Differentiable Vertex Fitting for Jet Flavour Tagging

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

ICHEP 2024

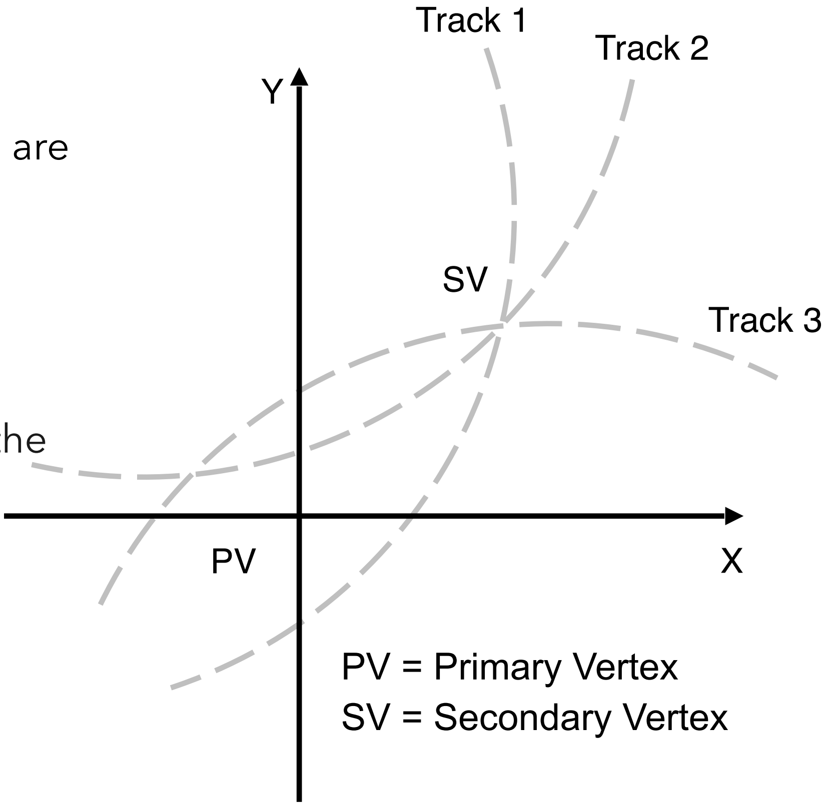
July 18-24, 2024

Fundação para a Ciência e a Tecnologia
"la Caixa"
LIP
SLAC NATIONAL ACCELERATOR LABORATORY

Introduction (I)

- **Secondary vertex (SV):** a point where particles are produced in a collision or a decay

- **SV reconstruction:**
  1. What set of particles have been produced at the same vertex?
  2. What is the vertex position?
  3. Can we improve the estimate of the track parameters by imposing a vertex constraint?



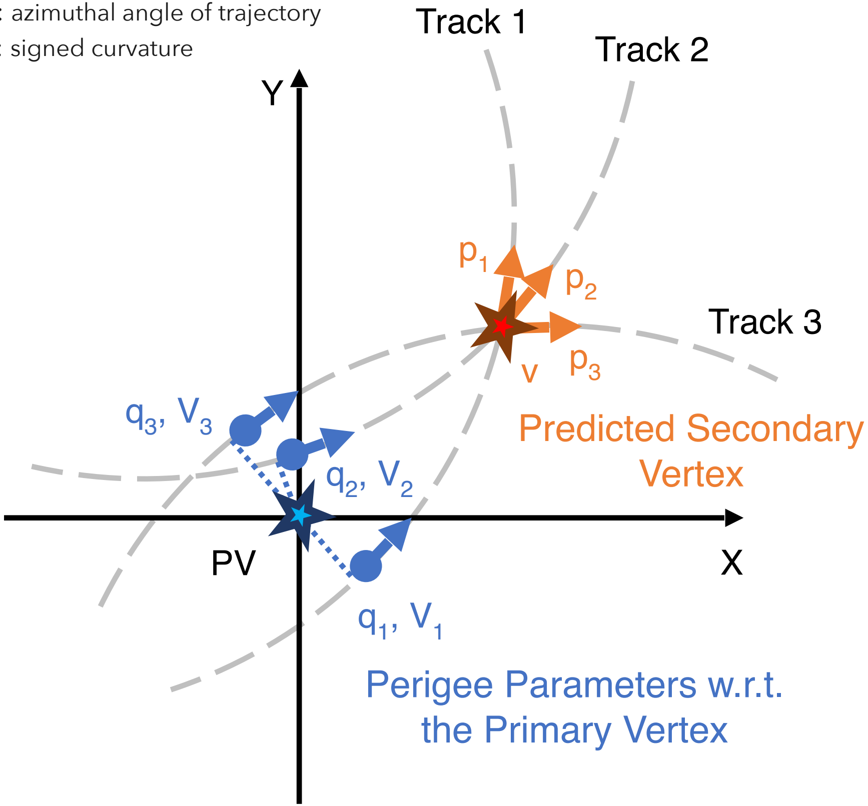
Introduction (II)

$q_i = (d_0, z_0, \phi, \theta, \rho)$
d_0 : signed transverse impact parameter
z_0 : longitudinal impact parameter
ϕ : polar angle of trajectory
θ : azimuthal angle of trajectory
ρ : signed curvature

- **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 vertex position $\mathbf{v}$ and the momentum vectors $\mathbf{p}_i$ of all tracks at the vertex.

➡ E.g. via the minimisation of a weighted χ^2



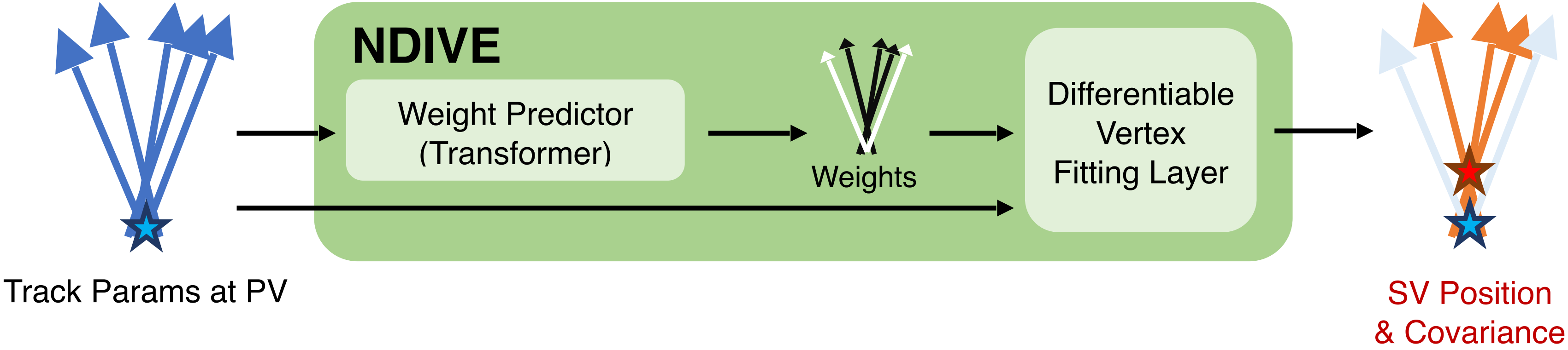
Aim of this work

- Secondary vertex reconstruction is usually performed by **manually optimised** / low-level algorithms.
  - The outcome is then fed into downstream machine learning algorithms (***DL1*** by ATLAS).
- In state-of-the-art algorithms (***GN1 & GN2*** by ATLAS), a single end-to-end neural network is employed with no intermediate low-level algorithms, but also **no explicit secondary (or tertiary) vertex reconstruction**.

