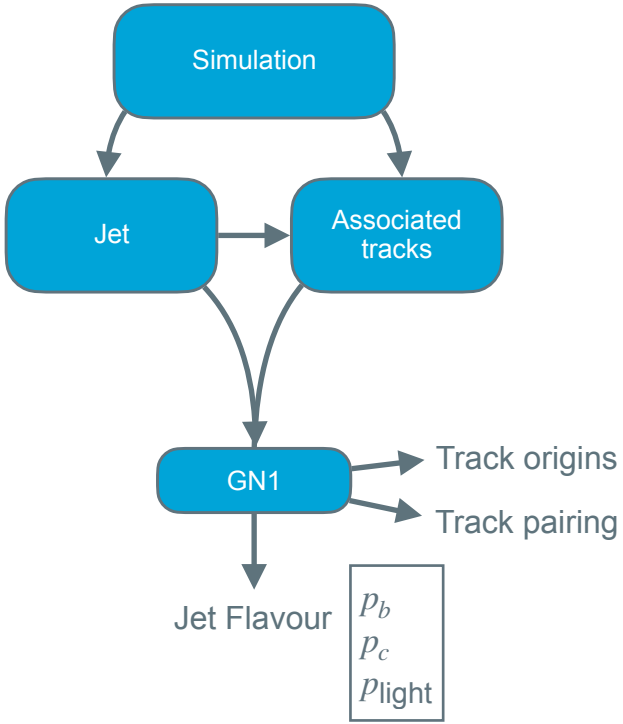
Overview of b-tagging algorithms in ATLAS (I)



Overview of b-tagging algorithms in ATLAS (II)

- New state-of-the-art: *GN1* & *GN2*
  - Single end-to-end neural network (graph or transformer) with auxiliary training objectives to learn jet origin.

- No usage of intermediate low-level algorithms!
- No explicit secondary (or tertiary) vertex reconstruction!
  - i.e. no *vertex fitting*



CERN Data Science Seminar by Sam Van Stroud

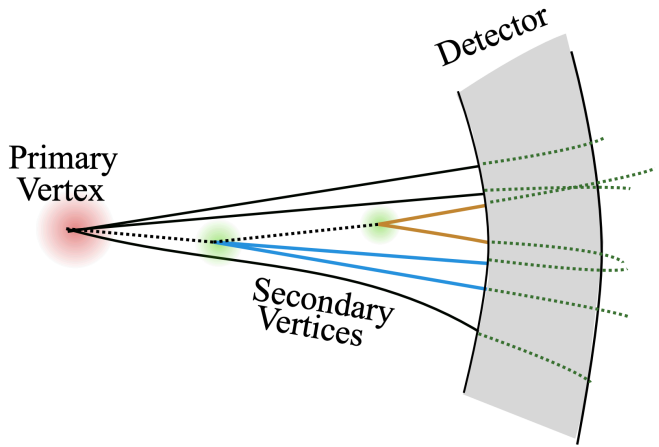
Vertex finding & vertex fitting

- **Vertex finding:** grouping tracks that originate at the same point in space

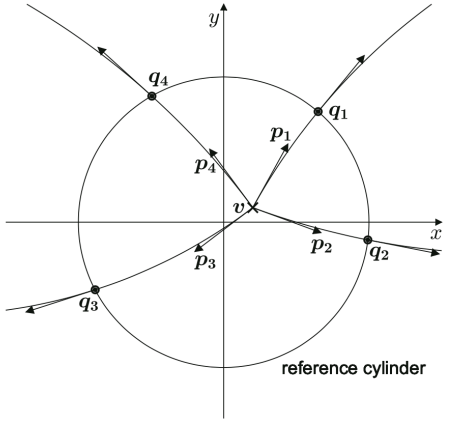
- **Vertex fitting:** given a set of N tracks and their track parameters $\mathbf{q}_i$ and associated covariance matrices $\mathbf{V}_i$, estimate the the vertex position $\mathbf{v}$ and the momentum vectors $\mathbf{p}_i$ of all tracks at the vertex.

Can we integrate vertex fitting into an ML end-to-end trainable algorithm?

Schlomi J., et al. Secondary Vertex Finding in Jets with Neural Networks

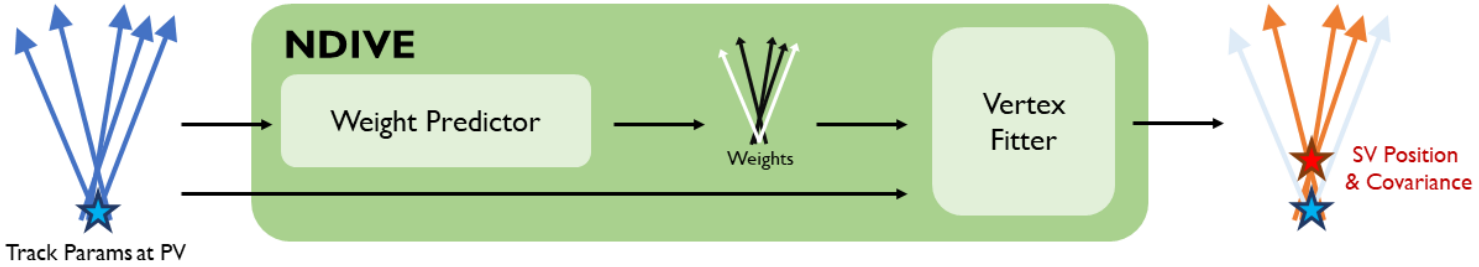


Frühwirth, Strandlie in Pattern Recognition, Tracking and Vertex Reconstruction in Particle Detectors



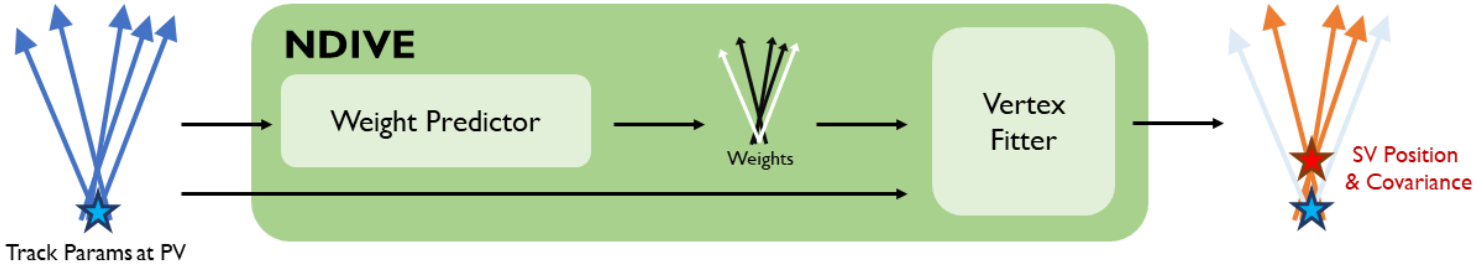
NDIVE: Neural Differentiable Vertexing layer

- We propose to explicitly reintroduce vertex reconstruction into end-to-end ML b-tagging algorithms via a vertexing layer that performs both vertex finding and vertex fitting.



