Differentiable Vertex Fitting for Jet Flavour Tagging

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

6th Inter-Experiment Machine Learning Workshop
30 January 2024

"la Caixa" LIP SLAC NATIONAL ACCELERATOR LABORATORY

Differentiable Vertex Fitting for Jet Flavour Tagging

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

Rúben is giving the poster!

6th Inter-Experiment Machine Learning Workshop
30 January 2024

"la Caixa" LIP SLAC NATIONAL ACCELERATOR LABORATORY

Outline

- A history of vertex fitting algorithms in ATLAS
- Implicit differentiation and NDIVE: a differentiable vertex fitting algorithm
- Application of NDIVE in a flavour-tagging network
- Future work

Differentiable Vertex Fitting for Jet Flavour Tagging

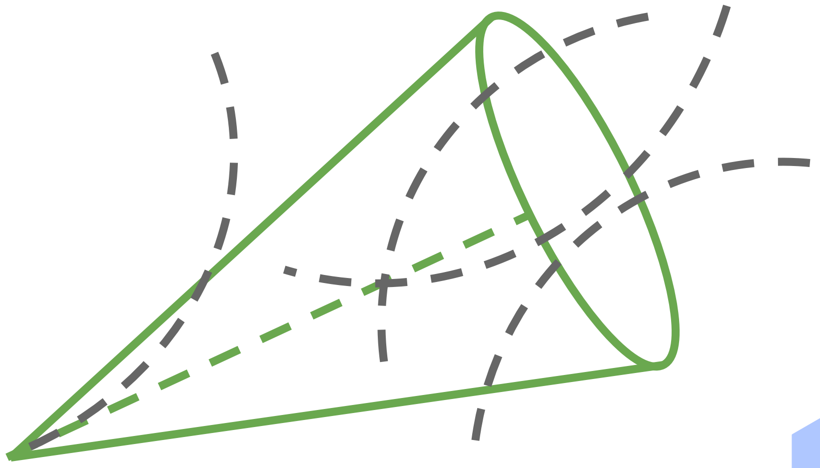
Rachel E. C. Smith,[1,*] Inês Ochoa,[2,*] Rúben Inácio,[2] Jonathan Shoemaker,[1] and Michael Kagan[1,*]
[1] *SLAC National Accelerator Laboratory*
[2] *Laboratory of Instrumentation and Experimental Particle Physics, Lisbon*

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.

arxiv preprint
github repo

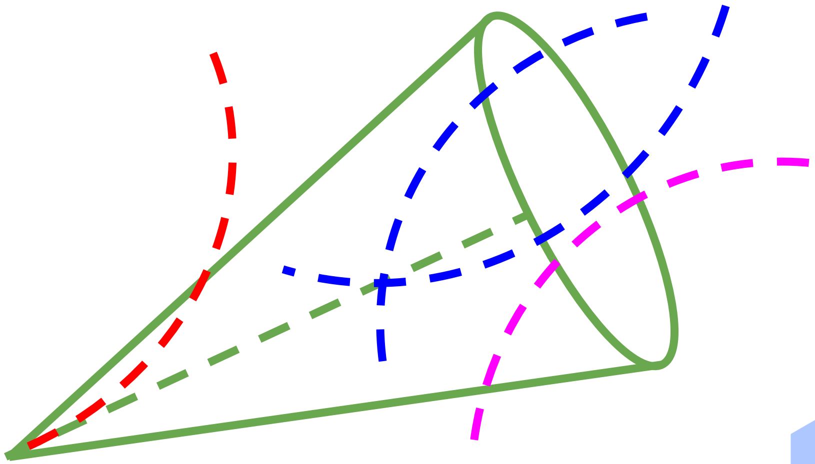
What do we mean by vertex *fitting*?

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



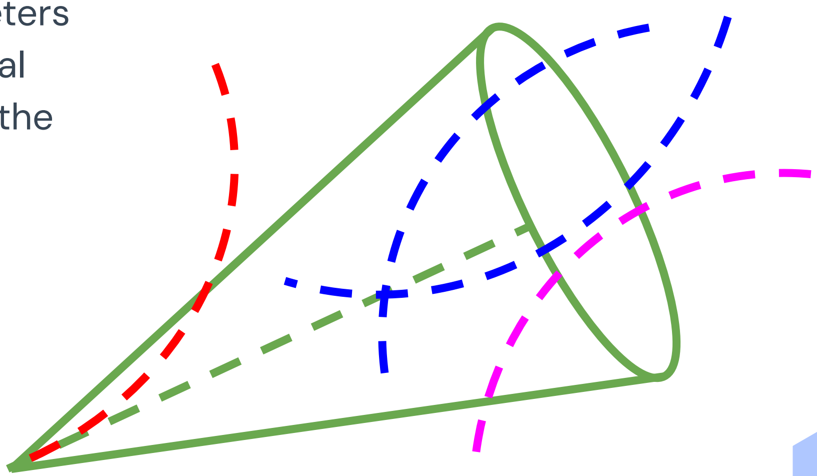
What do we mean by vertex *fitting*?

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



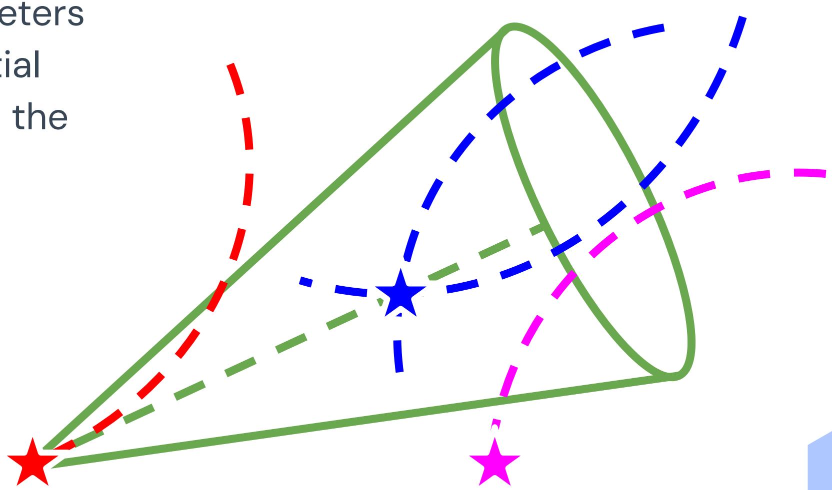
What do we mean by vertex *fitting*?

- **Vertex finding**: Identifying tracks that originate at the same point in space
- **Vertex fitting**: Given a set of tracks, compute the best estimate of the vertex parameters (e.g. predicted vertex position, and initial momentum vectors of all the tracks at the predicted vertex position)



What do we mean by vertex *fitting*?

- **Vertex finding**: Identifying tracks that originate at the same point in space
- **Vertex fitting**: Given a set of tracks, compute the best estimate of the vertex parameters (e.g. predicted vertex position, and initial momentum vectors of all the tracks at the predicted vertex position)



History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

1. Find all 2-track vertices

History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

1. Find all 2-track vertices
2. Reject vertices that come from unrelated vertices of interest (e.g. photon conversion, hadronic detector interaction, long-lived decays)

History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

1. Find all 2-track vertices
2. Reject vertices that come from unrelated vertices of interest (e.g. photon conversion, hadronic detector interaction, long-lived decays)
3. Using non-vetoed tracks, form a single inclusive secondary vertex

History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

1. Find all 2-track vertices
2. Reject vertices that come from unrelated vertices of interest (e.g. photon conversion, hadronic detector interaction, long-lived decays)
3. Using non-vetoed tracks, form a single inclusive secondary vertex
4. If the resulting vertex has small probability, discard the track with the highest contribution to the vertex X^2

History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

1. Find all 2-track vertices
2. Reject vertices that come from unrelated vertices of interest (e.g. photon conversion, hadronic detector interaction, long-lived decays)
3. Using non-vetoed tracks, form a single inclusive secondary vertex
4. If the resulting vertex has small probability, discard the track with the highest contribution to the vertex X^2
5. Repeat until given threshold (w/ invariant mass < 6 GeV)

History of vertex fitting in ATLAS

Inclusive SV algorithm (SV1)

https://cds.cern.ch/record/2270366/
https://arxiv.org/abs/2211.16345

1. Find all 2-track vertices
2. Reject vertices that come from unrelated vertices of interest (e.g. photon conversion, hadronic detector interaction, long-lived decays)
3. Using non-vetoed tracks, form a single inclusive secondary vertex
4. If the resulting vertex has small probability, discard the track with the highest contribution to the vertex X^2
5. Repeat until given threshold (w/ invariant mass < 6 GeV)
6. Final vertex typically combines decay products from b- and c-hadrons

History of vertex fitting in ATLAS

Decay chain fitter (JetFitter)

https://cds.cern.ch/record/2645405/
https://arxiv.org/abs/2211.16345

History of vertex fitting in ATLAS

Decay chain fitter (JetFitter)

https://cds.cern.ch/record/2645405/
https://arxiv.org/abs/2211.16345

1. Differs from SV1 in that several vertices can be reconstructed simultaneously

History of vertex fitting in ATLAS

Decay chain fitter (JetFitter)

https://cds.cern.ch/record/2645405/
https://arxiv.org/abs/2211.16345

1. Differs from SV1 in that several vertices can be reconstructed simultaneously
2. A modified Kalman filter is used to find a shared line on which the primary, b-, and c-vertices lie, approximating the flight path