Can we integrate vertex reconstruction into a ML end-to-end trainable algorithm?

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.



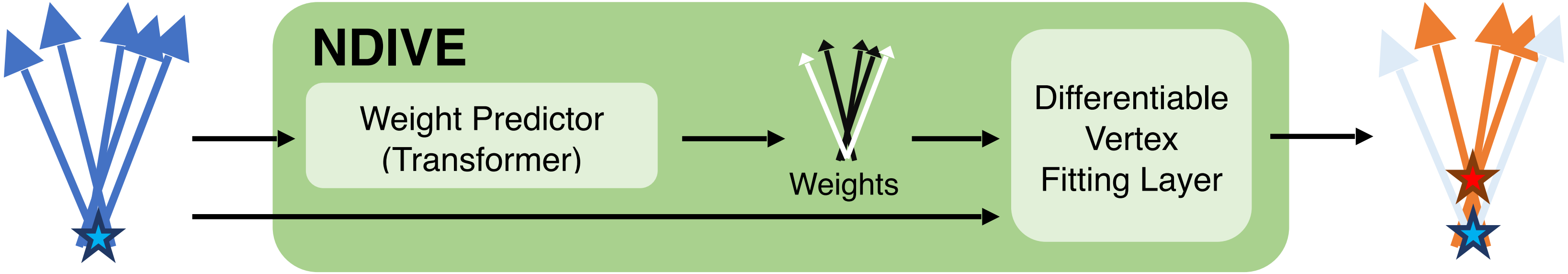
- Vertex fitting formulated as an optimization problem, and using implicit differentiation to compute the derivative of the fitted vertex.
- Differentiable programming for integrating domain knowledge into NN training.

NDIVE: Neural Differentiable Vertexing layer

Transformer neural network trained to assign weights to tracks

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

Loss function: mean euclidean distance between true and predicted vertex



Track Params at PV

$\{w_i\}$

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

SV Position & Covariance

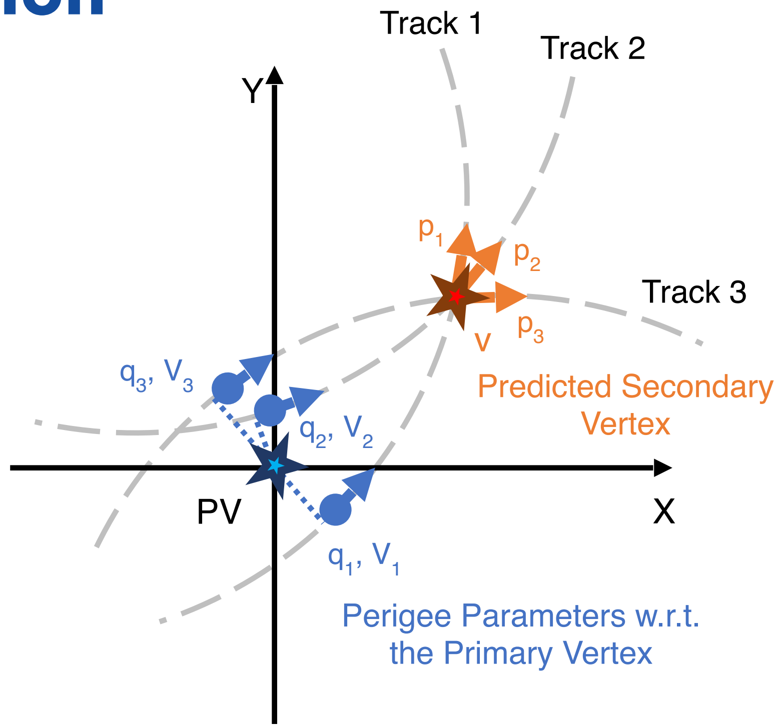
q_i, V_i

Once a vertex solution is found, the track weights are adjusted by leveraging the differentiable vertex fitter.

Billoir, Qian in Fast vertex fitting with a local parametrization of tracks

Inclusive Vertex Fit formulation

- Values to be optimised: $\mathbf{x} = (\mathbf{v}, \{\mathbf{p}_i\})$
- Input data: $q_i = (d_0, z_0, \phi, \theta, \rho)$ and V_i



Billoir, Qian in Fast vertex fitting with a local parametrization of tracks

Inclusive Vertex Fit formulation

- Values to be optimised: $\mathbf{x} = (\mathbf{v}, \{\mathbf{p}_i\})$
- Input data: $q_i = (d_0, z_0, \phi, \theta, \rho)$ and V_i
- The following objective function is minimised:

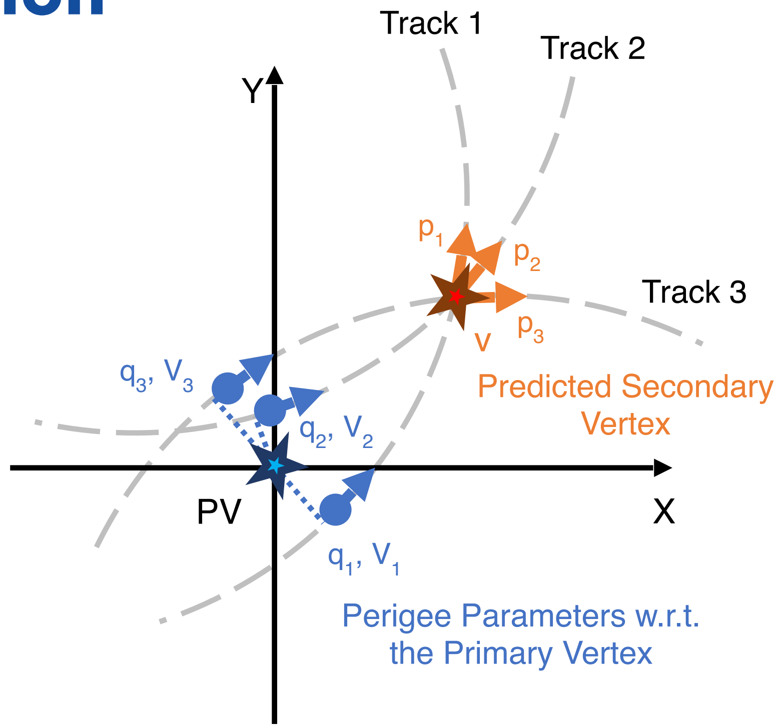
$$\chi^2 = \sum_{i=1}^{N} w_i (\mathbf{q}_i - \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i))^T V_i^{-1} (\mathbf{q}_i - \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i))$$

$$\hat{\mathbf{x}}(\text{tracks}, w) = \arg\min_x \chi^2(\mathbf{x}; \text{tracks}, w)$$