NDIVE: Neural Differentiable Vertexing layer

- We propose to explicitly reintroduce vertex reconstruction into end-to-end ML b-tagging algorithms via a vertexing layer that performs both **vertex finding** and **vertex fitting**.



Neural network trained to assign weights to tracks per vertex

Vertex fitting algorithm with tracks and weights as inputs and no trainable parameters

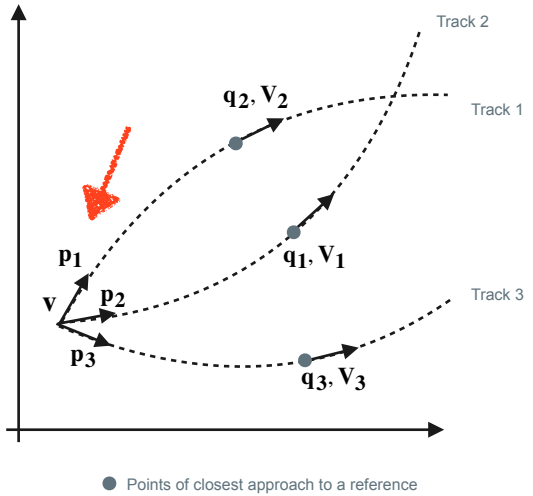
Loss function = mean euclidean distance (true, pred)

Inclusive Vertex Fit formulation (I)

- We perform an *inclusive vertex fit with per-track weights*, following closely Billoir's algorithm, using a local parameterisation of tracks around a reference point.

  - The goal is to find a common point of production for a set of tracks.
  - The values to be optimised are the vertex position $\mathbf{v}$ and track momentum at the vertex $\{\mathbf{p}_i\}$:

$$\mathbf{x} = (\mathbf{v}, \{\mathbf{p}_i\})$$

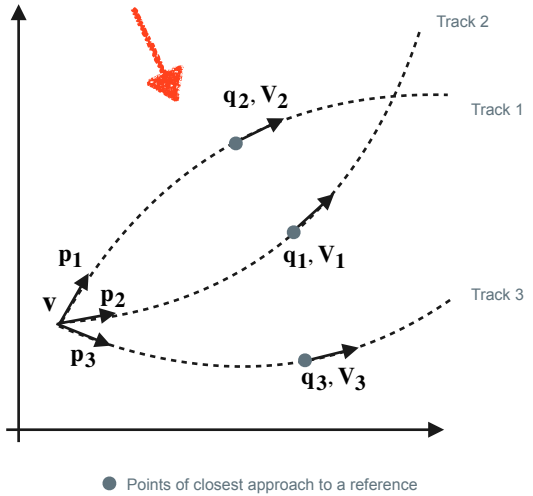


Inclusive Vertex Fit formulation (II)

- We perform an *inclusive vertex fit with per-track weights*, following closely Billoir's algorithm, using a local parameterisation of tracks around a reference point.

- The input data are the measured track parameters $q_i = (d_0, z_0, \phi, \theta, \rho)$ and their covariance matrix V_i.
  - *Perigee* representation.
- Additionally, a set of per-track weights w_i which determine how much each track contributes to the vertex fit.

d_0 : signed transverse impact parameter
z_0 : longitudinal impact parameter
ϕ : polar angle of trajectory
θ : azimuthal angle of trajectory
ρ : signed curvature



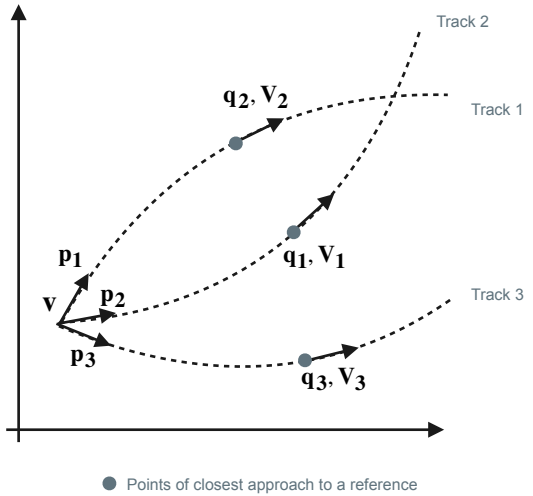
Inclusive Vertex Fit formulation (III)

- We perform an *inclusive vertex fit with per-track weights*, following closely Billoir's algorithm, using a local parameterisation of tracks around a reference point.

- The following objective function $\mathcal{S}$ is minimised:

$$\mathcal{S} = \chi^2 = \sum_{i=1}^{N} = w_i(\mathbf{q}_i - \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i))^T V_i^{-1}(\mathbf{q}_i - \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i))$$

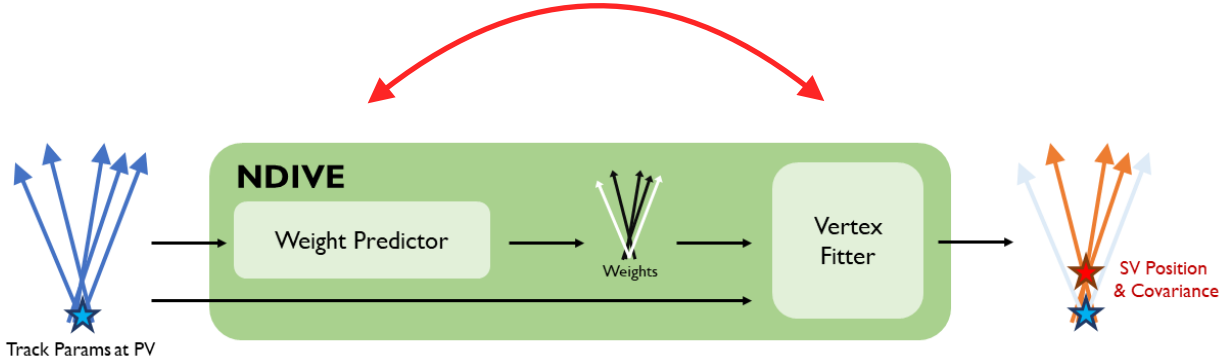
Track and fit parameters are related via a track model $\mathbf{q}_{\text{model},i} = \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i)$ (e.g. a helical model of a curved track)



The derivatives of a fit vertex $\mathbf{v}$ with respect to the weights w_i are needed to train any downstream or upstream neural networks!

