Vertexing using SaltModel: Experiments with ColliderML dataset

Diptaparna Biswas

Universität Siegen

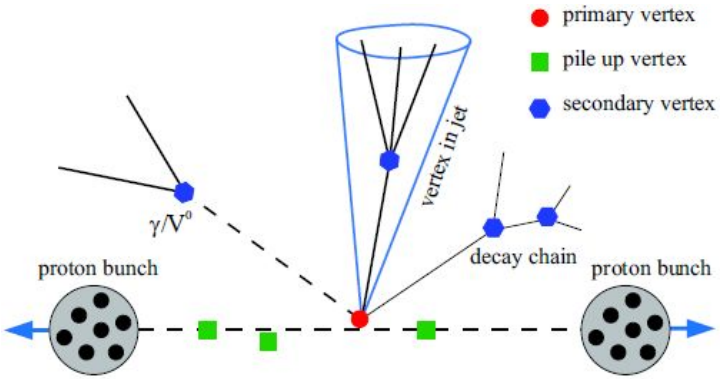
HEP Experimental High Energy Physics

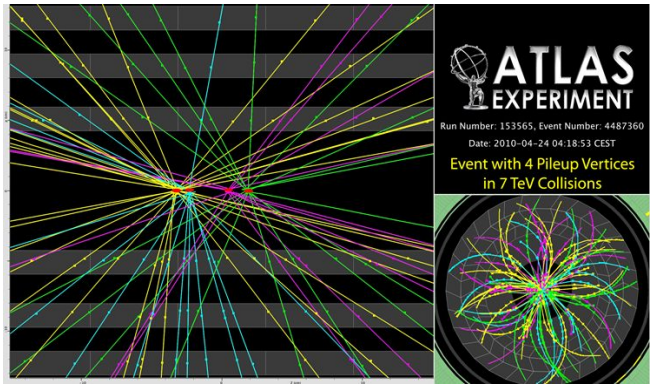
CPPS Center for Particle Physics Siegen

TrackOpt meeting

7th September 2026

Recap: Primary vertexing vs. secondary vertexing





- For our purpose, we call both “primary vertex” and “pileup vertex” inclusively as the primary vertices.
- Each track in an event is associated to one of these primary vertices.
  - ➢ This includes tracks originating from the secondary vertices.

The ColliderML dataset

- Large-scale simulated dataset intended for algorithm research.
- Modelled after a realistic detector for HL-LHC, consists of information from different sub-detectors:
    1. particles (truth-level)
    2. tracker_hits (detector-level)
    3. calo_hits (detector-level)
    4. tracks (reconstruction-level)
- MC events are generated for:
    - ➢ Different Physics process (e.g., ttbar, ggf, dihiggs)
    - ➢ Different pileup conditions (pu0 = no pileup, pu200 = HL-LHC pileup)

Using ColliderML dataset for vertexing research

Field	Description
event_id	Unique event identifier
track_id	Unique track identifier within event
majority_particle_id	Truth particle with most hits on this track
d0	Transverse impact parameter (mm)
z0	Longitudinal impact parameter (mm)
phi	Azimuthal angle (radians)
theta	Polar angle (radians)
qop	Charge divided by momentum (e/GeV)
hit_ids	List of tracker hit IDs on this track

Group by (vx,vy,vz)

Field	Description
event_id	Unique event identifier
particle_id	Unique particle ID within event
pdg_id	PDG particle code (11=electron, 13=muon, 211=pion, etc.)
mass	Particle rest mass (GeV/c²)
energy	Particle total energy (GeV)
charge	Electric charge (units of e)
px, py, pz	Momentum components (GeV/c)
vx, vy, vz	Vertex position (mm)
time	Production time (ns)
perigee_d0, z0	Perigee impact parameters (mm)
primary	Whether particle is primary
vertex_primary	Primary vertex index (1=hard scatter)

Using ColliderML dataset for vertexing research

```
Event number: 0
            vx         vy          vz
640  23.791132 -5.039636  130.088516
643  23.791132 -5.039636  130.088516
------------------------------
            vx         vy          vz
551  28.624298 -22.364754 -344.434326
575  28.624298 -22.364754 -344.434326
------------------------------
==============================
Event number: 1
            vx         vy         vz
454  34.311501 -43.547958 -616.7771
472  34.311501 -43.547958 -616.7771
------------------------------
==============================
Event number: 2
             vx         vy           vz
1319 -804.349915 -15.901301 -1276.860107
1330 -804.349915 -15.901301 -1276.860107
------------------------------
            vx         vy         vz
882  23.057499  29.151842  33.008316
921  23.057499  29.151842  33.008316
------------------------------
==============================
```