$\mathbf{h}_i(\mathbf{v}, \mathbf{p}_i)$: track model

w_i : weight of track i to the vertex fit

implicitly dependent on w

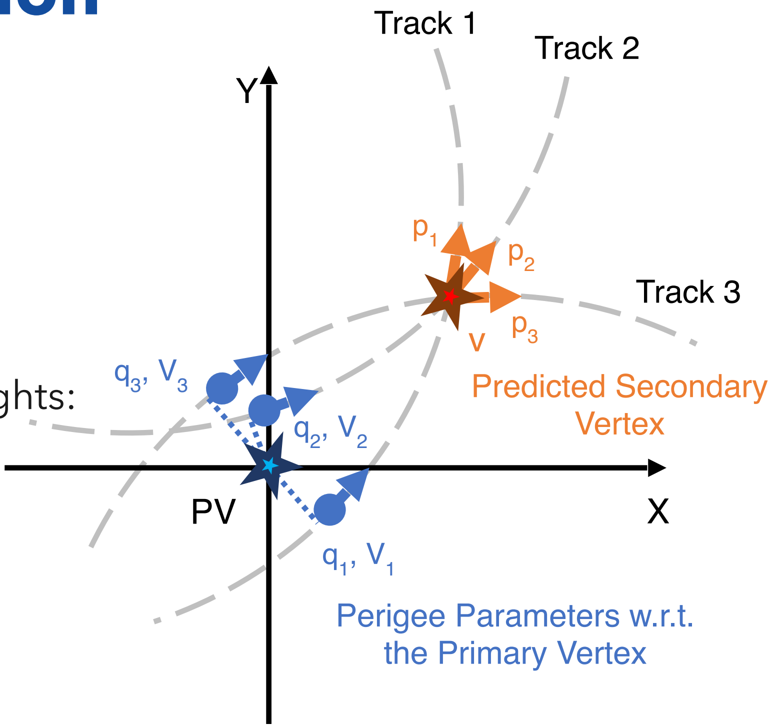


Billoir, Qian in Fast vertex fitting with a local parametrization of tracks

Inclusive Vertex Fit formulation

- Values to be optimised: $\mathbf{x} = (\mathbf{v}, \{\mathbf{p}_i\})$
- Input data: $q_i = (d_0, z_0, \phi, \theta, \rho)$ and V_i
- The implicit function theorem tells us we can take the derivatives of the fitted vertex with respect to the weights:

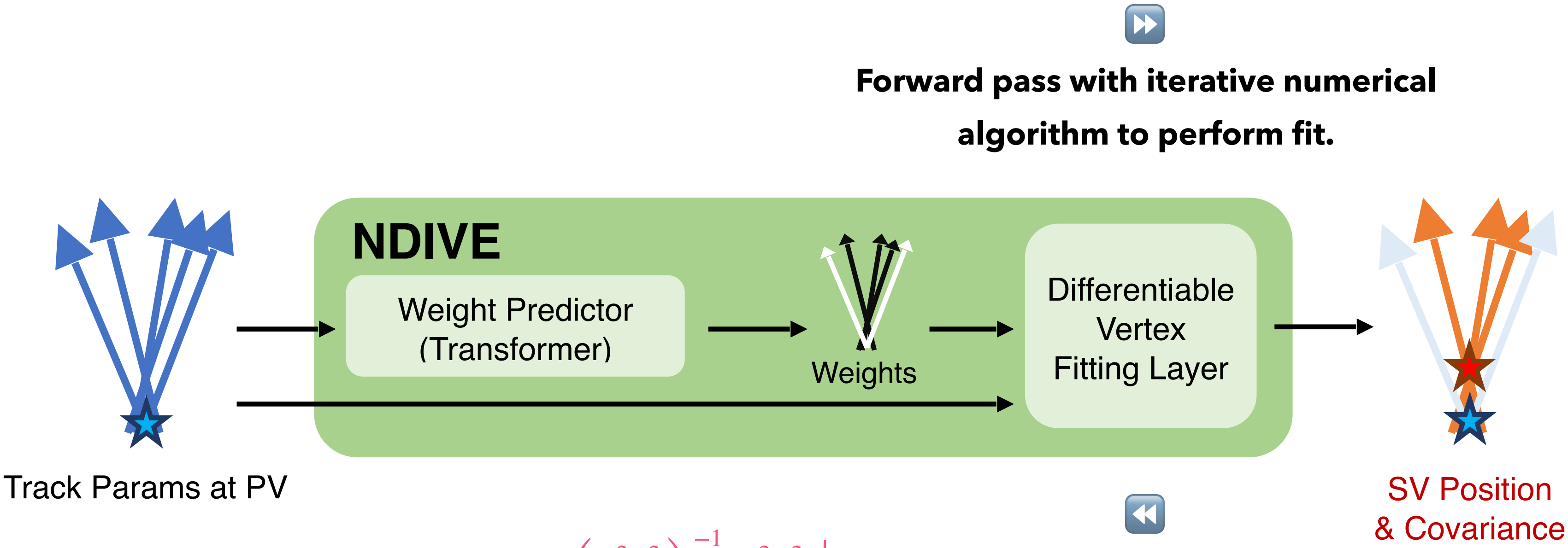
Note: $\left.\frac{\partial \chi^2}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}} = 0$, and $\frac{d}{dw}\left(\left.\frac{\partial \chi^2}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}}\right) = \frac{d}{dw}(0) = 0$