Inclusive Vertex Fit formulation (IV)

Once a vertex solution is found, the gradient of the solution vertex with respect to input weights and particle features is defined using implicit differentiation.



Enabling differentiation through the optimisation of the vertex fit:

1. Forward pass with iterative numerical algorithm to perform optimisation.
2. Backward pass done with a custom derivative.

PyTorch

Optimisation as a layer (implicit differentiation)

- Specify the conditions we want the layer's output to satisfy:

$$\hat{\mathbf{x}}(\alpha) = \arg\min_{x} \mathcal{S}(\mathbf{x}, \alpha)$$

$\mathbf{x} = (\text{vertex}, \{\mathbf{p}_i\})$

$\alpha = (\text{weights}, \text{tracks}, \text{cov})$

- We need the derivative of a fit vertex ($\mathbf{x}$) with respect to the parameters α to train the upstream neural network.
- Note that at the minimum of $\mathcal{S}$ we have:

$$\frac{\partial \mathcal{S}(\mathbf{x}, \alpha)}{\partial \mathbf{x}} = 0$$

$\hat{\mathcal{G}} = \partial_x \mathcal{S}(\hat{\mathbf{x}}, \alpha)$

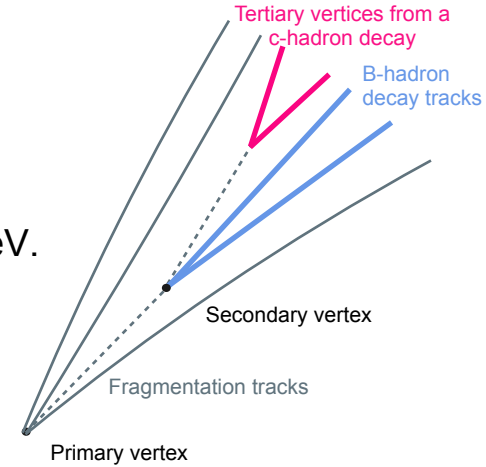
- Taking the derivative wrt α and accounting for the implicit dependence of $\hat{\mathbf{x}}$ on α:

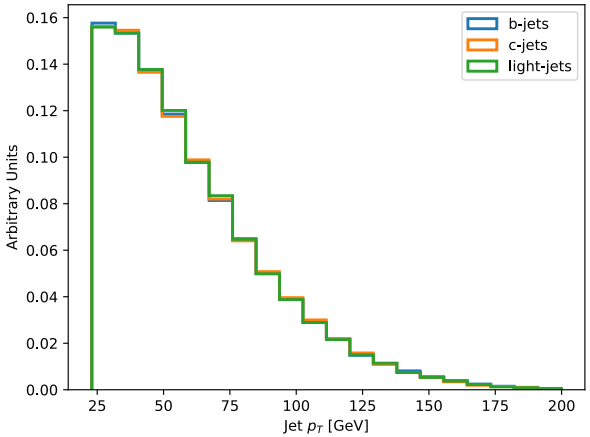
$$0 = \frac{d}{d\alpha}\hat{\mathcal{G}} = \frac{\partial \hat{\mathcal{G}}}{\partial \alpha} + \frac{\partial \hat{\mathcal{G}}}{\partial \mathbf{x}}\frac{\partial \mathbf{x}}{\partial \alpha} \Rightarrow \boxed{\frac{\partial \mathbf{x}}{\partial \alpha} = -\left(\frac{\partial \hat{\mathcal{G}}}{\partial \mathbf{x}}\right)^{-1}\frac{\partial \hat{\mathcal{G}}}{\partial \alpha}}$$

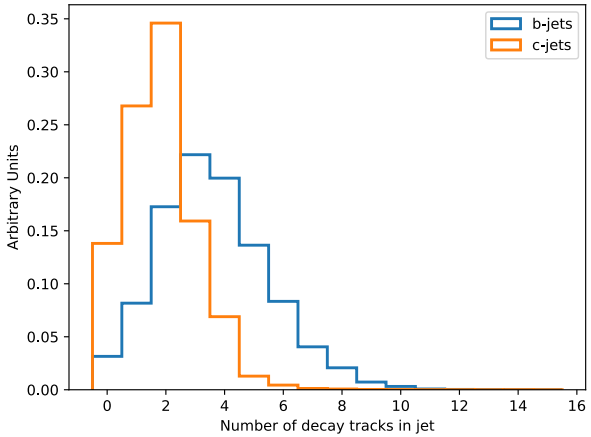
Derivatives of optimisation solution wrt objective parameters needed for training.

Dataset & Inputs

- Top-pair production from proton-proton collisions simulated at $\sqrt{s} = 14$ TeV.
  - Generated with Pythia8 with ATLAS detector parameterisation via Delphes.







Training features:

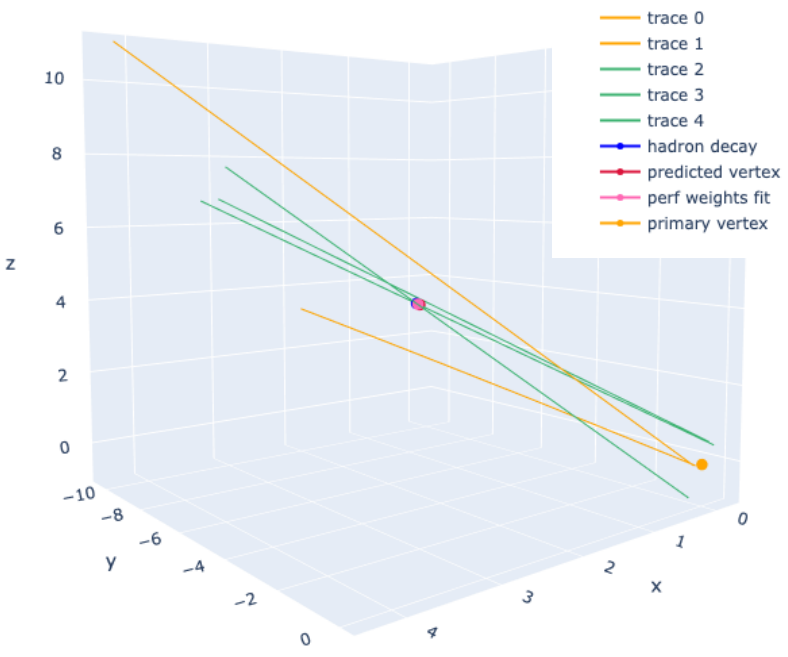
- Track perigee parameters and their errors
- Signed d_0 and z_0 significances
- log(track p_T / jet p_T)
- ΔR (track, jet)