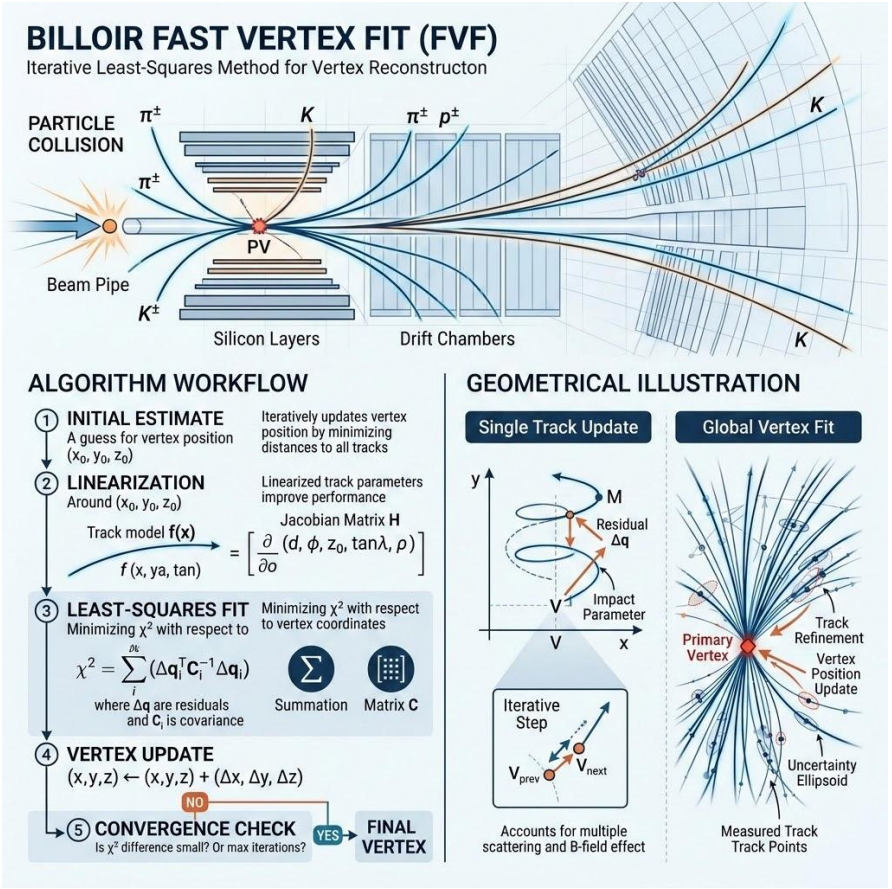
primary == 0

```
Event number: 3
==============================
Event number: 4
            vx         vy          vz
445 -8.490943 -37.420902  154.639587
448 -8.490943 -37.420902  154.639587
------------------------------
              vx          vy          vz
1167  556.873718  594.055725 -748.714478
1168  556.873718  594.055725 -748.714478
------------------------------
==============================
Event number: 5
             vx         vy         vz
1990 -22.302839  31.07262  31.330093
2022 -22.302839  31.07262  31.330093
------------------------------
             vx        vy         vz
2115 -1.526555  0.612422  36.911095
2248 -1.526555  0.612422  36.911095
------------------------------
             vx         vy         vz
1119  23.798487 -0.784967 -40.47855
1141  23.798487 -0.784967 -40.47855
------------------------------
==============================
```

```
Event number: 6
             vx         vy          vz
351 -23.370489 -31.059721  170.855377
355 -23.370489 -31.059721  170.855377
------------------------------
             vx         vy          vz
1440  25.479441  26.947424  112.898849
1444  25.479441  26.947424  112.898849
------------------------------
==============================
Event number: 7
             vx         vy          vz
304 -25.171206 -25.622263 -170.397568
311 -25.171206 -25.622263 -170.397568
------------------------------
             vx         vy          vz
1736  24.66535  25.430759  291.385071
1737  24.66535  25.430759  291.385071
------------------------------
             vx         vy          vz
1735  40.272182  42.37019 -529.946106
1738  40.272182  42.37019 -529.946106
1741  40.272182  42.37019 -529.946106
------------------------------
              vx         vy           vz
1497  220.518051  522.27655  1149.442993
1950  220.518051  522.27655  1149.442993
------------------------------
==============================
Event number: 8
==============================
Event number: 9
==============================
```

Billoir fit in Python

- Invented by Pierre Billoir.
- Standard technique for *vertexing*:
  - ➢ Whether a set of tracks is originated from the same vertex.
  - ➢ Calculate the position of this vertex.
  - ➢ Correct the tracks momenta.
- The current implementation (from Vadim) is in ROOT, and not-so-easy to use with the ColliderML dataset.