Accounting for the implicit dependence of $\hat{\mathbf{x}}$ on w:

$$0 = \left.\frac{\partial^2 \chi^2}{\partial \mathbf{x}}\right|_{\mathbf{x}=\hat{\mathbf{x}}} \frac{d\hat{\mathbf{x}}}{dw} + \left.\frac{\partial^2 \chi^2}{\partial \mathbf{x} \partial w}\right|_{\mathbf{x}=\hat{\mathbf{x}}} \Rightarrow \frac{\partial \hat{\mathbf{x}}}{\partial w} = -\left(\frac{\partial^2 \chi^2}{\partial \mathbf{x}^2}\right)^{-1} \left.\frac{\partial^2 \chi^2}{\partial \mathbf{x} \partial w}\right|_{\mathbf{x}=\hat{\mathbf{x}}}$$

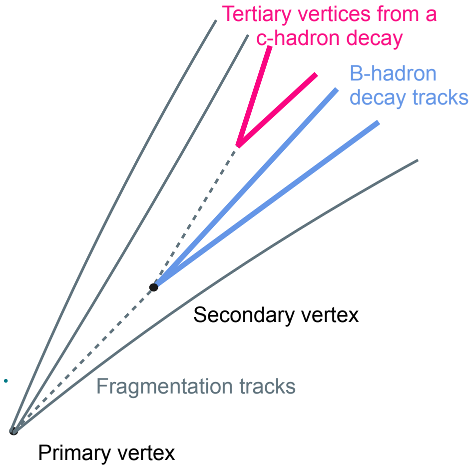
NDIVE: Neural Differentiable Vertexing layer



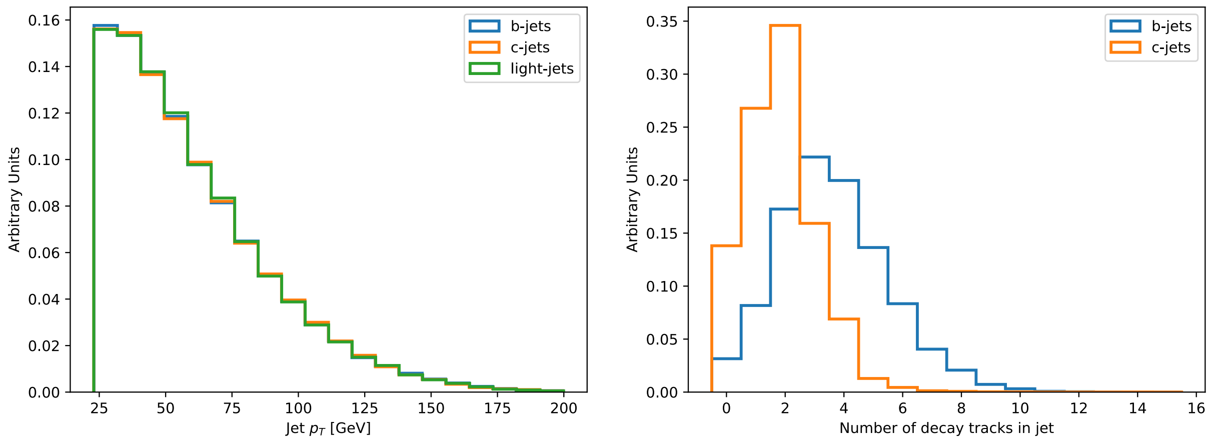
$$\frac{\partial \hat{\mathbf{x}}}{\partial w} = -\left(\frac{\partial^2 \chi^2}{\partial \mathbf{x}^2}\right)^{-1} \frac{\partial^2 \chi^2}{\partial \mathbf{x} \partial w}\Bigg|_{\mathbf{x}=\hat{\mathbf{x}}}$$

Backward pass done with a custom derivative

Dataset & Inputs



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

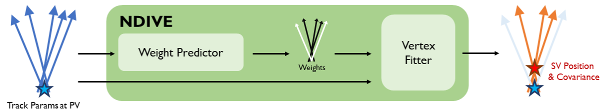


Training features:

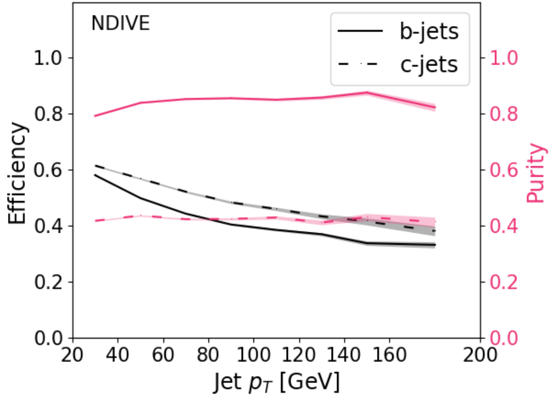
- Track perigee parameters and their errors
- Signed d_0 and z_0 significances
- log(track p_T / jet p_T)
- ΔR (track, jet)

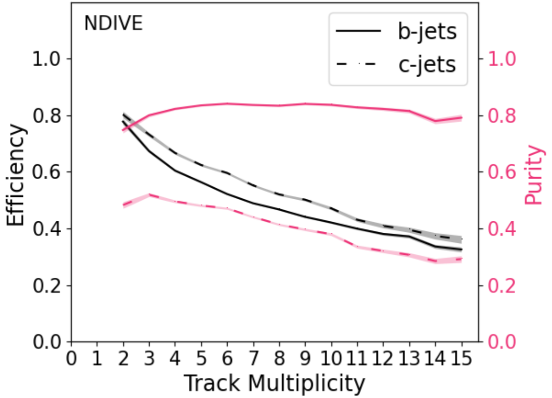
[Zenodo: Secondary Vertex Finding in Jets Dataset]