History of vertex fitting in ATLAS

https://cds.cern.ch/record/2645405/
https://arxiv.org/abs/2211.16345

Decay chain fitter (JetFitter)

1. Differs from SV1 in that several vertices can be reconstructed simultaneously
2. A modified Kalman filter is used to find a shared line on which the primary, b-, and c-vertices lie, approximating the flight path
3. Minimize X^2 containing the weighted residuals of the tracks wrt their vertices

History of vertex fitting in ATLAS

Decay chain fitter (JetFitter)

https://cds.cern.ch/record/2645405/
https://arxiv.org/abs/2211.16345

1. Differs from SV1 in that several vertices can be reconstructed simultaneously
2. A modified Kalman filter is used to find a shared line on which the primary, b-, and c-vertices lie, approximating the flight path
3. Minimize X^2 containing the weighted residuals of the tracks wrt their vertices
4. Cluster combinations of two vertices until some compatibility threshold, then do another X^2 fit

History of vertex fitting in ATLAS

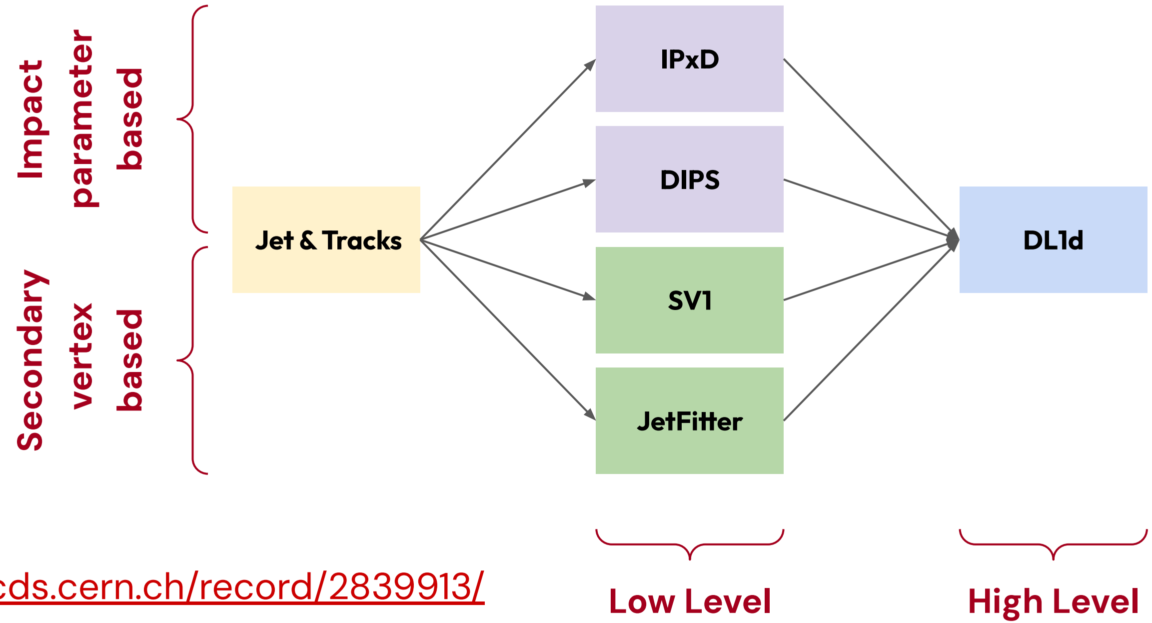
Decay chain fitter (JetFitter)

https://cds.cern.ch/record/2645405/
https://arxiv.org/abs/2211.16345

1. Differs from SV1 in that several vertices can be reconstructed simultaneously
2. A modified Kalman filter is used to find a shared line on which the primary, b-, and c-vertices lie, approximating the flight path
3. Minimize X^2 containing the weighted residuals of the tracks wrt their vertices
4. Cluster combinations of two vertices until some compatibility threshold, then do another X^2 fit
5. Iterate until no more pairs of vertices above certain compatibility threshold

History of vertex fitting in ATLAS

Current ATLAS baseline flavour tagger (DL1d)

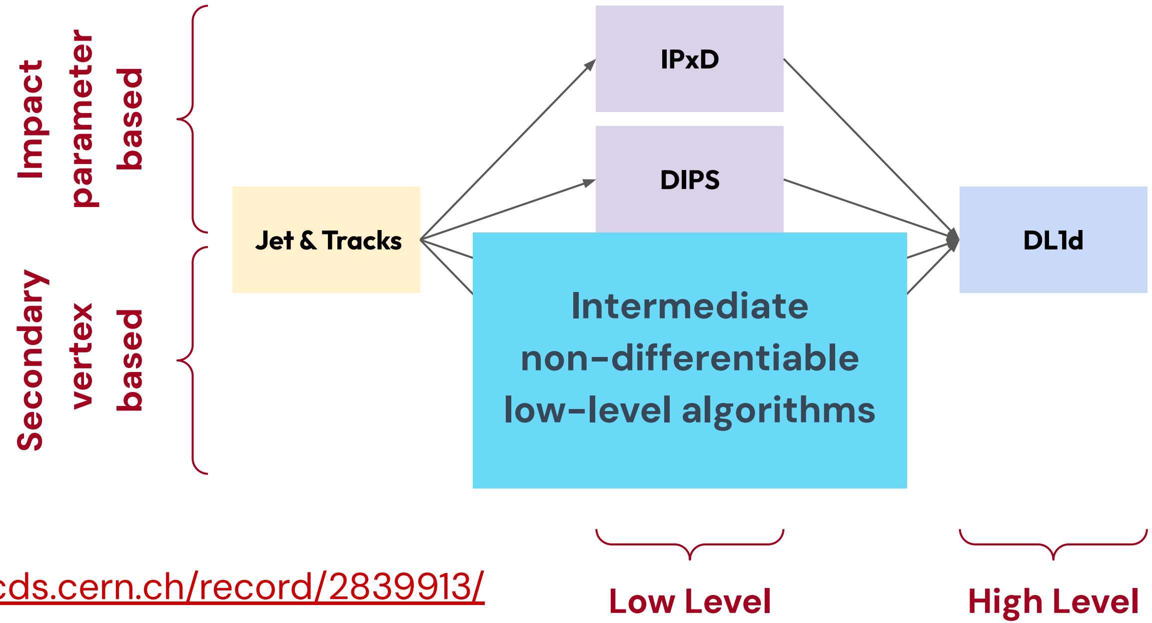


https://cds.cern.ch/record/2839913/

SV1 | JetFitter | **DL1d** | GN1 & GN2

History of vertex fitting in ATLAS

Current ATLAS baseline flavour tagger (DL1d)



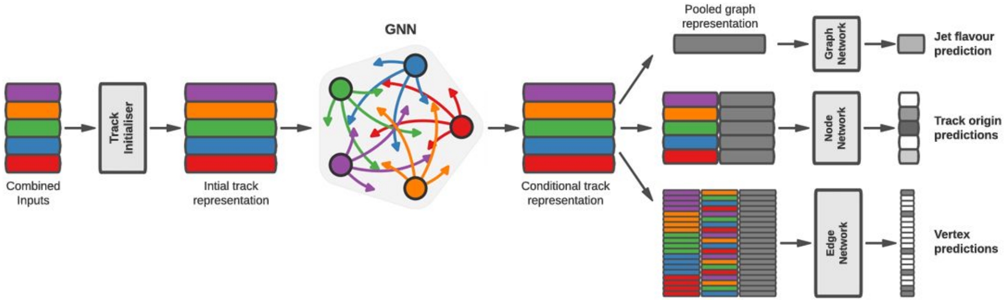
https://cds.cern.ch/record/2839913/

SV1 | JetFitter | **DL1d** | GN1 & GN2

History of vertex fitting in ATLAS

GN1 & GN2

https://cds.cern.ch/record/2811135/

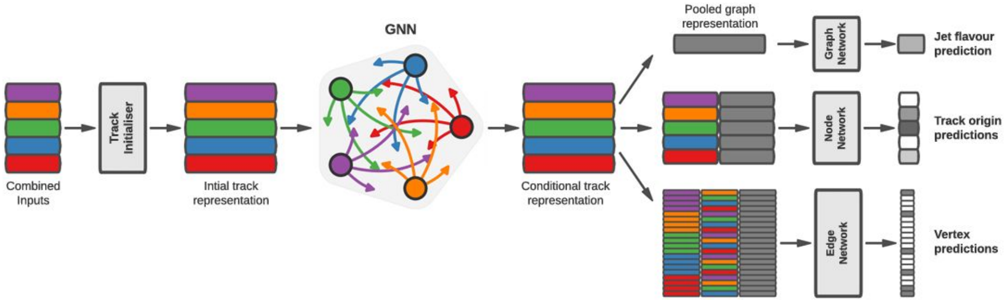


History of vertex fitting in ATLAS

GN1 & GN2

https://cds.cern.ch/record/2811135/

1. New state-of-the-art end-to-end neural networks (graph or transformer)

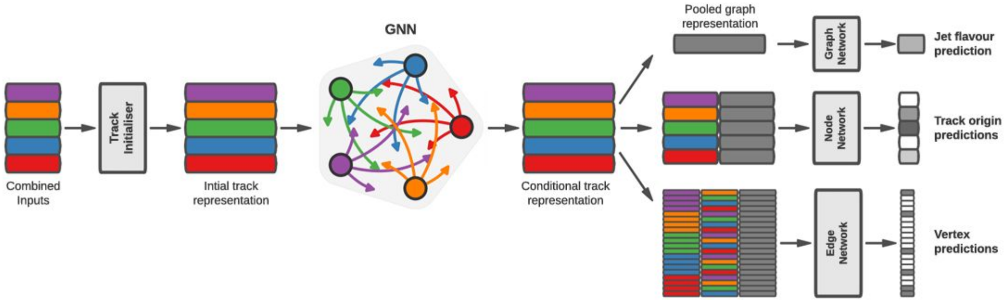


History of vertex fitting in ATLAS

GN1 & GN2

https://cds.cern.ch/record/2811135/

1. New state-of-the-art end-to-end neural networks (graph or transformer)
2. Auxiliary training objectives (track origin, track pair, jet flavour)