[Zenodo: Secondary Vertex Finding in Jets Dataset]

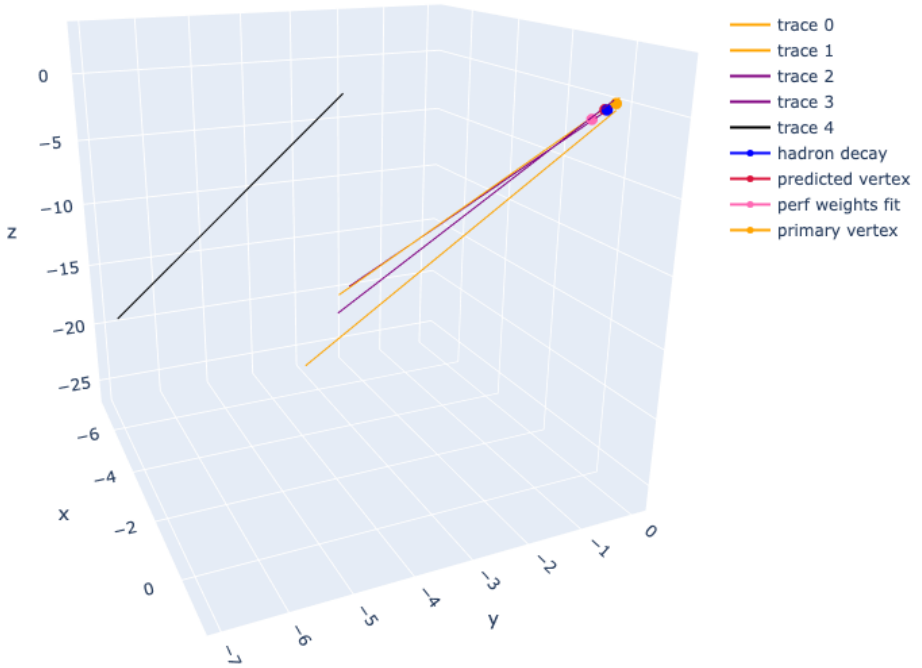
Good tracks → Good vertices

Track origin color code:
From B-hadron
From C-hadron
From primary vertex
Other

b-jet



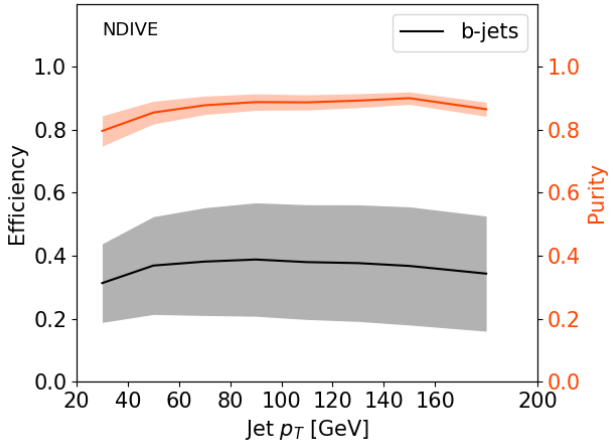
c-jet

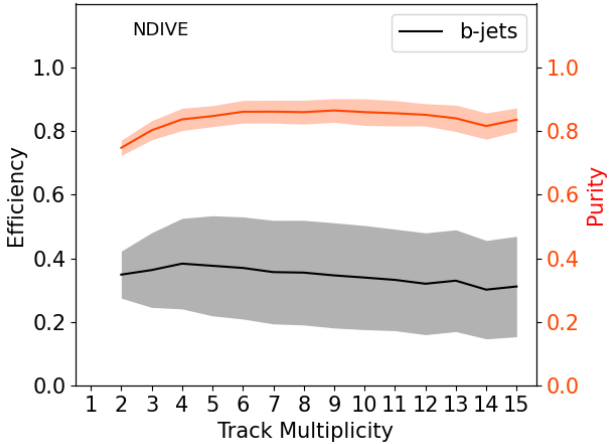


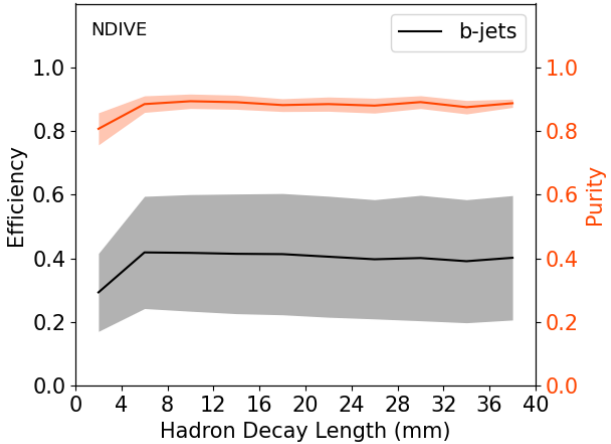
Track selection performance

Work in progress!

- Efficiency: number of decay tracks selected over all decay tracks
- Purity: number of decay tracks selected over all selected tracks