Billoir fit in Python: ROOT vs numpy

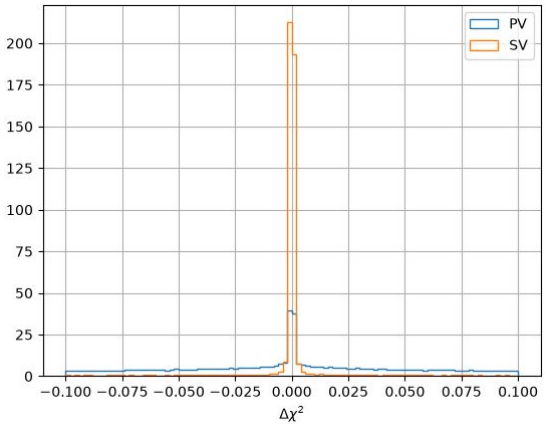
Implemented in ~150 lines of numpy code. The implementation is slow, but I could readily translate it to numba and pytorch using LLMs, which are significantly faster.

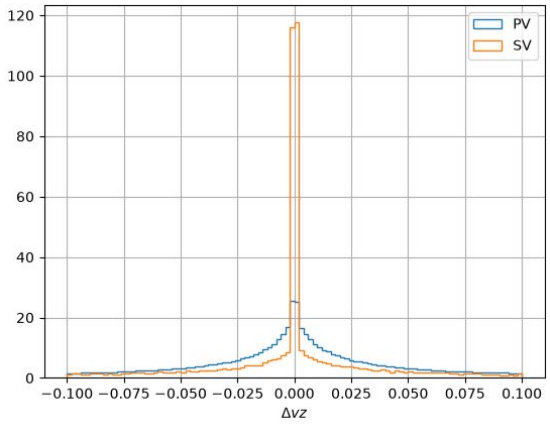
t_vx	vadim_vx	dipta_vx
-0.335286	-0.502742	-0.502699
-13.725449	-13.852321	-13.851948
11.359540	11.833704	11.833023
1.530382	1.521566	1.521566
-32.385628	-32.434696	-32.434685
-1.512289	-1.701477	-1.701271
-3.292901	-3.282216	-3.282206
-3.292901	-3.471185	-3.471200
-3.292901	-3.489085	-3.489002
-3.304003	-3.350973	-3.350984

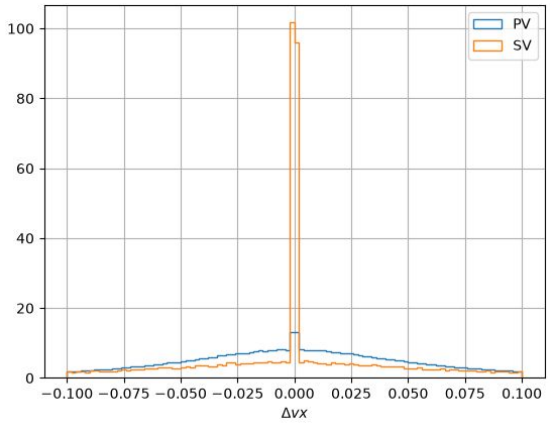
t_vy	vadim_vy	dipta_vy
-0.153082	-0.304685	-0.304674
-29.455383	-29.655619	-29.654898
3.374577	3.507995	3.507789
0.813375	0.852308	0.852308
17.882717	17.917532	17.917527
-2.867919	-3.218182	-3.217758
7.530176	7.467003	7.466973
7.530176	8.032998	8.033029
7.530176	7.981745	7.981614
7.553091	7.653723	7.653755

t_vz	vadim_vz	dipta_vz
-7.037555	-7.451469	-7.451402
-118.613693	-119.481049	-119.478010
-19.370573	-20.469828	-20.468259
40.106155	40.079216	40.079215
-347.082581	-347.721069	-347.721056
-11.381413	-11.699656	-11.699209
-101.123314	-100.693695	-100.693542
-101.123314	-104.247940	-104.248151
-101.123314	-104.191864	-104.191009
-101.282906	-102.079597	-102.079826