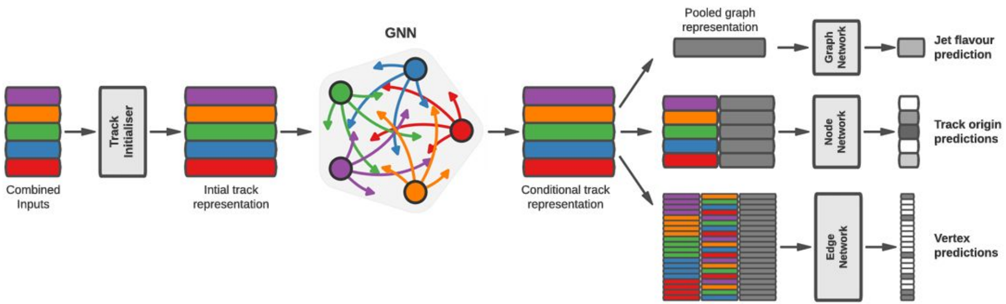


History of vertex fitting in ATLAS

GN1 & GN2

https://cds.cern.ch/record/2811135/

1. New state-of-the-art end-to-end neural networks (graph or transformer)
2. Auxiliary training objectives (track origin, track pair, jet flavour)
3. No usage of intermediate low-level algorithms

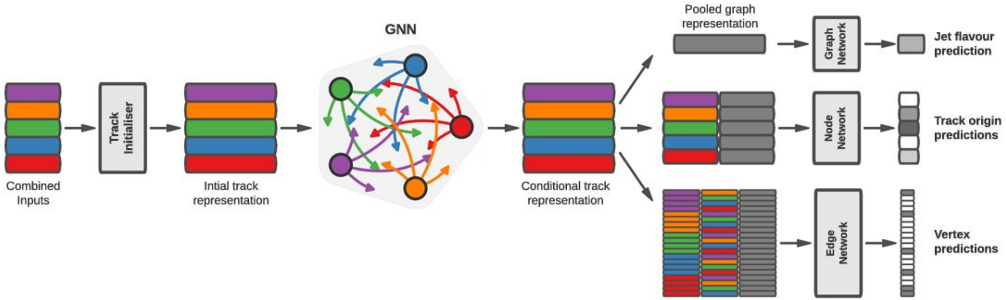


History of vertex fitting in ATLAS

GN1 & GN2

https://cds.cern.ch/record/2811135/

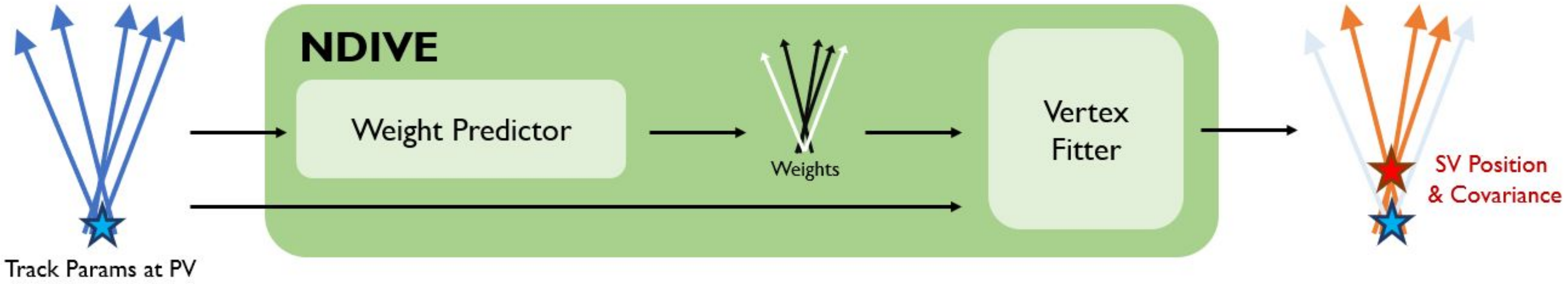
1. New state-of-the-art end-to-end neural networks (graph or transformer)
2. Auxiliary training objectives (track origin, track pair, jet flavour)
3. No usage of intermediate low-level algorithms
4. <u>No explicit secondary (or tertiary) vertex fitting!</u>



SV1 | JetFitter | DL1d | **GN1 & GN2**

Neural Differentiable Vertexing Layer (NDIVE)

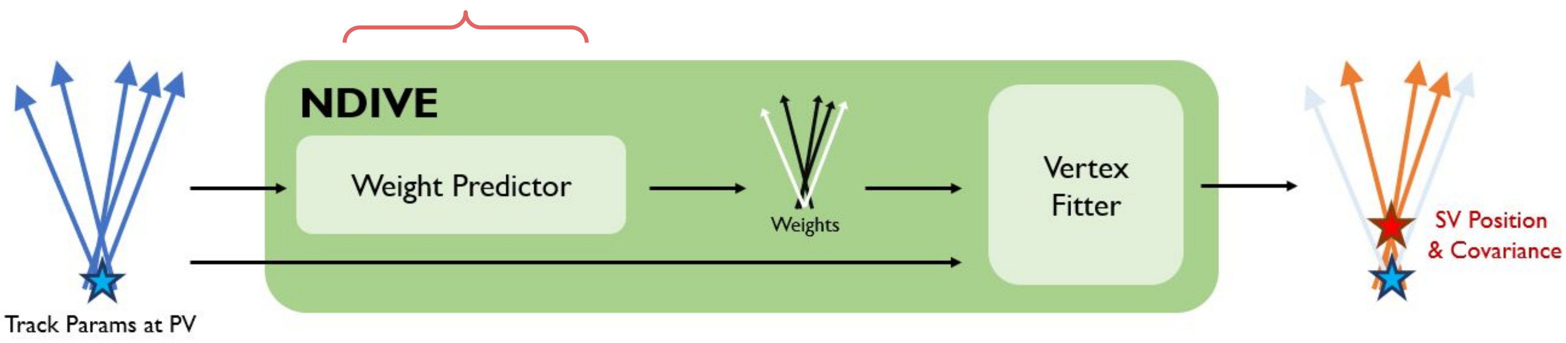
We propose to reintroduce explicit vertex reconstruction into end-to-end ML b-tagging algorithms via a vertexing layer that performs both **vertex finding** and **vertex fitting**



Neural Differentiable Vertexing Layer (NDIVE)

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

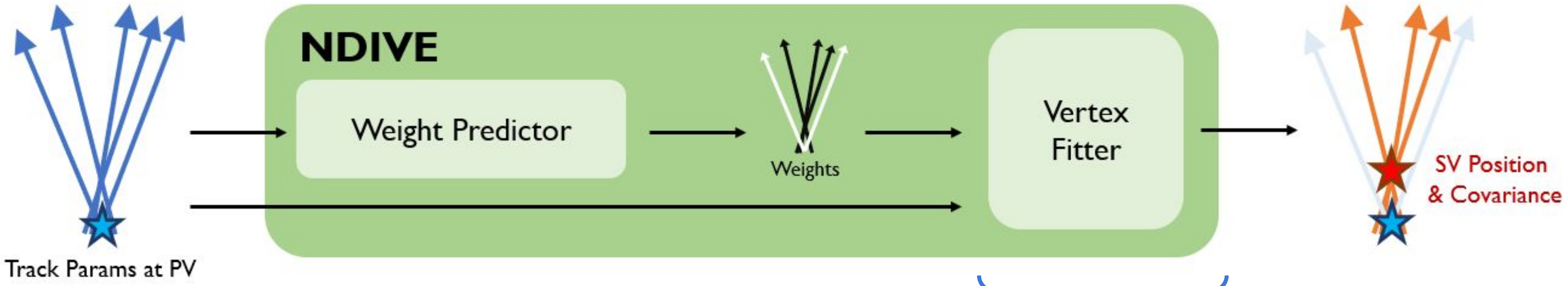
Neural network trained to assign weights to tracks



Neural Differentiable Vertexing Layer (NDIVE)

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

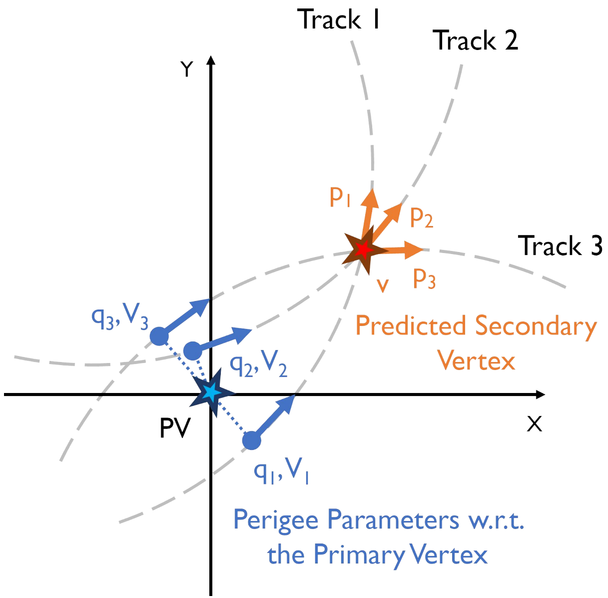
Neural network trained to assign weights to tracks