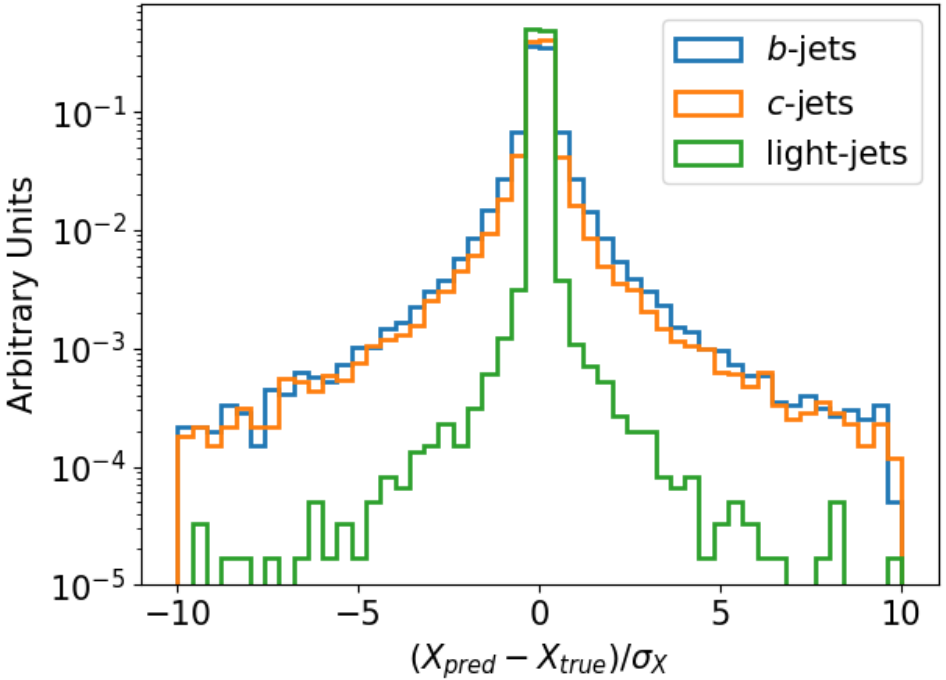


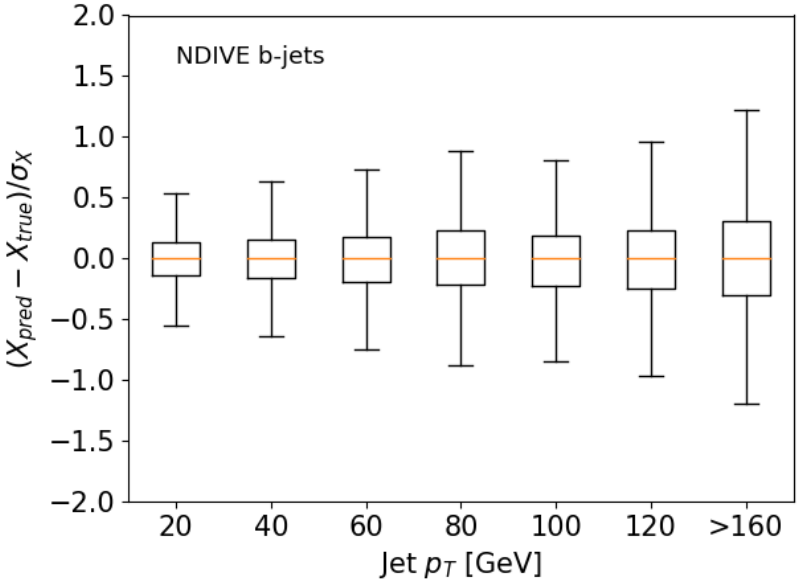


- "Selected tracks": per-track weights normalised by maximum weight in each jet and required to be above > 0.5

Vertex reconstruction performance

Work in progress!