Billoir fit in Python: ROOT vs numpy

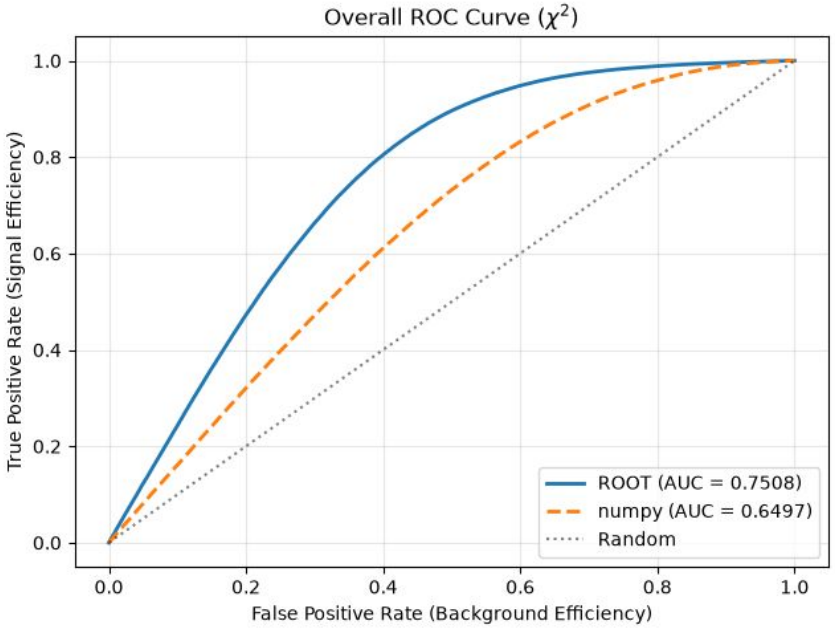


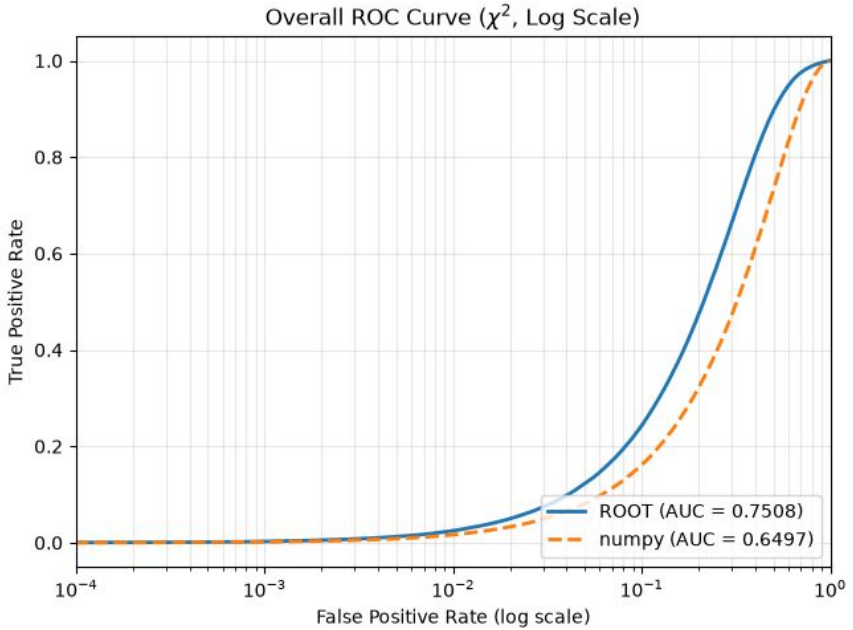




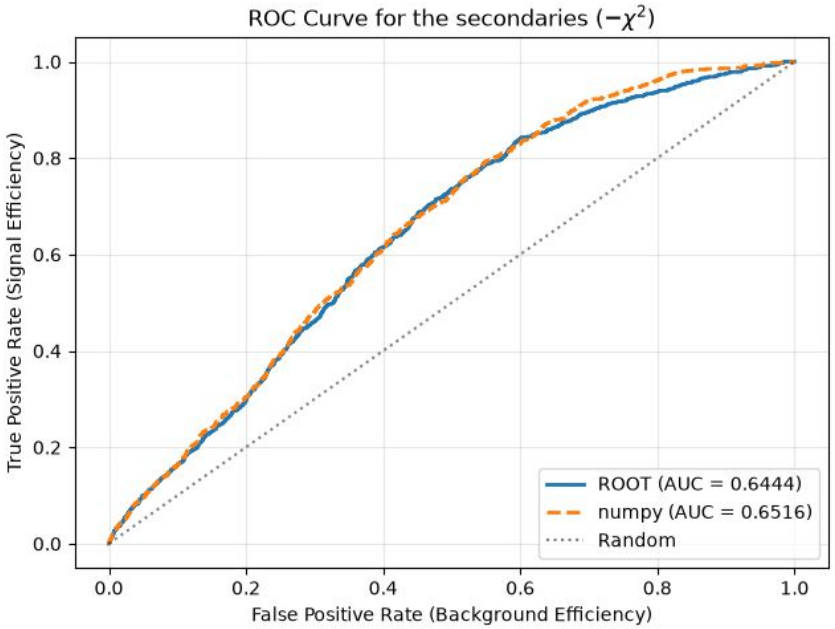

```
PV: t_vx**2 + t_vy**2 < 0.01
SV: t_vx**2 + t_vy**2 > 0.01
```