Vertex fitting algorithm w/ tracks and weights as inputs and no trainable parameters

Inclusive vertex fit formulation

- We perform an inclusive vertex fit with per-track weights, closely following Billoir's algorithm
- The values to be optimized are the vertex position **v** and track momentum at the vertex $\{p_i\}$: $\mathbf{x} = (\mathbf{v}, \{\mathbf{p}_i\})$
- The input data are the measured track parameters $\mathbf{q}_i = (d_0, z_0, \phi, \theta, \varrho)$ and their covariance matrix $\mathbf{V}_i$ (perigee representation)
- Additionally, a set of per-track weights $\mathbf{w}_i$ which determines how much each track contributes to the vertex fit



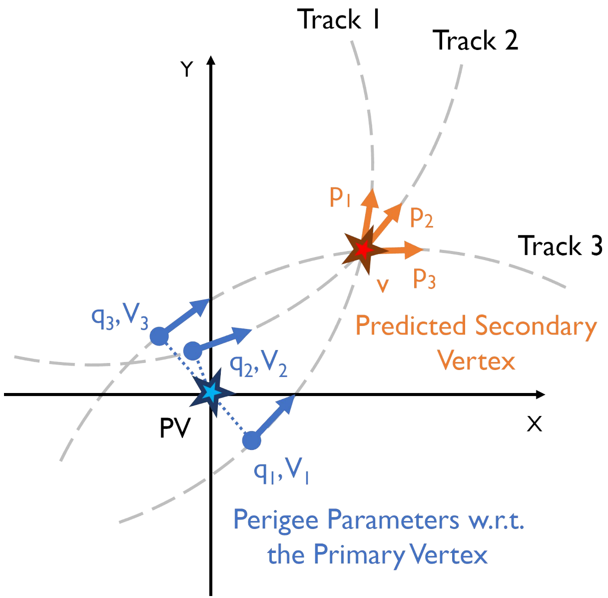
d_0	: signed transverse impact parameter
z_0	: longitudinal impact parameter
ϕ	: polar angle of trajectory
θ	: azimuthal angle of trajectory
ϱ	: signed curvature

Inclusive vertex fit formulation

- The following objective function S is minimized:

$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$

- We minimize the difference between the **measured track parameters** and the **track parameters obtained by extrapolating from the predicted vertex to the perigee**
- Measured and fit parameters are related via a track model $\mathbf{q}_{model,i} = \mathbf{h}_i(\mathbf{v}, \mathbf{p}_i)$ (e.g. a helical model of a curved track)

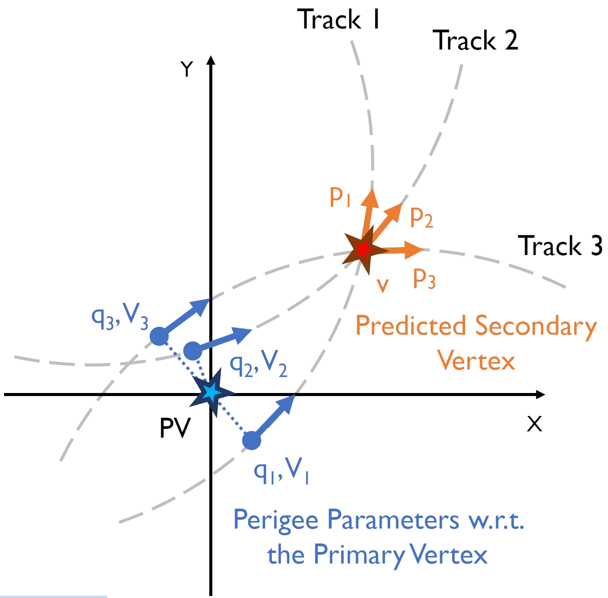


Inclusive vertex fit formulation

- The following objective function S is minimized:

$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$

- We minimize the difference between the **measured track parameters** and the **track parameters obtained by extrapolating from the predicted vertex to the perigee**
- Measured and fit parameters are related via a track model $\mathbf{q}_{\mathbf{model,i}} = \mathbf{h_i(v, p_i)}$ (e.g. a helical model of a curved track)



We need the derivatives of the fitted vertex v wrt the weights w_i to train downstream or upstream neural networks

Implicit differentiation

- 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 joint function of both **x** and **z**, where the output **z** of the layer is required to satisfy some constraint such as finding the root of an equation:

    Find z such that g(x,z) = 0

http://implicit-layers-tutorial.org

Implicit differentiation

- Explicit vs. Implicit layers:
  - An **explicit layer** with input **x** and ou
    of some explicit function **f**:

    z = f(x)

  - An **implicit layer** would *instead* be d
    where the output **z** of the layer is req
    finding the root of an equation:

    Find z such that g(x,z) = 0

Implicit layers have the advantage that we can use the *implicit function theorem* to directly compute gradients at the solution point of the equation, without having to store any intermediate variables

http://implicit-layers-tutorial.org

Implicit differentiation

- Explicit vs. Implicit layers:
  - An **explicit layer** with input **x** and ou
    of some explicit function **f**:

    $$z = f(x)$$

  - An **implicit layer** would *instead* be d
    where the output **z** of the layer is req
    finding the root of an equation:

    Find z such that g(x,z) = 0

Implicit layers have the advantage that we can use the *implicit function theorem* to directly compute gradients at the solution point of the equation, without having to store any intermediate variables

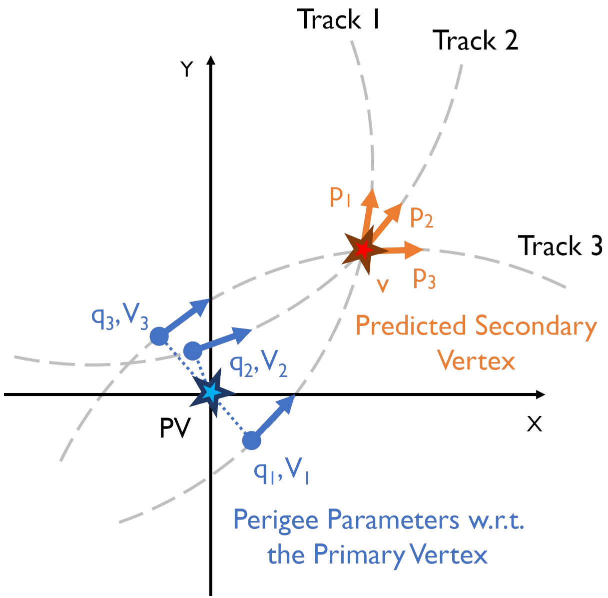
This improves memory consumption and often numerical accuracy (i.e. because we do not need to backpropagate through all fitting iterations)

http://implicit-layers-tutorial.org

Implicit differentiation

- The following objective function S is minimized:

$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$



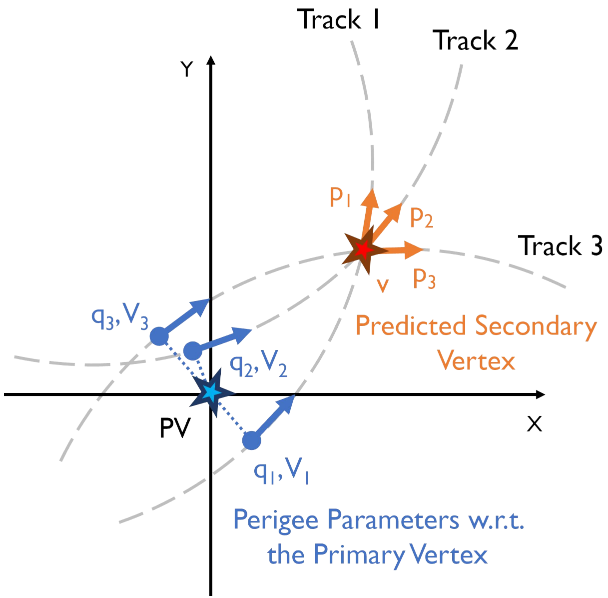
Implicit differentiation

- The following objective function S is minimized:

$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$

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

$$v^*(tracks, \vec{w}_{\text{tracks}}) \equiv \underset{v}{\text{argmin}}\ \chi^2(v; \text{tracks}, \vec{w}_{\text{tracks}})$$



Implicit differentiation

- The following objective function S is minimized:

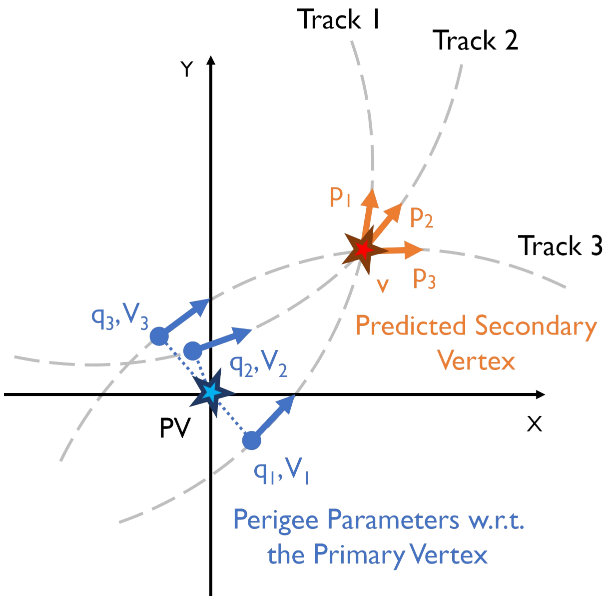
$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$

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

$$v^*(tracks, \vec{w}_{\text{tracks}}) \equiv \underset{v}{\operatorname{argmin}}\, \chi^2(v; \text{tracks}, \vec{w}_{\text{tracks}})$$

- Note that at the minimum of the X^2 we have:

$$\left.\frac{\partial \chi^2}{\partial v}\right|_{v=v^*} = 0 \longrightarrow \frac{\mathrm{d}}{\mathrm{d}w}\left(\left.\frac{\partial \chi^2}{\partial v}\right|_{v=v^*}\right) = \frac{\mathrm{d}}{\mathrm{d}w}(0) = 0$$



Implicit differentiation

- The following objective function S is minimized:

$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$

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

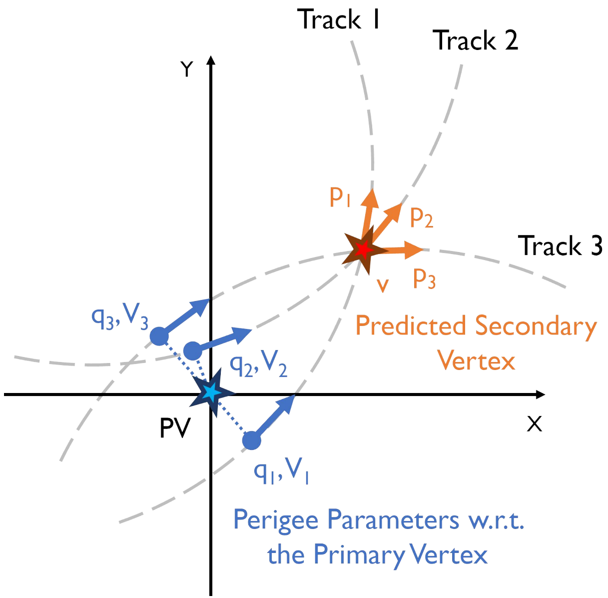
$$v^*(tracks, \vec{w}_{\text{tracks}}) \equiv \underset{v}{\text{argmin}}\ \chi^2(v; \text{tracks}, \vec{w}_{\text{tracks}})$$

- Note that at the minimum of the X^2 we have:

$$\left.\frac{\partial \chi^2}{\partial v}\right|_{v=v^*} = 0 \longrightarrow \frac{\mathrm{d}}{\mathrm{d}w}\left(\left.\frac{\partial \chi^2}{\partial v}\right|_{v=v^*}\right) = \frac{\mathrm{d}}{\mathrm{d}w}(0) = 0$$

- Expand derivative:

$$\left.\frac{\partial^2 \chi^2}{\partial v^2}\right|_{v=v^*} \frac{\mathrm{d}v^*}{\mathrm{d}w} + \left.\frac{\partial^2 \chi^2}{\partial v \partial w}\right|_{v=v^*} = 0 \longrightarrow \frac{\mathrm{d}v^*}{\mathrm{d}w} = -\left(\frac{\partial^2 \chi^2}{\partial v^2}\right)^{-1} \left.\frac{\partial^2 \chi^2}{\partial v \partial w}\right|_{v=v^*}$$



Implicit differentiation

- The following objective function S is minimized:

$$S = X^2 = \Sigma\, w_i\, (q_i - h_i(v, p_i))^T\, V_i^{-1}\, (q_i - h_i(v, p_i))$$

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

$$v^*(tracks, \vec{w}_{\text{tracks}}) \equiv \underset{v}{\text{argmin}}\ \chi^2(v; \text{tracks}, \vec{w}$$

- Note that at the minimum of the X^2 we h

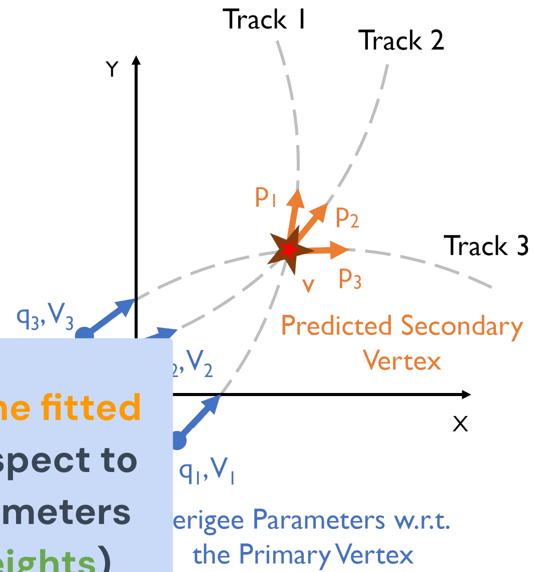
$$\left.\frac{\partial \chi^2}{\partial v}\right|_{v=v^*} = 0 \longrightarrow \frac{\mathrm{d}}{\mathrm{d}w}\left(\left.\frac{\partial \chi^2}{\partial v}\right|_{v=v^*}\right) =$$

- Expand derivative:

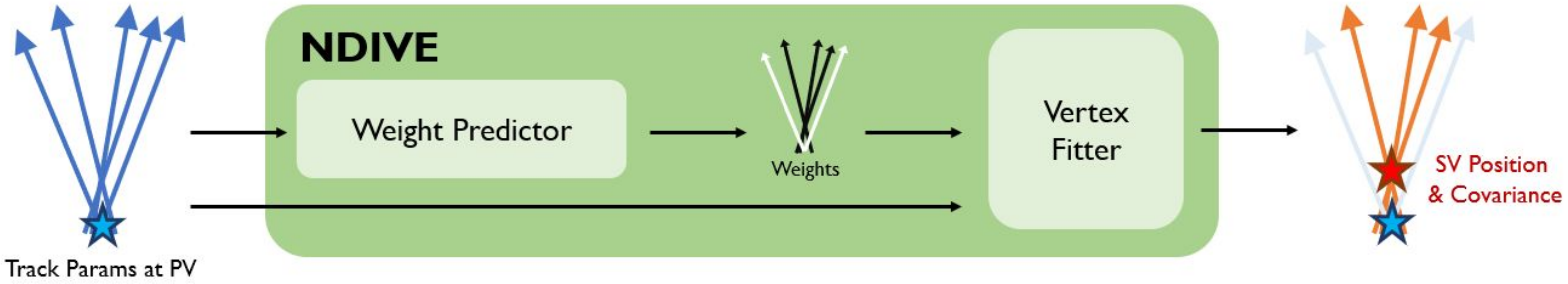
$$\left.\frac{\partial^2 \chi^2}{\partial v^2}\right|_{v=v^*} \frac{\mathrm{d}v^*}{\mathrm{d}w} + \left.\frac{\partial^2 \chi^2}{\partial v \partial w}\right|_{v=v^*} = 0 \longrightarrow$$

Derivative of the fitted vertex with respect to the input parameters (the track weights)

$$\frac{\mathrm{d}v^*}{\mathrm{d}w} = -\left(\frac{\partial^2 \chi^2}{\partial v^2}\right)^{-1} \left.\frac{\partial^2 \chi^2}{\partial v \partial w}\right|_{v=v^*}$$