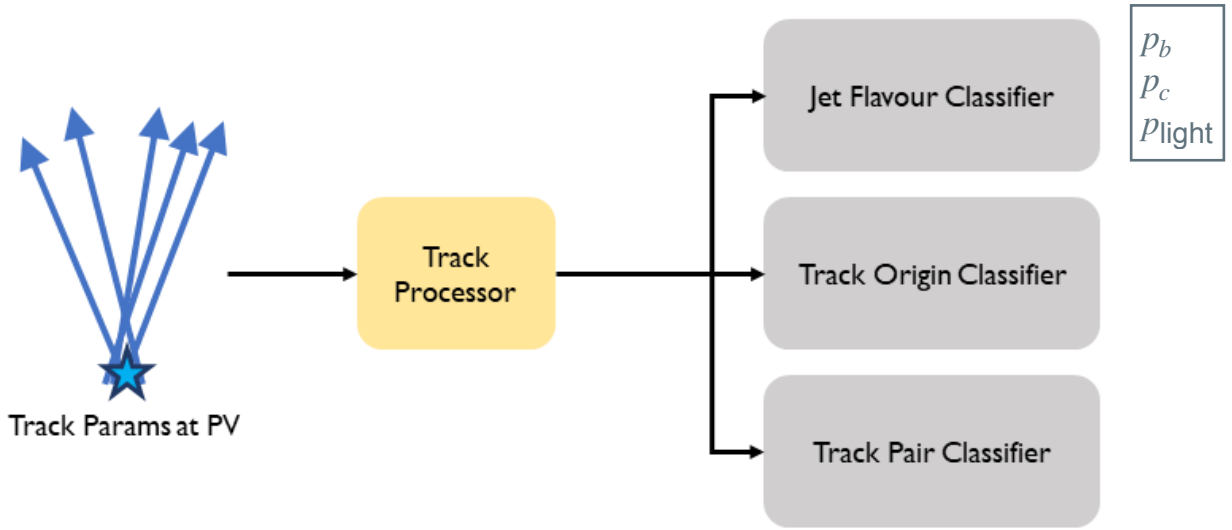




Unbiased and highly peaked at 0

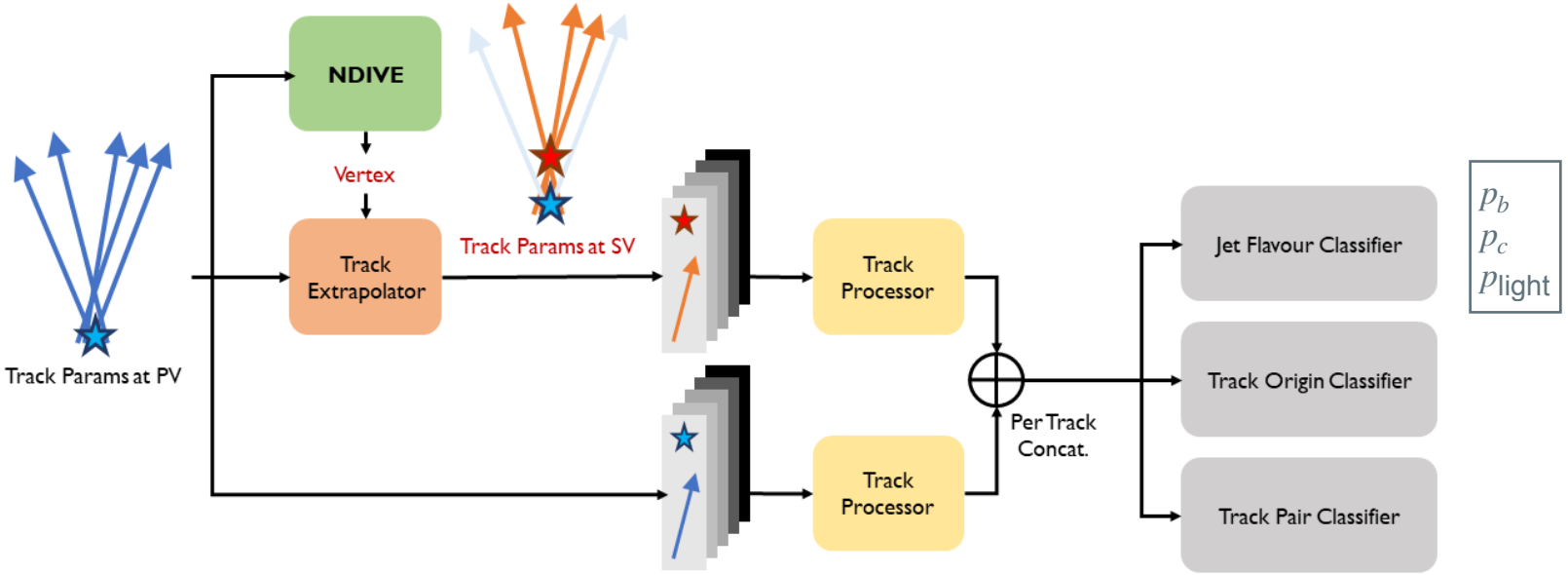
Integration in a flavour-tagging model

FTAG (Baseline)



Integration in a flavour-tagging model

FTAG+NDIVE

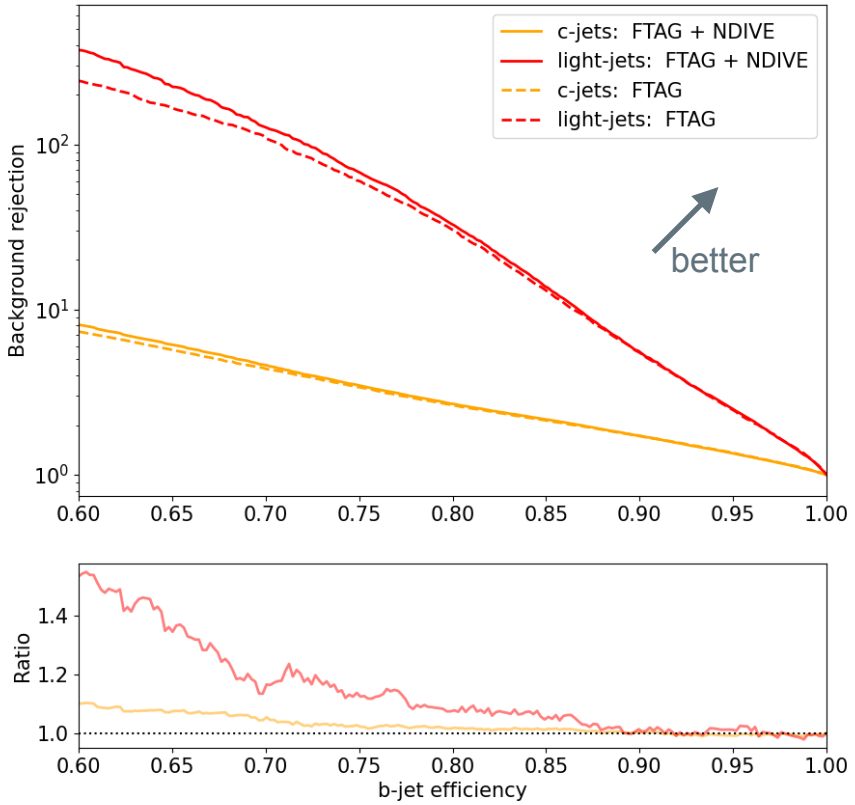


Model comparison: ROC curve

Work in progress!

$$D_b = \log \frac{p_b}{(1 - f_c)p_l + f_c p_c}$$

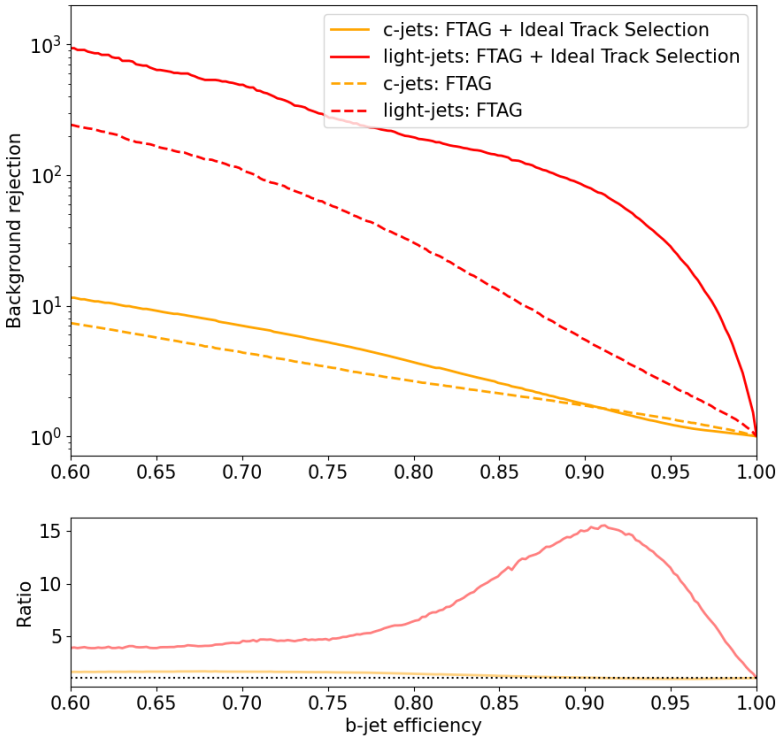
$f_c = 0.05$



Future prospects

Work in progress!

- These methodological developments are generic, applicable to other vertex fitting algorithms and other schemes for integrating vertex information into neural networks.
- To illustrate possible improvements, we show the potential for huge rejection rate gains in an ideal scenario with perfect track selection.



In summary…

- We introduce NDIVE: a neural differentiable vertexing layer
  - First differentiable vertex fitting algorithm.
  - Paper coming soon!

- Vertex fitting formulated as an optimisation problem:
  - Gradients of optimised solution vertex defined through implicit differentiation.
  - Can be passed to upstream or downstream NN components for training.