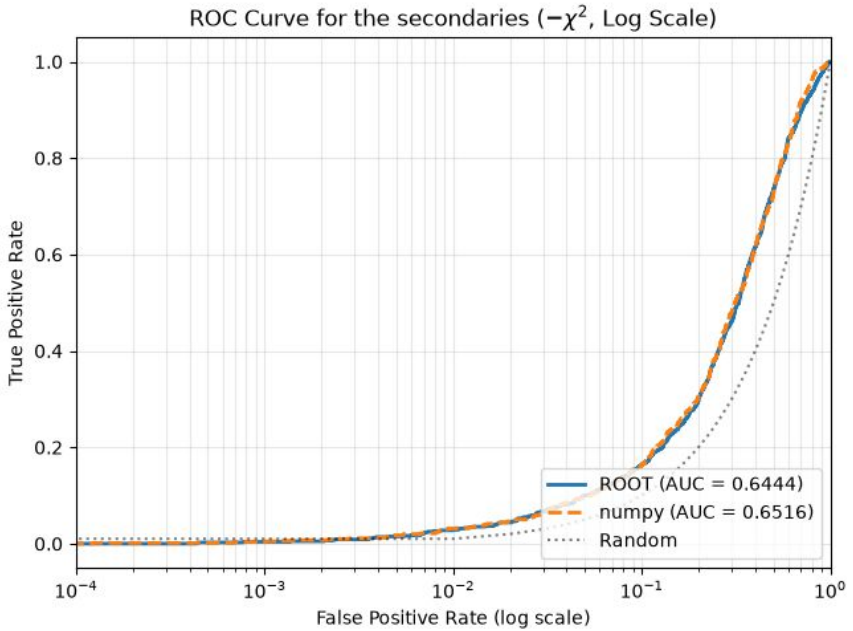
Billoir fit in python: ROOT vs numpy



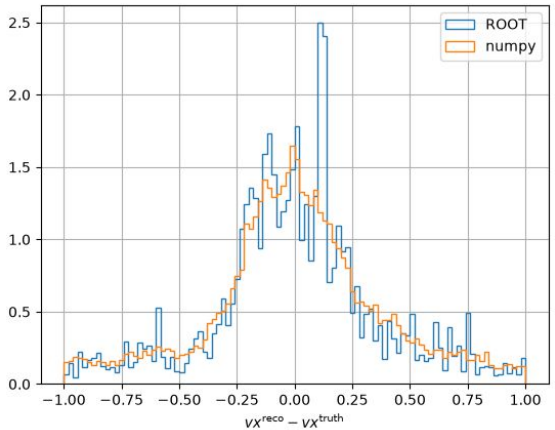


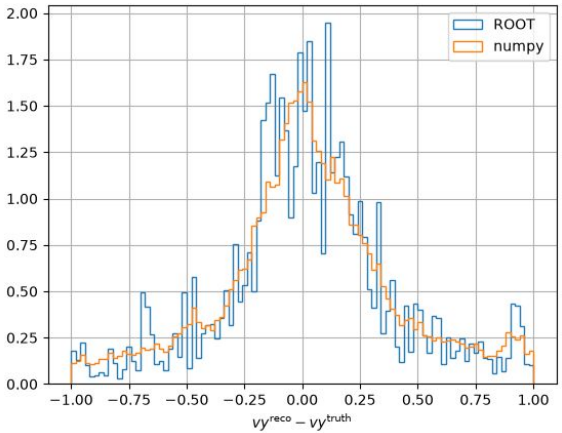
Billoir fit in python: ROOT vs numpy

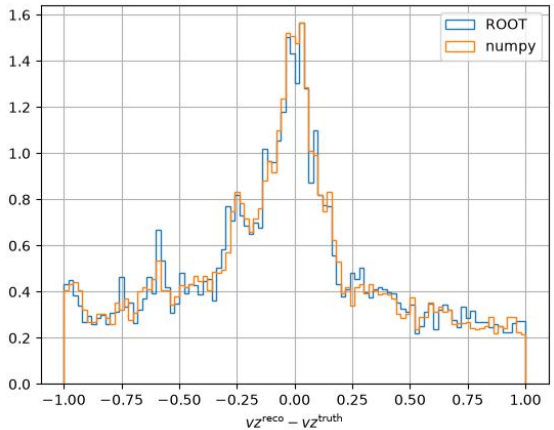




Billoir fit in python: ROOT vs numpy



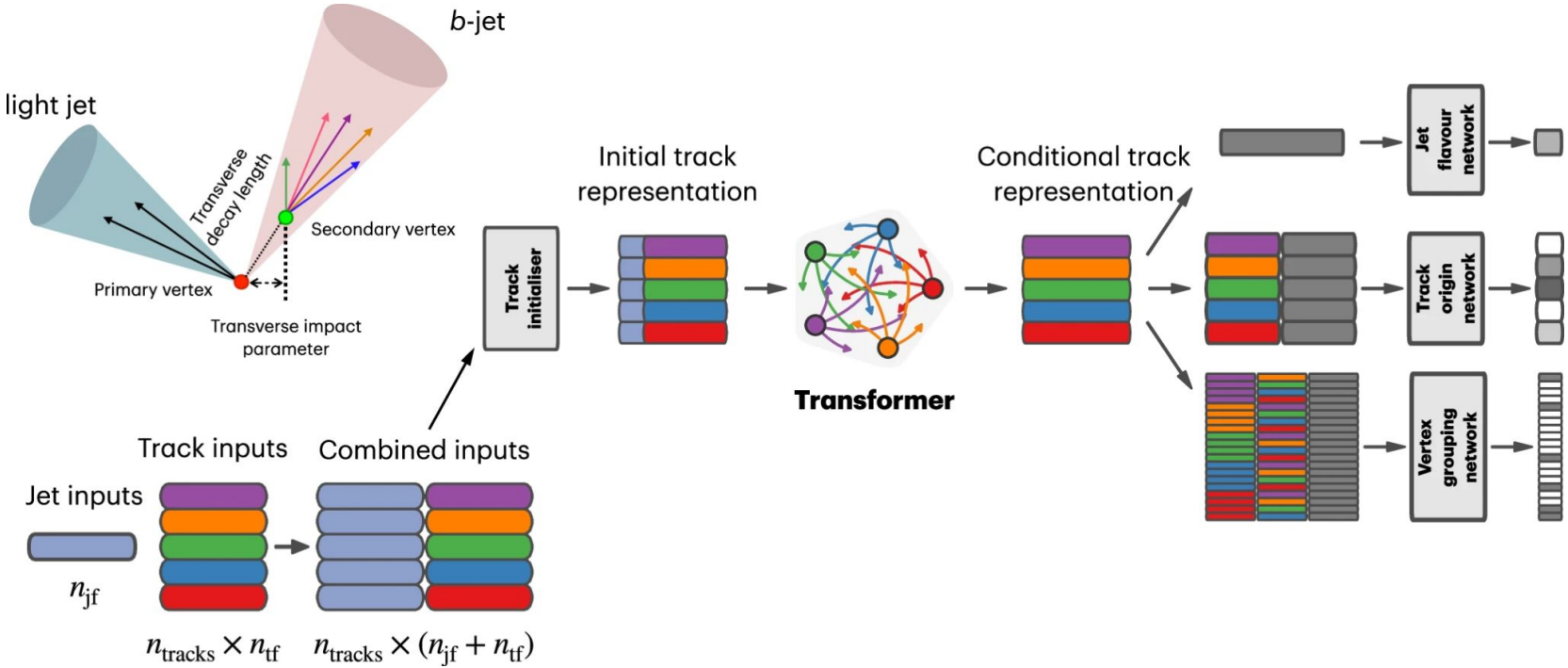




Using SVs only

`SV: t_vx**2 + t_vy**2 > 0.01`

Vertexing using SaltModel



The GN2 flavour tagger implemented as a SaltModel

What is *salt* and `SaltModel`?

Salt is the state-of-the-art ML framework maintained by ATLAS Jet Flavour Tagging group, intended for building end-to-end unified transformer-based models.