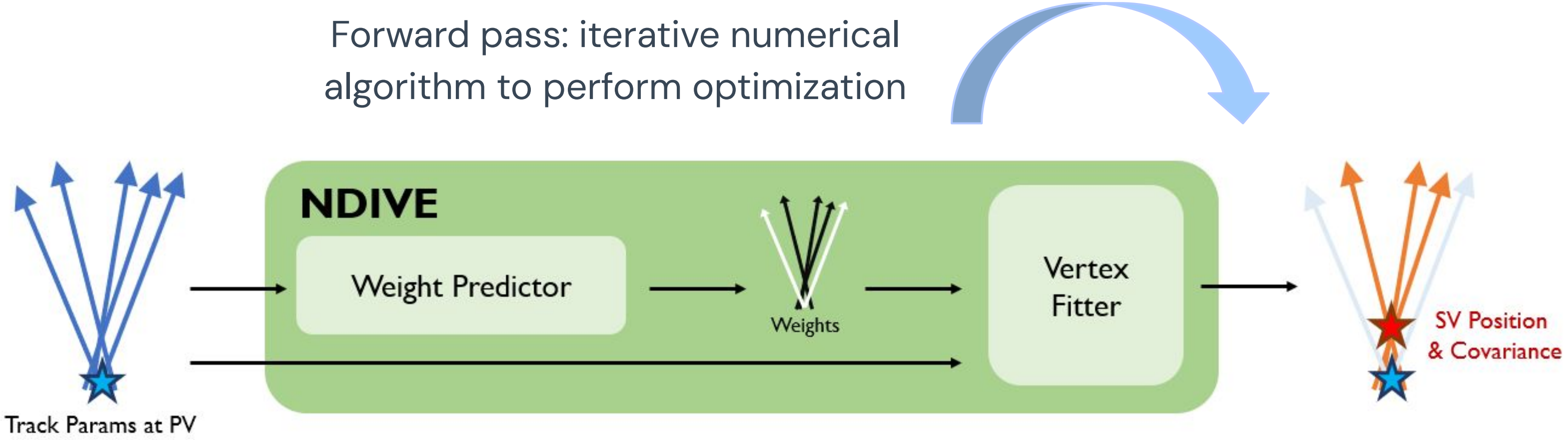


Implicit differentiation



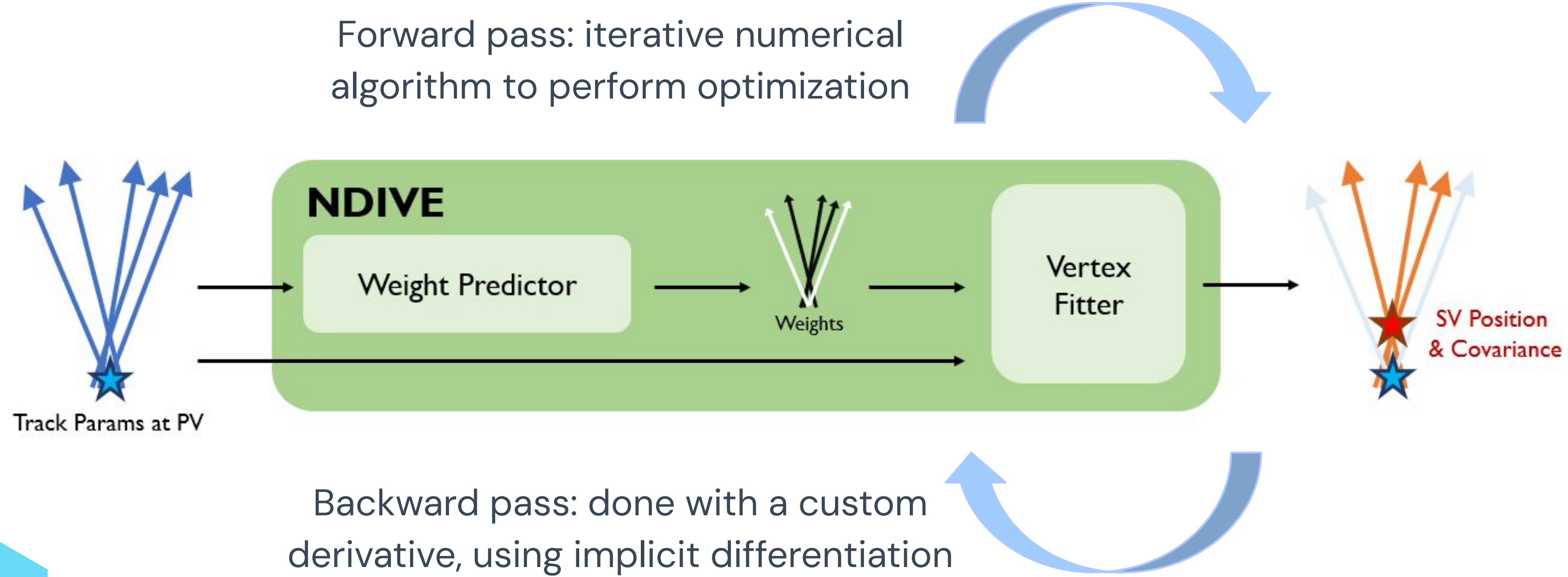
Implicit differentiation

Forward pass: iterative numerical algorithm to perform optimization



Implicit differentiation

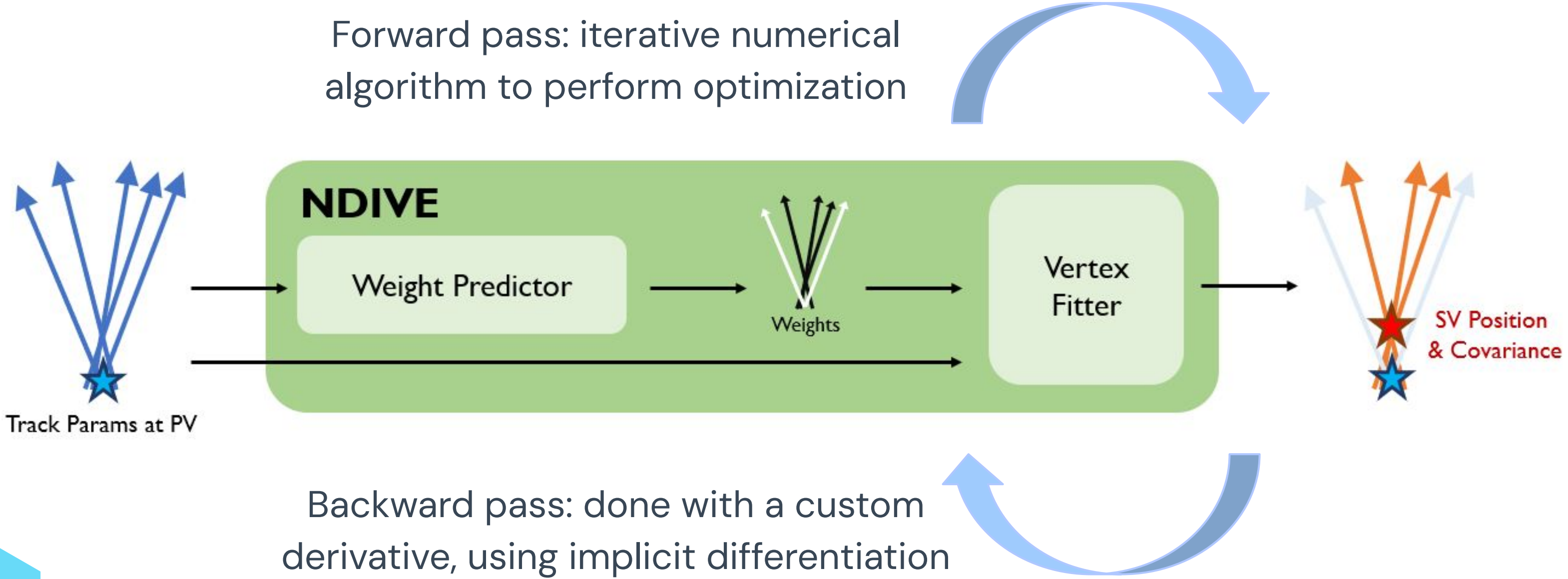
Forward pass: iterative numerical algorithm to perform optimization



Backward pass: done with a custom derivative, using implicit differentiation

Implicit differentiation

Forward pass: iterative numerical algorithm to perform optimization



Backward pass: done with a custom derivative, using implicit differentiation

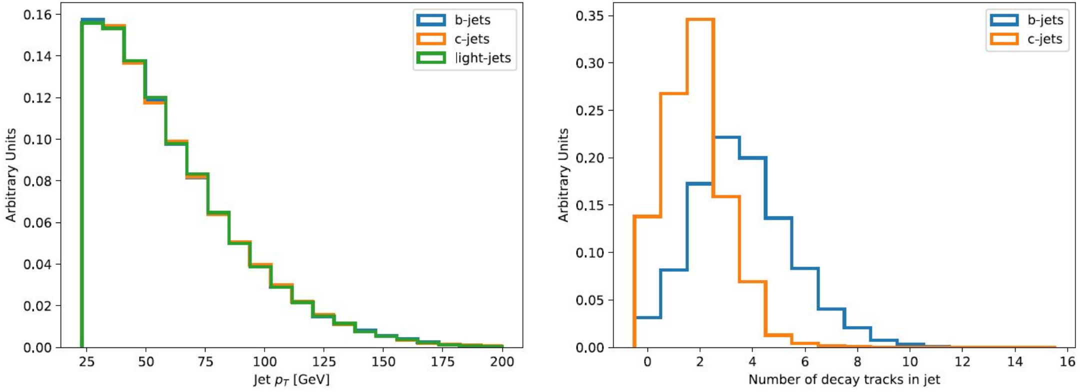
Our differentiable vertex fitting layer is now ready to be inserted into any neural network! (i.e. integrating domain knowledge)

Samples & Input Variables

- Top pair production from proton-proton collisions simulated at $\sqrt{s}$ = 14 TeV
- Generated with Pythia8 with ATLAS detector parameterization in Delphes
- 500k training jets, 180k validation, 180k testing

Training features:

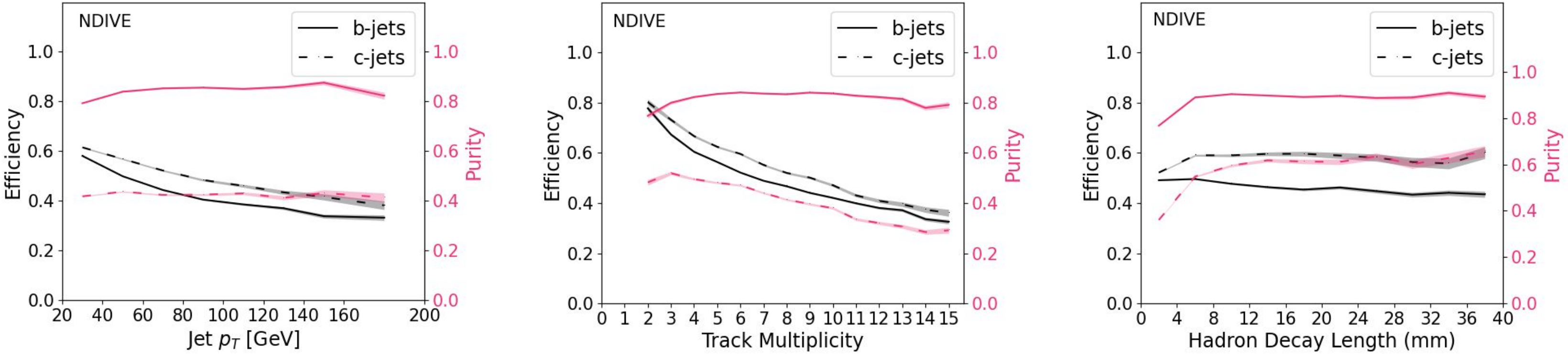
- Track perigee parameters and their errors
- Signed d_0 and z_0 significances
- log (track pT / jet pT)
- ΔR (track, jet)



https://zenodo.org/records/4044628

Track selection performance

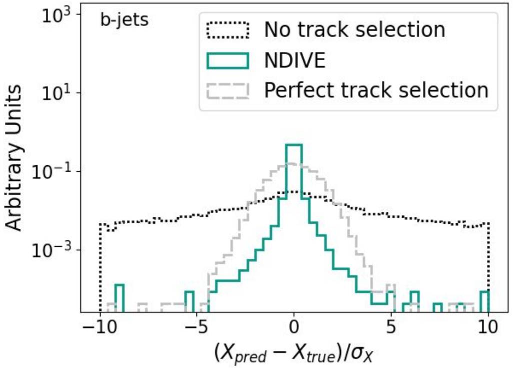
- Efficiency: number of decay tracks selected* over all decay tracks
- Purity: number of decay tracks selected* over all selected tracks

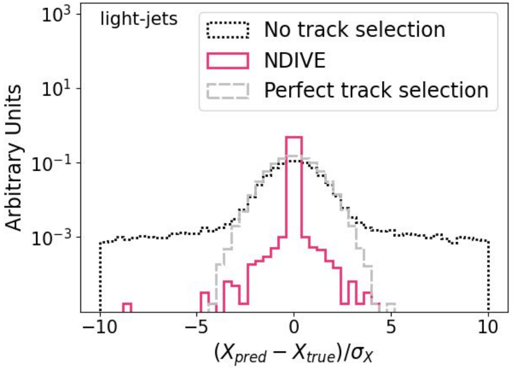


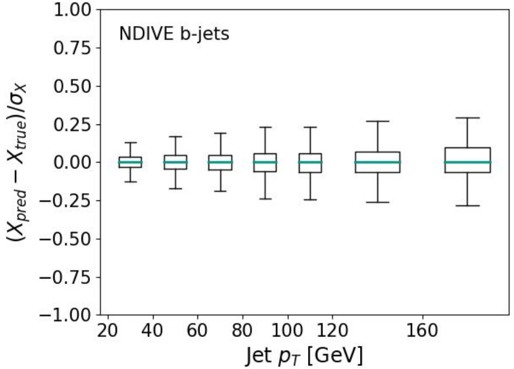
* "selected tracks" => per-track weights normalized by maximum weight in each jet and required to be above > 0.5

Vertex reconstruction performance

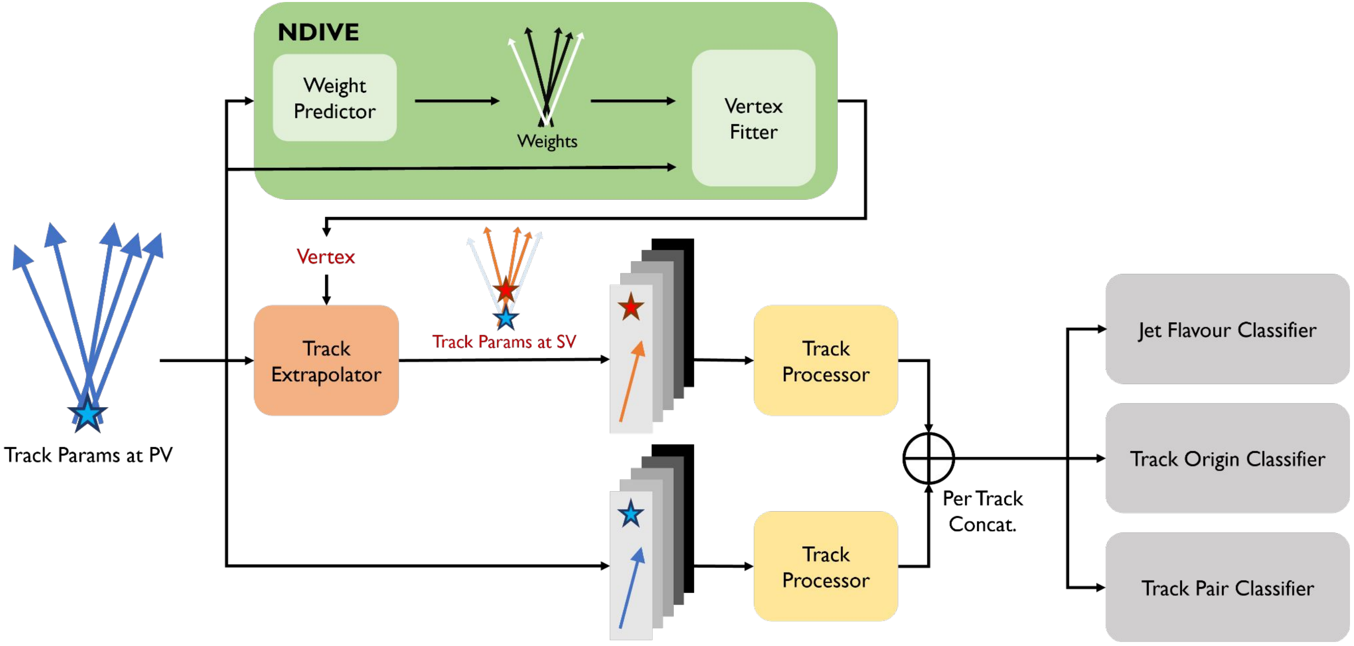
- “Perfect track selection” => weights set to 0 or 1 based on true origin of track
- “No track selection” => all tracks in the jet are used in the fit with weight 1







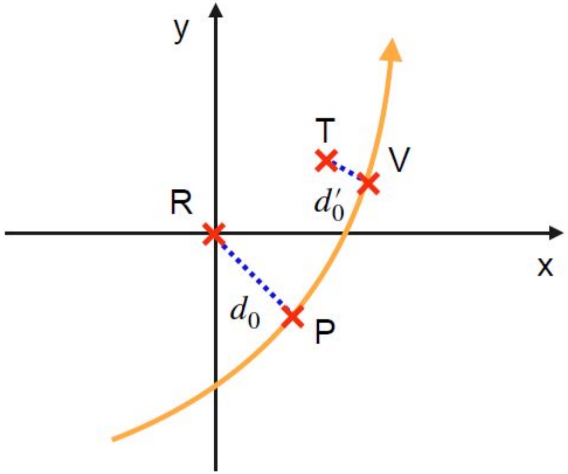
Integrating vertex fitting into a flavour tagging model (FTAG+NDIVE)



Track Extrapolator

- Measured track parameters are typically expressed at a given point along the trajectory wrt a reference point (commonly, the primary vertex)
- Track Extrapolator module incorporates knowledge of expected track geometry to extrapolate each track to the point of closest approach to the vertex predicted by NDIVE, enabling us to construct an alternative representation of the tracks
- Implemented with JAX's autodiff

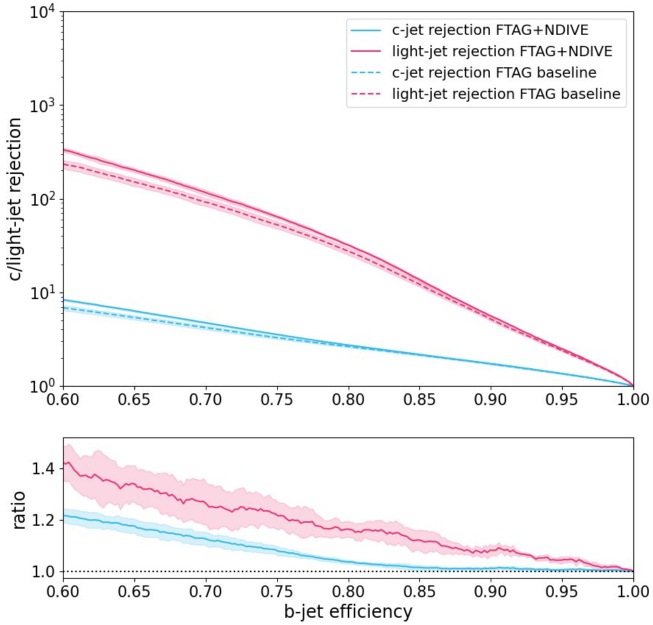
$$x_V = x_P + d_0 \cos\left(\phi + \frac{\pi}{2}\right) + \rho\left[\cos\left(\phi_V + \frac{\pi}{2}\right) - \cos\left(\phi + \frac{\pi}{2}\right)\right]$$
$$y_V = y_P + d_0 \sin\left(\phi + \frac{\pi}{2}\right) + \rho\left[\sin\left(\phi_V + \frac{\pi}{2}\right) - \sin\left(\phi + \frac{\pi}{2}\right)\right]$$
$$z_V = z_P + z_0 - \frac{\rho}{\tan(\theta)}\left[\phi_V - \phi\right]$$