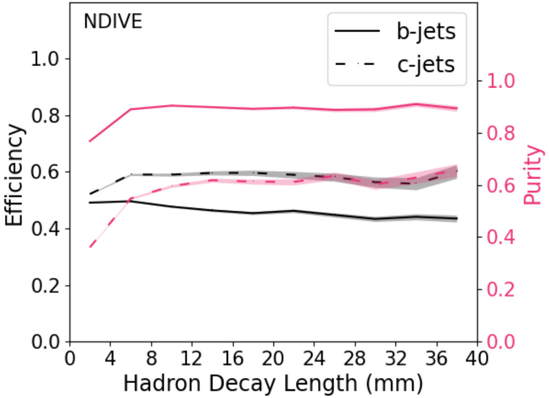
Track selection performance



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

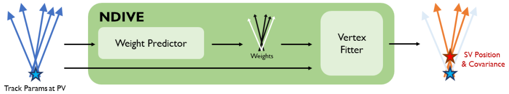




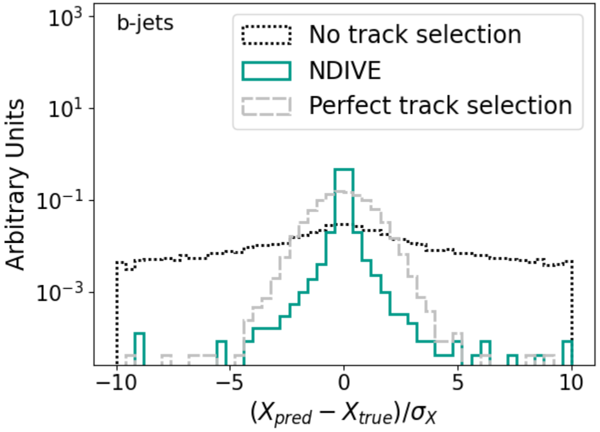


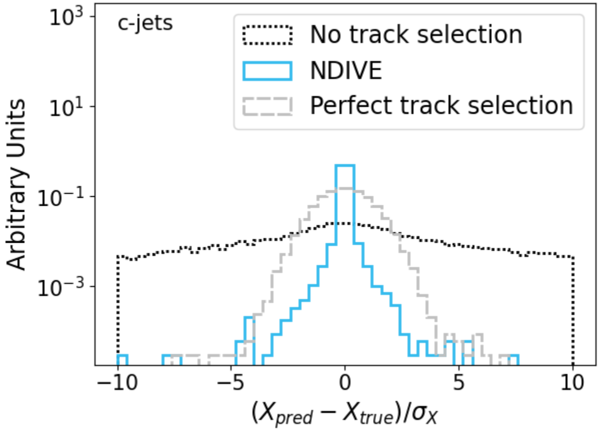
- "Selected tracks": per-track weights normalised by maximum weight in each jet and required to be above > 0.5

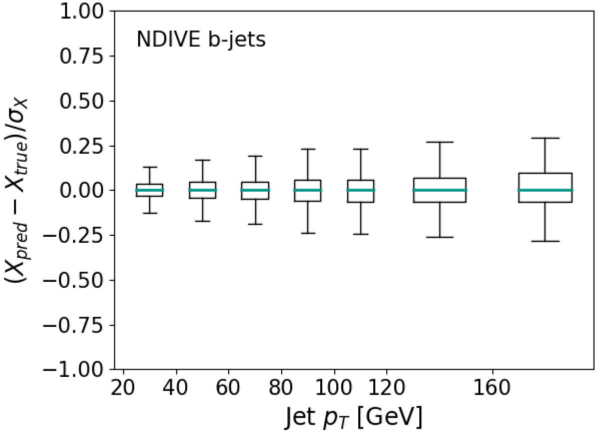
Vertex reconstruction performance



- NDIVE makes accurate unbiased estimates of secondary vertex positions.





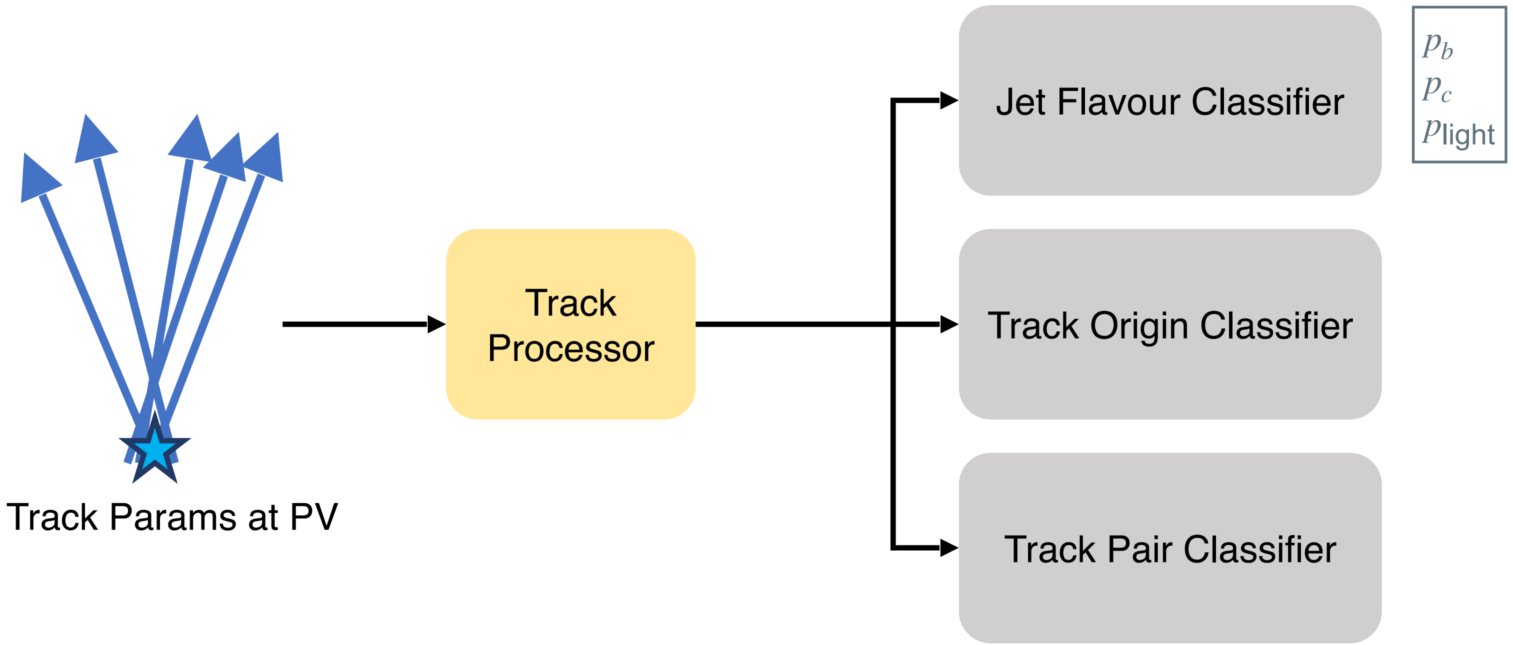


- "Perfect track selection": weights set to 0 or 1 based on true origin of track.
- "No track selection": all tracks in the jet are used in the fit.
- (Right) Boxes indicate IQR of distributions; error bars cover data points that fall within 1.5 x IQR.