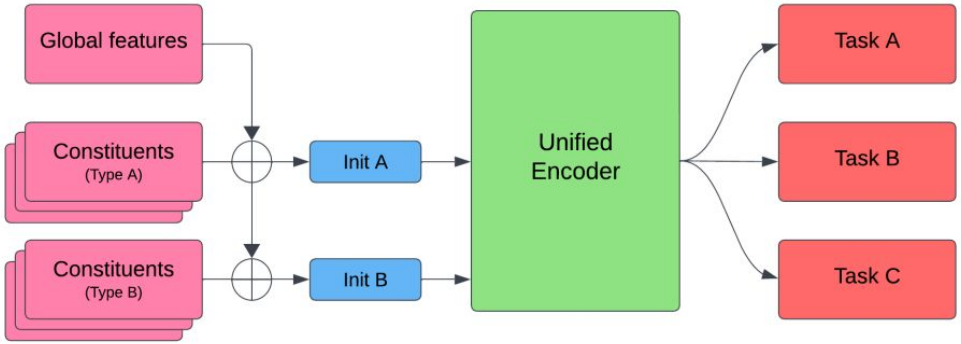


Figure 1: This diagram illustrates the flow of information within a generic model trained using `Salt`. In this example, global object features are provided alongside two types of constituents, "Type A" and "Type B", which represent different input modalities such as charged particle trajectories or calorimeter energy depositions. The model is configured with three training objectives, each of which may relate to the global object or one of the constituent modalities. Concatenation is denoted by $\oplus$.

Reference: Salt: Multimodal Multitask Machine Learning for High Energy Physics

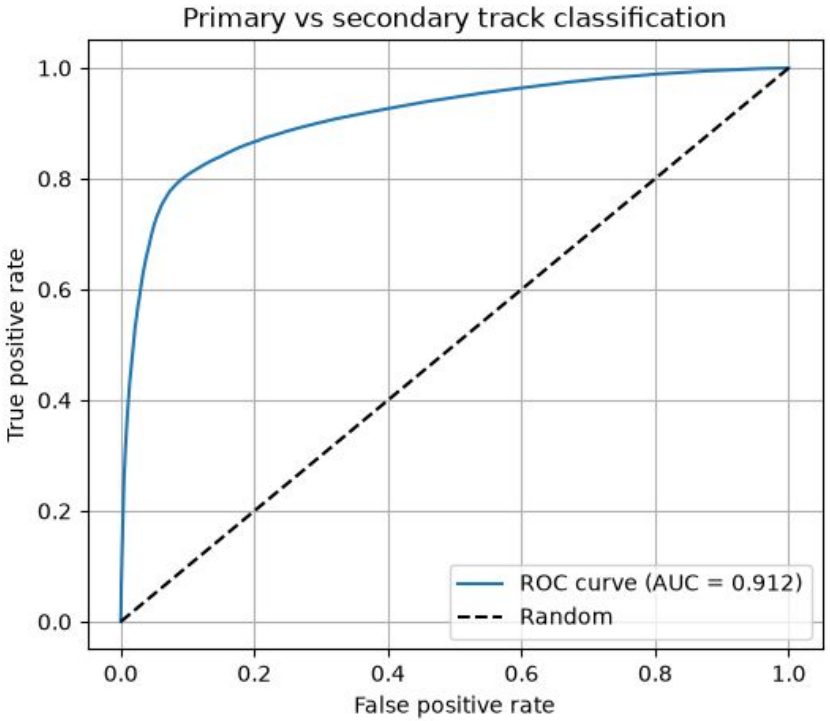
Jackson Barr [1], Diptaparna Biswas [2], Maxence Draguet [3], Philipp Gadow [4], Emil Haines [1], Osama Karkout [5], Dmitrii Kobylianskii [6], Wei Sheng Lai [1], Matthew Leigh [7], Nicholas Luongo [10], Ivan Oleksiyuk [7], Nikita Pond [1], Sébastien Rettie [4], Andrius Vaitkus [1], Samuel Van Stroud [1], and Johannes Wagner [9]