- Application of *differential programming* for integrating physics knowledge into HEP neural networks:
  - NDIVE can be integrated into future end-to-end b-tagging algorithms, explicitly reintroducing vertex geometry.
  - Part of wider application of differentiable programming to HEP!

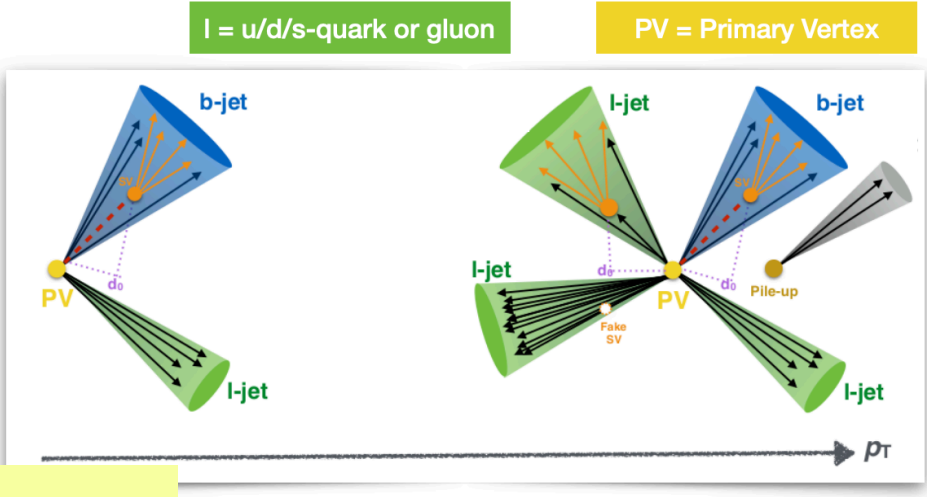
Backup

Features of b-jets

The relatively long lifetime of B-hadrons (~1.5ps) can allow for significant displacement before decay

Typical b-jet signatures:

- Hard fragmentation
- Displaced secondary vertices (high mass)
- Displaced tertiary vertices from $B \rightarrow C$
- Large track impact parameters (d_0)
- Missing hits on inner detector layers
- Semileptonic decays



These characteristics result in specific signatures we can look for to identify b-jets.

Inclusive SV algorithm (“SV1”)

- Finding strategy:
  - Find all displaced 2-track vertices within the jet (χ^2)
  - Remove all vertices with di-track mass compatible with KS, Lambda decay or photon conversion
  - Remove all vertices in correspondence of pixel layers (likely from material interactions)
- Using only tracks from any of the non-vetoed 2-track vertices, form a single inclusive secondary vertex.
- Combine variables following variables into a 3D likelihood function:
  - Invariant mass at vertex
  - Number of non-vetoed 2-track vertices
  - Energy fraction of tracks at vertex wrt all tracks in jet

Track parameterisation

Tracks described by five parameters and a reference point (typically the origin), using a *perigee* representation:

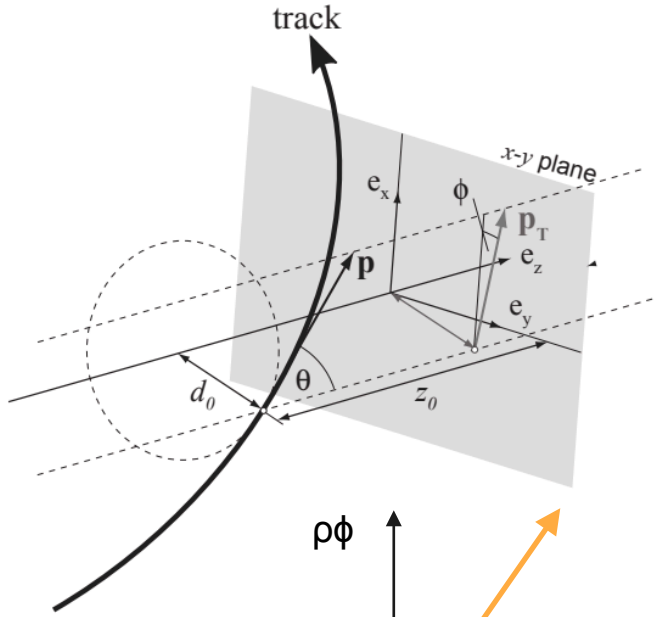
d_0 : signed transverse impact parameter