ATL-PHYS-PUB-2022-027
ATL-PHYS-PUB-2023-021

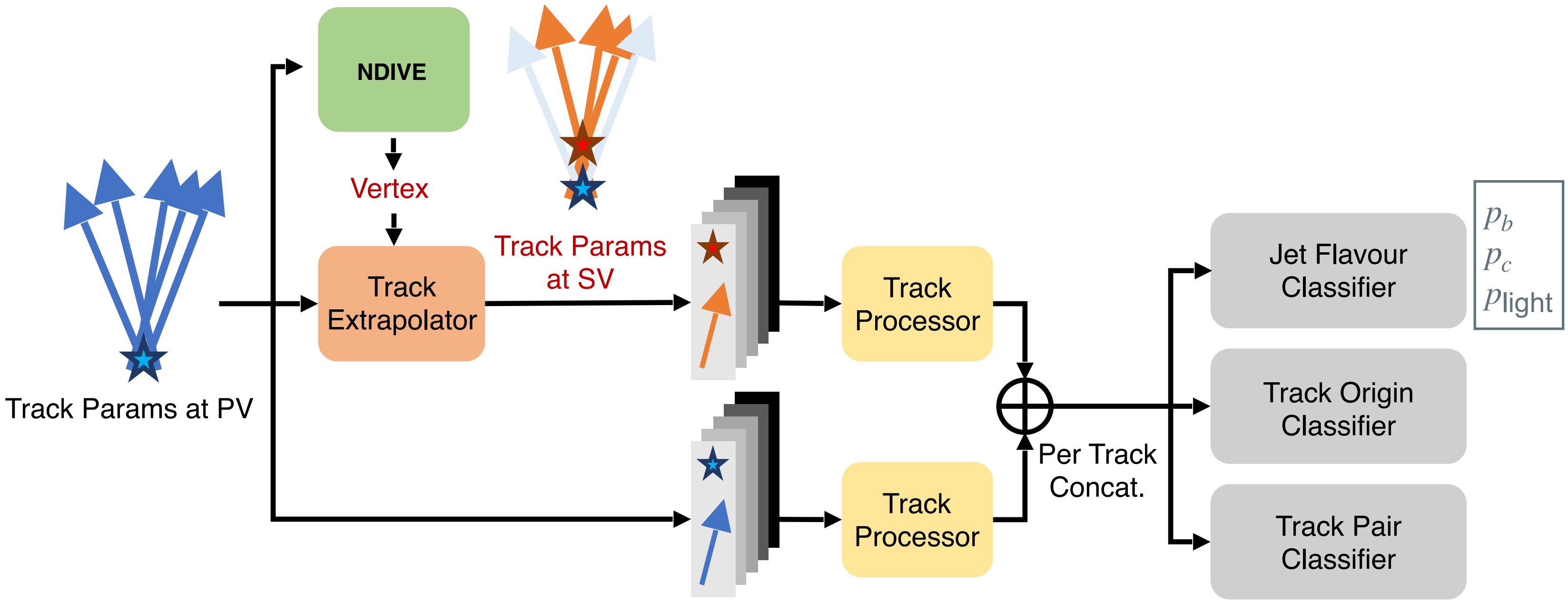
Integration in a flavour-tagging model

FTAG baseline



Integration in a flavour-tagging model

FTAG+NDIVE



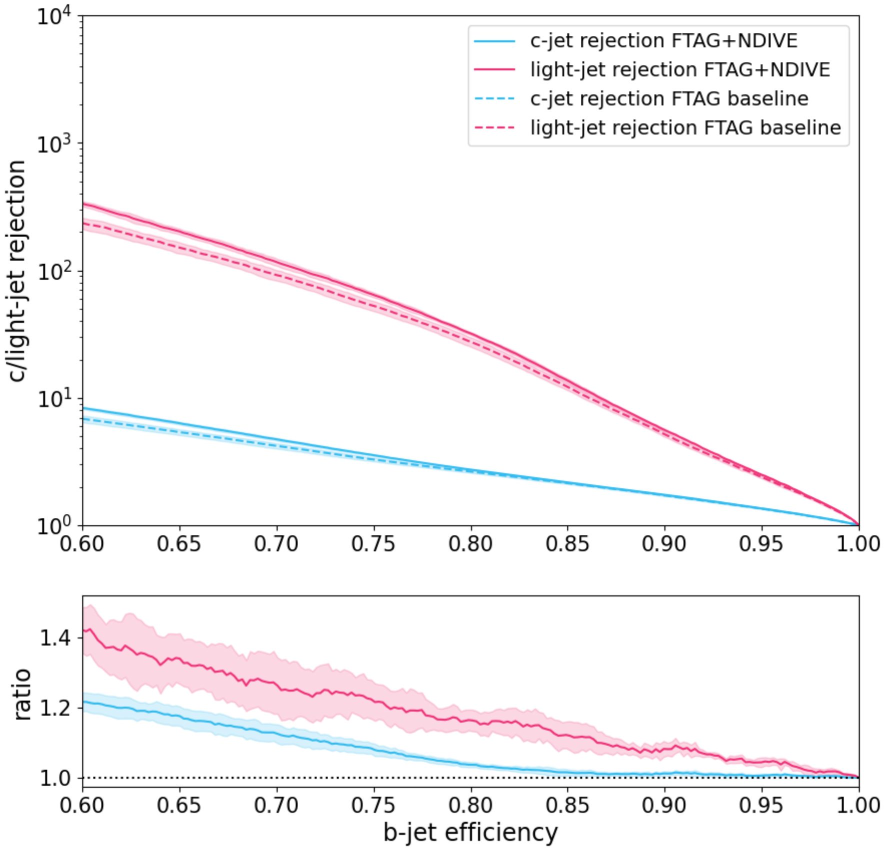
This is one possible way of integrating NDIVE, other formulations are possible.