ROC curves

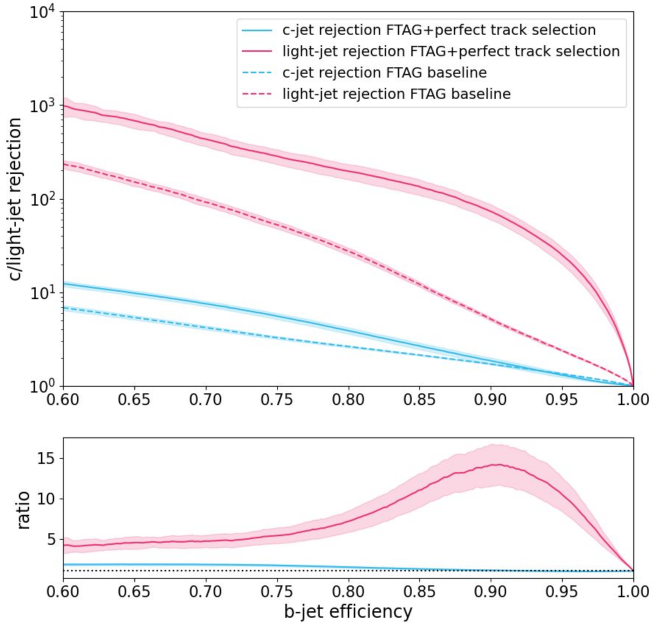
$$D_b = \log \frac{p_b}{(1-f_c)p_l + f_c p_c}$$

$f_c = 0.05$



Future Work

- We don't claim that this is the best way to integrate differentiable vertex fitting into flavour tagging
- These developments are generic, applicable to other vertex fitting algorithms and other schemes for integrating the vertex information into neural network architecture
- To illustrate possible improvements, we show the potential for huge gains given an ideal scenario with perfect track selection



Conclusion

- **We introduce NDIVE, the first differentiable vertex fitting algorithm**

- **We formulate vertex fitting as an optimization problem**
  - We define gradients of the optimized vertex through implicit differentiation
  - Can be passed to upstream or downstream networks for training

- **This is an application of differential programming for integrating physics knowledge into neural networks**
  - NDIVE can be integrated into end-to-end b-tagging algorithms, explicitly reintroducing vertex reconstruction geometry
  - Part of the larger effort to apply differentiable programming in HEP

Backup

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 hi expanded at an estimate of the vertex position and track momenta: $\mathbf{q}_i \approx \mathbf{A}_i\mathbf{v} + \mathbf{B}_i\mathbf{p}_i + \mathbf{c}_i$

$$\mathbf{A}_i = \left.\frac{\partial \mathbf{h}_i}{\partial \mathbf{v}}\right|_{e_0}$$

- Iterate fit until convergence, expanding the functions $\mathbf{h}_i$ around the new expansion point each time:

$$\mathbf{B}_i = \left.\frac{\partial \mathbf{h}_i}{\partial \mathbf{p}_i}\right|_{e_0}$$

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

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

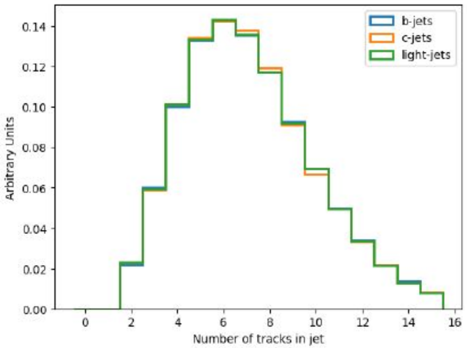
$$\begin{aligned} \mathbf{G}_i &= \mathbf{V}_i^{-1} \\ \mathbf{D}_i &= \mathbf{A}_i^T \mathbf{G}_i \mathbf{B}_i \\ \mathrm{D}_0 &= \sum_{i=1}^{N} \mathbf{A}_i^T \mathbf{G}_i \mathbf{A}_i \\ \mathbf{W}_i^{-1} &= \mathbf{B}_i^T \mathbf{G}_i \mathbf{B}_i \end{aligned}$$

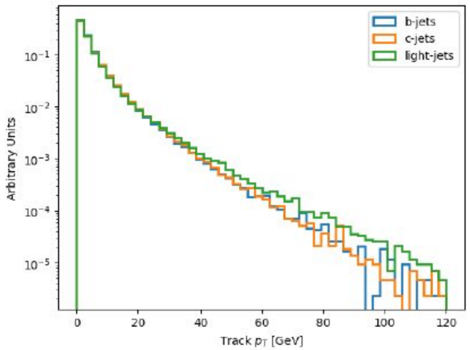
- Afterwards we rewrote the track parameters $\hat{\mathbf{q}}_i = \mathbf{h}_i(\hat{\mathbf{v}}, \hat{\mathbf{p}}_i)$.
- The χ^2 statistic of the fit is then:

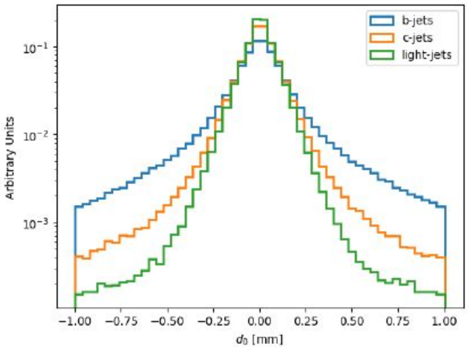
$$\mathbf{C} = \left(\mathrm{D}_0 - \sum_{i=1}^{N} \mathbf{D}_i \mathbf{W}_i \mathbf{D}_i^T\right)^{-1}$$

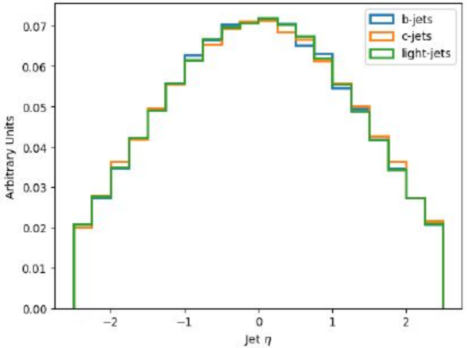
$$\chi^2 = \sum_{i=1}^{N} (\mathbf{q}_i - \hat{\mathbf{q}}_i)^T \mathbf{G}_i (\mathbf{q}_i - \hat{\mathbf{q}}_i)$$

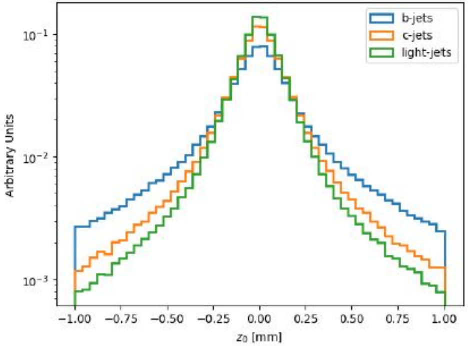
Dataset


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

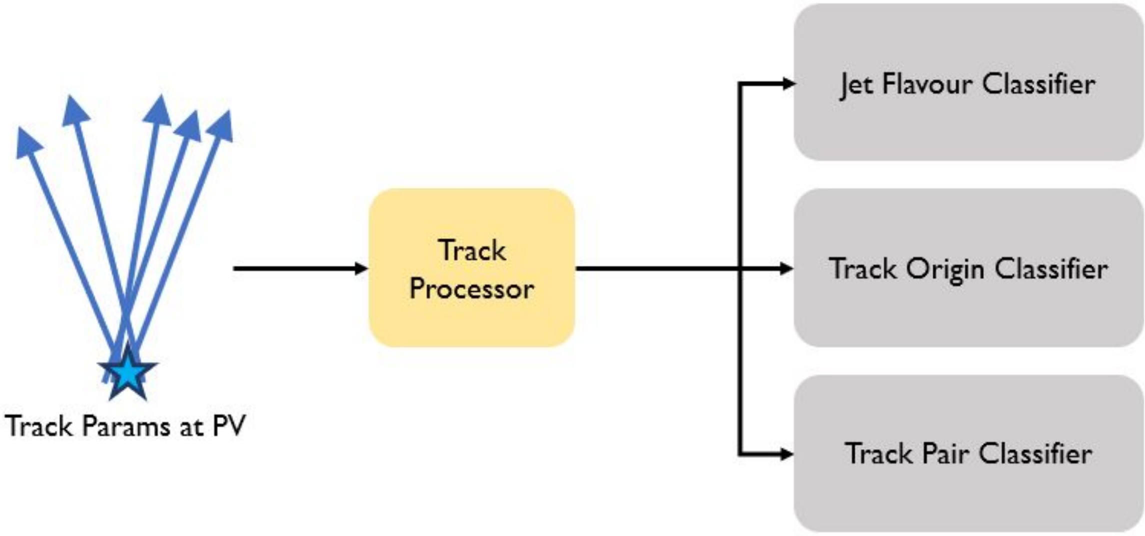

Arbitrary Units
Track p_T [GeV]
b-jets
c-jets
light-jets


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


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


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

FTAG baseline



CREDITS: This presentation template was created by **Slidesgo**, and includes icons by **Flaticon**, and infographics & images by **Freepik**