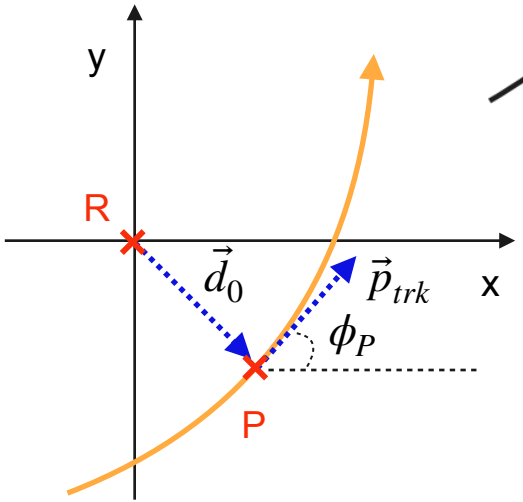
z_0 : longitudinal impact parameter

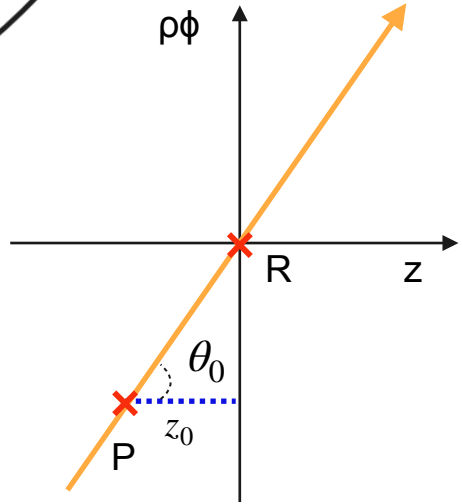
ϕ : polar angle of trajectory

θ : azimuthal angle of trajectory

ρ : signed curvature







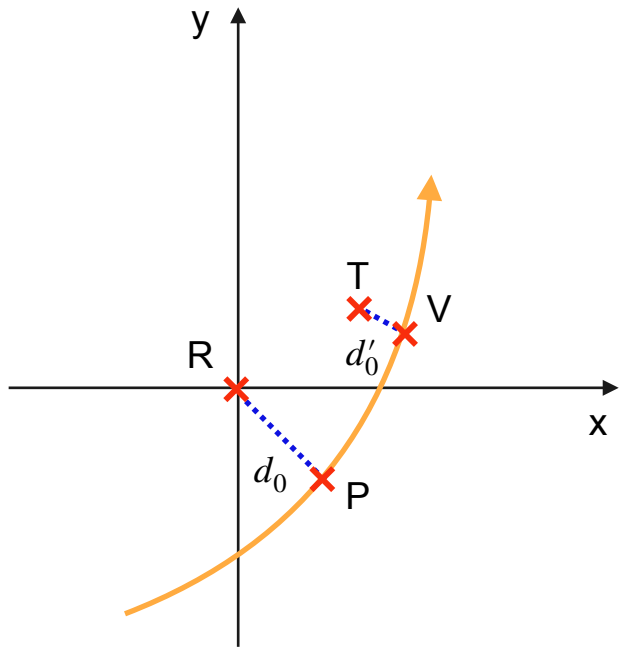
Track extrapolation

Generic position V along the track trajectory parameterised by considering the track's perigee representation wrt a reference R:

$$x_V = x_P + d_0 \cos\left(\phi + \frac{\pi}{2}\right) + \rho\left[\cos\left(\phi_V + \frac{\pi}{2}\right) - \cos\left(\phi + \frac{\pi}{2}\right)\right]$$

$$y_V = y_P + d_0 \sin\left(\phi + \frac{\pi}{2}\right) + \rho\left[\sin\left(\phi_V + \frac{\pi}{2}\right) - \sin\left(\phi + \frac{\pi}{2}\right)\right]$$

$$z_V = z_P + z_0 - \frac{\rho}{\tan(\theta)}\left[\phi_V - \phi\right]$$



Deep Implicit Layers (I)

http://implicit-layers-tutorial.org

- Explicit vs implicit layers
  - An explicit layer with input x and output z corresponding to the application of some explicit function f:

$$z = f(x)$$

  - An implicit layer would instead be defined via a joint function of both x and z, where the output of of the layer z is required to satisfy some constraint such as finding the root of an equation:

$$\text{Find } z \text{ such that } g(x, z) = 0$$

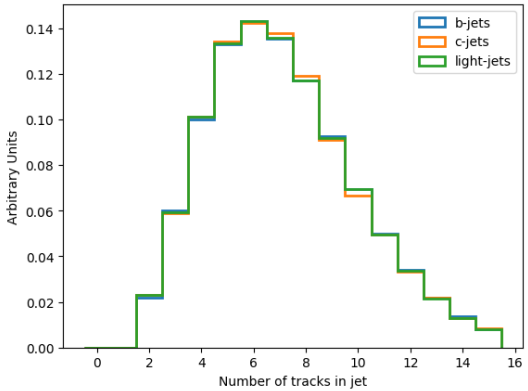
Deep Implicit Layers (II)

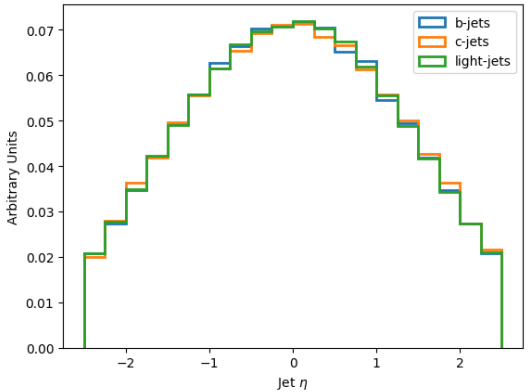
- Differentiable optimisation as a layer
  - Implicit differentiation to compute gradients of solutions of implicit functions, optimisations or differential equations.

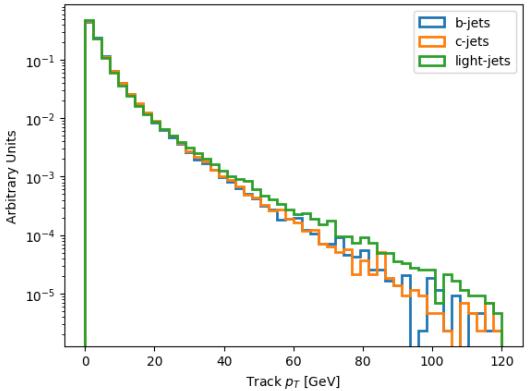
http://implicit-layers-tutorial.org

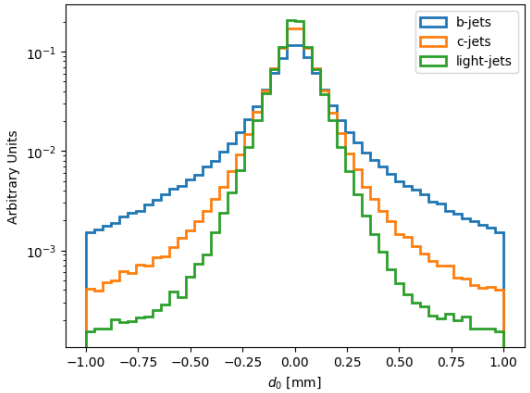
Implicit layers have the notably advantage that *we can use the implicit function theorem to directly compute gradients at the solution point of these equations*, without having to store any intermediate variables along the way. This vastly improves the memory consumption and often the numerical accuracy of these methods, providing another notable benefit for implicit models in the setting of deep learning in particular.

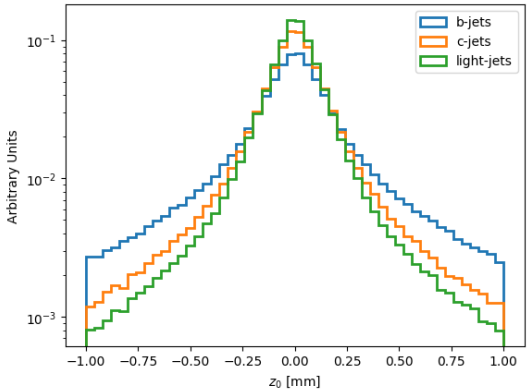
Dataset


b-jets
c-jets
light-jets
Arbitrary Units
Number of tracks in jet


b-jets
c-jets
light-jets
Arbitrary Units
Jet η


b-jets
c-jets
light-jets
Arbitrary Units
Track p_T [GeV]


b-jets
c-jets
light-jets
Arbitrary Units
d_0 [mm]


b-jets
c-jets
light-jets
Arbitrary Units
z_0 [mm]