Model comparison: ROC curve

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

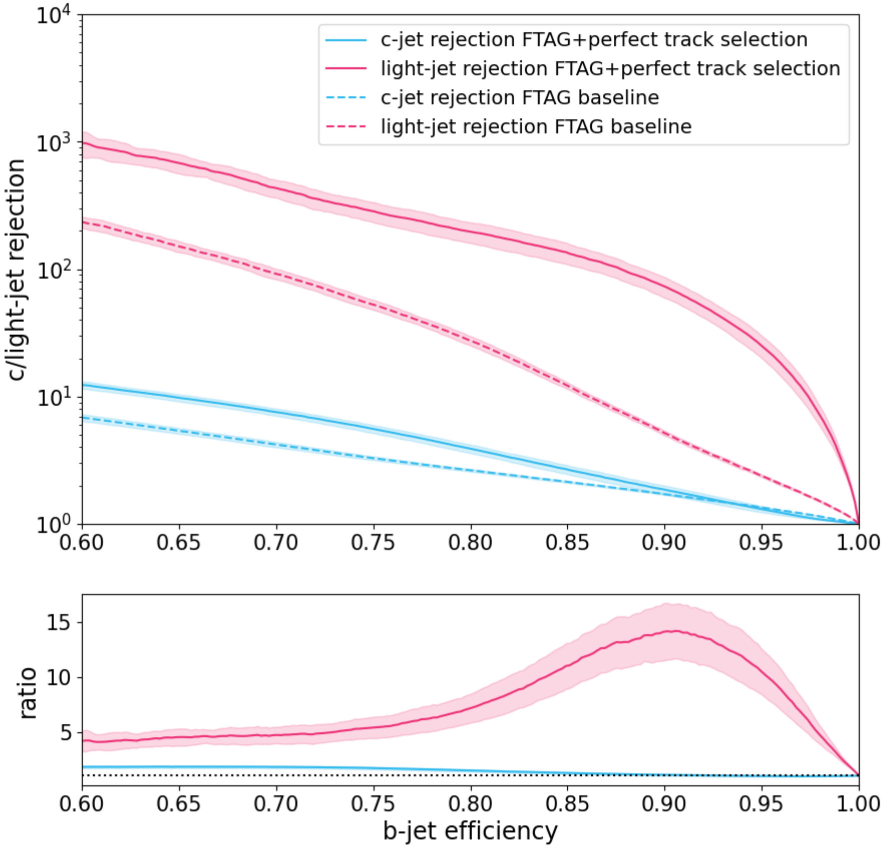
$f_c = 0.05$

- NDIVE integration improves flavour tagging performance.



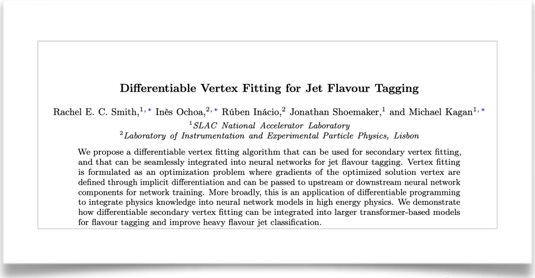
Future prospects

- These methodological developments are generic, applicable to other vertex fitting algorithms and other schemes for integrating vertex information into neural networks.
- Further improvement is possible with better track selection methods, as represented by the ideal scenario model where the tracks are selected “perfectly”.



Summary

[preprint](#) arXiv


Differentiable Vertex Fitting for Jet Flavour Tagging

Rachel E. C. Smith,[1,*] Inês Ochoa,[2,*] Rúben Inácio,[2] Jonathan Shoemaker,[1] and Michael Kagan[1,*]
[1]*SLAC National Accelerator Laboratory*
[2]*Laboratory of Instrumentation and Experimental Particle Physics, Lisbon*

We propose a differentiable vertex fitting algorithm that can be used for secondary vertex fitting, and that can be seamlessly integrated into neural networks for jet flavour tagging. Vertex fitting is formulated as an optimization problem where gradients of the optimized solution vertex are defined through implicit differentiation and can be passed to upstream or downstream neural network components for network training. More broadly, this is an application of differentiable programming to integrate physics knowledge into neural network models in high energy physics. We demonstrate how differentiable secondary vertex fitting can be integrated into larger transformer-based models for flavour tagging and improve heavy flavour jet classification.

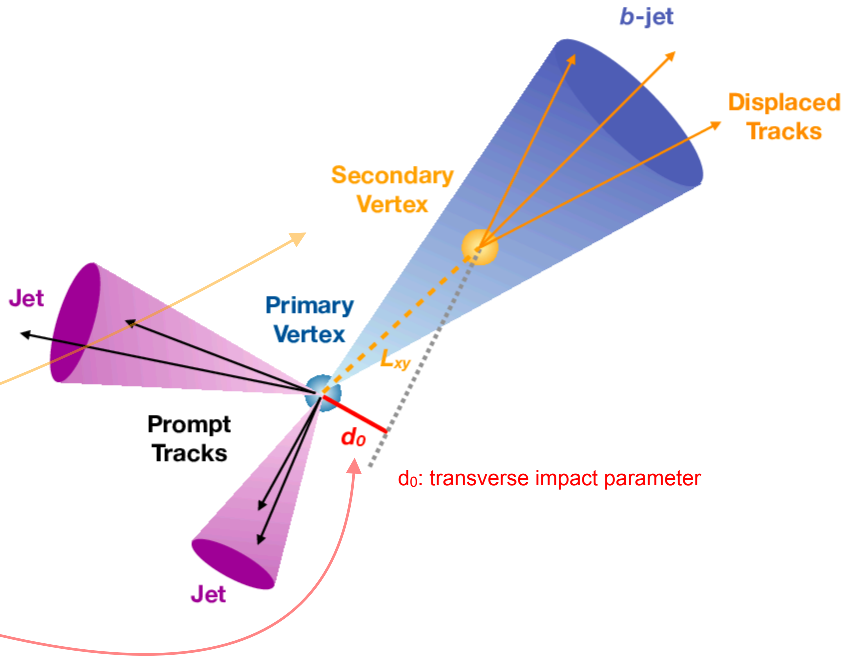
github repository

- We introduce NDIVE: a neural differentiable vertexing layer
  - First differentiable vertex fitting algorithm.
- 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 NNs:
  - NDIVE can be integrated into b-tagging algorithms, explicitly reintroducing vertex geometry.
  - Part of wider application of differentiable programming to HEP!

Backup

b-quarks → b-hadrons → b-jets

- **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



Track parameterisation

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

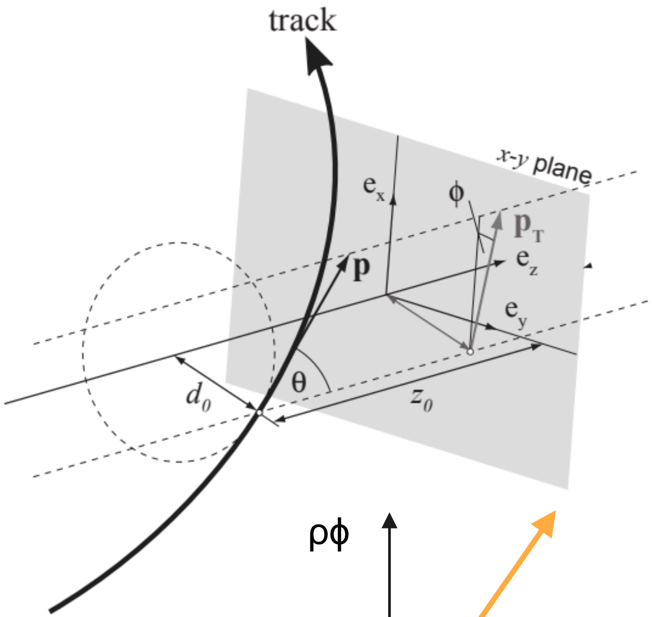
d_0 : signed transverse impact parameter

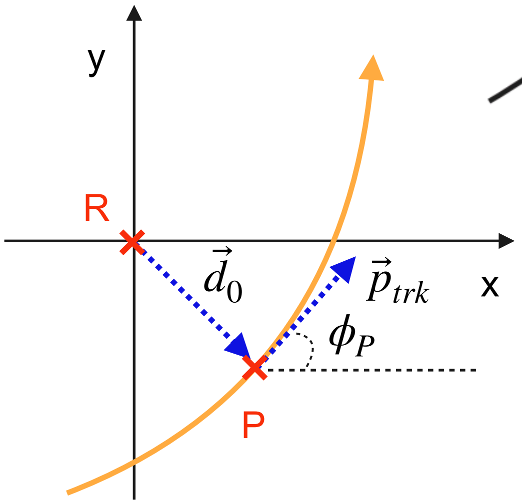
z_0 : longitudinal impact parameter

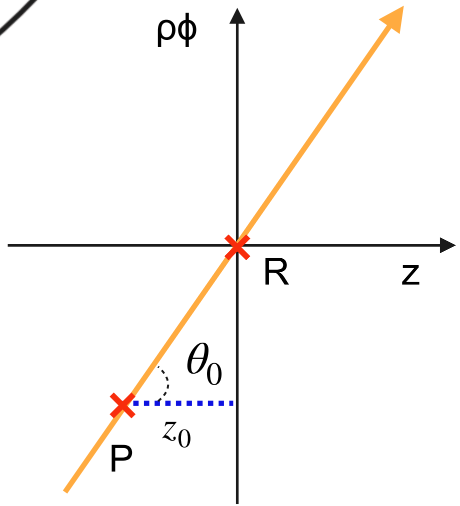
ϕ : polar angle of trajectory

θ : azimuthal angle of trajectory

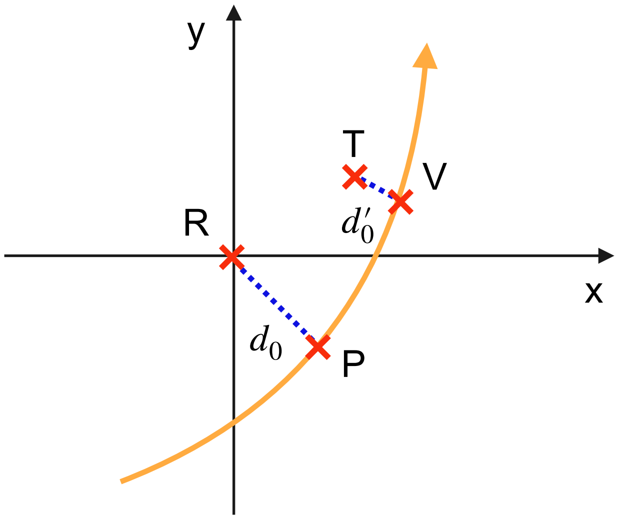
ρ : signed curvature







Track Extrapolator



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

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

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

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

Additional track representations can be defined by considering alternative reference points (the NDIVE secondary vertex estimate) and finding the point of closest approach to the trajectory.

- Implemented using JAX's autodiff.

Billoir algorithm for inclusive vertex fitting

- Track parameters defined as nonlinear function of the vertex position and momentum vectors of the tracks at that position: $\mathbf{q}_i = \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i)$
- First-order Taylor expansion of $\mathbf{h}_i$ expanded at an estimate of the vertex position and track momenta: $\mathbf{A}_i = \left.\frac{\partial \mathbf{h}_i}{\partial \mathbf{v}}\right|_{\mathbf{e}_0}$
  $\mathbf{q}_i \approx \mathbf{A}_i\mathbf{v} + \mathbf{B}_i\mathbf{p}_i + \mathbf{c}_i$
- Iterate fit until convergence, expanding the functions $\mathbf{h}_i$ around the new expansion point each time: $\mathbf{B}_i = \left.\frac{\partial \mathbf{h}_i}{\partial \mathbf{p}_i}\right|_{\mathbf{e}_0}$

$$\hat{\mathbf{v}} = \mathbf{C}\sum_{i=1}^{N} \mathbf{A}_i^T \mathbf{G}_i (\mathbf{I} - \mathbf{B}_i \mathbf{W}_i \mathbf{B}_i^T \mathbf{G}_i)(\mathbf{q}_i - \mathbf{c}_i)$$

$$\hat{\mathbf{p}}_i = \mathbf{W}_i \mathbf{B}_i^T \mathbf{G}_i (\mathbf{q}_i - \mathbf{c}_i - \mathbf{A}_i \hat{\mathbf{v}}), \quad i = 1, \ldots, N$$

$$\begin{aligned} \mathbf{G}_i &= \mathbf{V}_i^{-1} \\ \mathbf{D}_i &= \mathbf{A}_i^T \mathbf{G}_i \mathbf{B}_i \\ \mathbf{D}_0 &= \sum_{i=1}^{N} \mathbf{A}_i^T \mathbf{G}_i \mathbf{A}_i \\ \mathbf{W}_i^{-1} &= \mathbf{B}_i^T \mathbf{G}_i \mathbf{B}_i \end{aligned}$$

$$\mathbf{C} = \left(\mathbf{D}_0 - \sum_{i=1}^{N} \mathbf{D}_i \mathbf{W}_i \mathbf{D}_i^T\right)^{-1}$$

- Afterwards we rewrite the track parameters $\hat{\mathbf{q}}_i = \mathbf{h}_i(\hat{\mathbf{v}}, \hat{\mathbf{p}}_i)$.
- The χ^2 statistic of the fit is then:

$$\chi^2 = \sum_{i=1}^{N} (\mathbf{q}_i - \hat{\mathbf{q}}_i)^T \mathbf{G}_i (\mathbf{q}_i - \hat{\mathbf{q}}_i)$$

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 (when evaluated at $\hat{\mathbf{x}}(\alpha)$):

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

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

- 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 the fitted vertex with respect to the input parameters (at the solution point!)

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:

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

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

Implicit layers (II)

http://implicit-layers-tutorial.org

The implicit function theorem. *Let* $f : \mathbb{R}^p \times \mathbb{R}^n \to \mathbb{R}^n$ *and* $a_0 \in \mathbb{R}^p$, $z_0 \in \mathbb{R}^n$ *be such that*

1. $f(a_0, z_0) = 0$, *and*
2. f *is continuously differentiable with non-singular Jacobian* $\partial_1 f(a_0, z_0) \in \mathbb{R}^{n \times n}$.

Then there exist open sets $S_{a_0} \subset \mathbb{R}^p$ *and* $S_{z_0} \subset R^n$ *containing* a_0 *and* z_0, *respectively, and a unique continuous function* $z^* : S_{a_0} \to S_{z_0}$ *such that*

1. $z_0 = z^*(a_0)$,
2. $f(a, z^*(a)) = 0 \quad \forall a \in S_{a_0}$, *and*
3. z^* *is differentiable on* S_{a_0}.