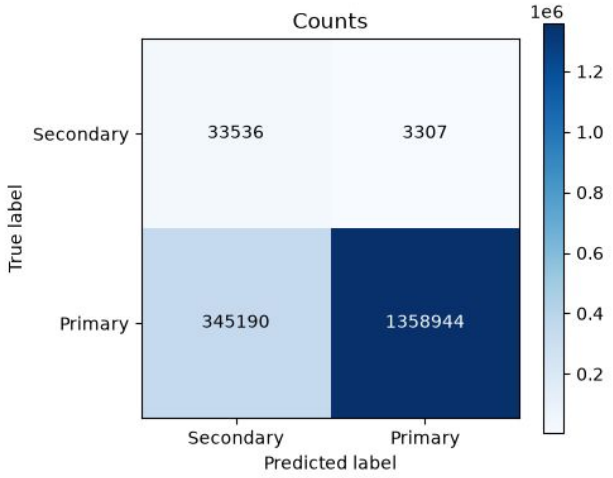
Vertexing using SaltModel

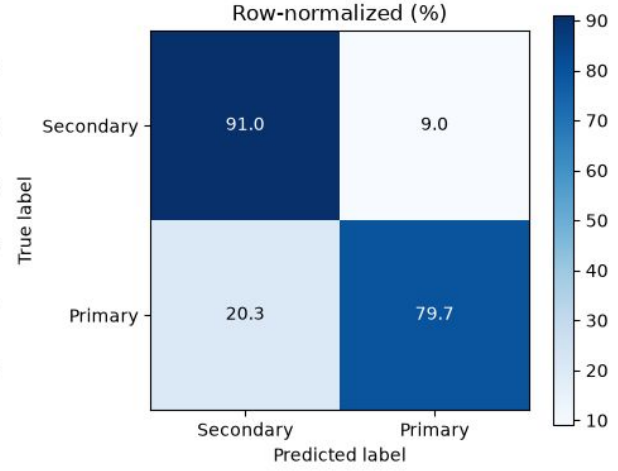
```
variables:
    events:
      - ntracks
    tracks:
      - d0
      - z0
      - qop
      - phi
      - theta

encoder:
    class_path: salt.models.Transformer
    init_args:
      num_layers: 4
      embed_dim: 32
      out_dim: 32
      attn_type: flash-varlen
      norm: RMSNorm
      norm_type: hybrid
      dense_kwargs:
        activation: SiLU
        gated: True
      attn_kwargs:
        num_heads: 4
      num_registers: 4
```

```
tasks:
    - class_path: salt.models.ClassificationTask
      init_args:
        name: track_origin
        input_name: tracks
        label: primary
        use_class_dict: True
        class_names: [secondary, primary]
        loss: torch.nn.CrossEntropyLoss
        dense_config:
          output_size: 2
          hidden_layers: [64, 32]

    - class_path: salt.models.VertexingTask
      init_args:
        name: track_vertexing
        input_name: tracks
        label: vertex_id
        loss:
          class_path: torch.nn.BCEWithLogitsLoss
          init_args: { reduction: none }
        dense_config:
          input_size: 64
          output_size: 1
          hidden_layers: [64, 32]
```

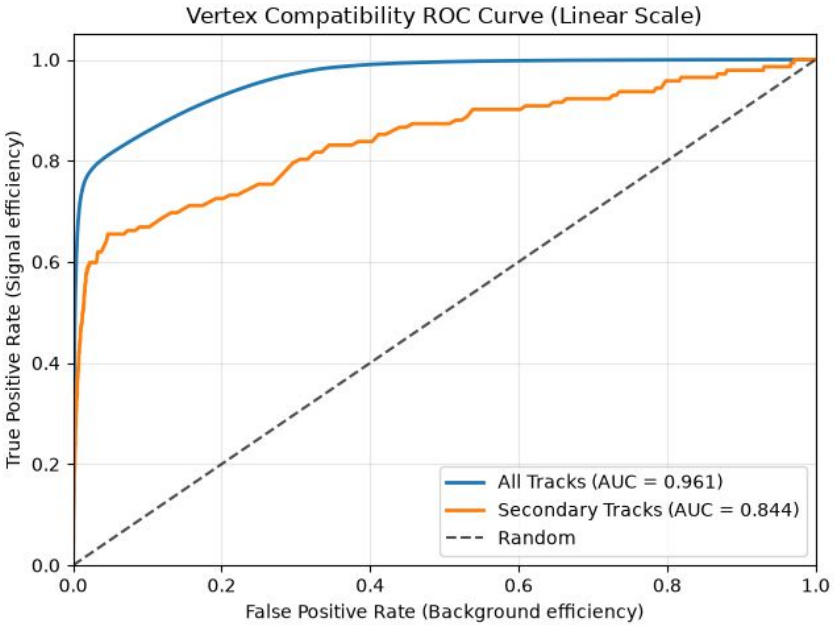
Track-origin classification

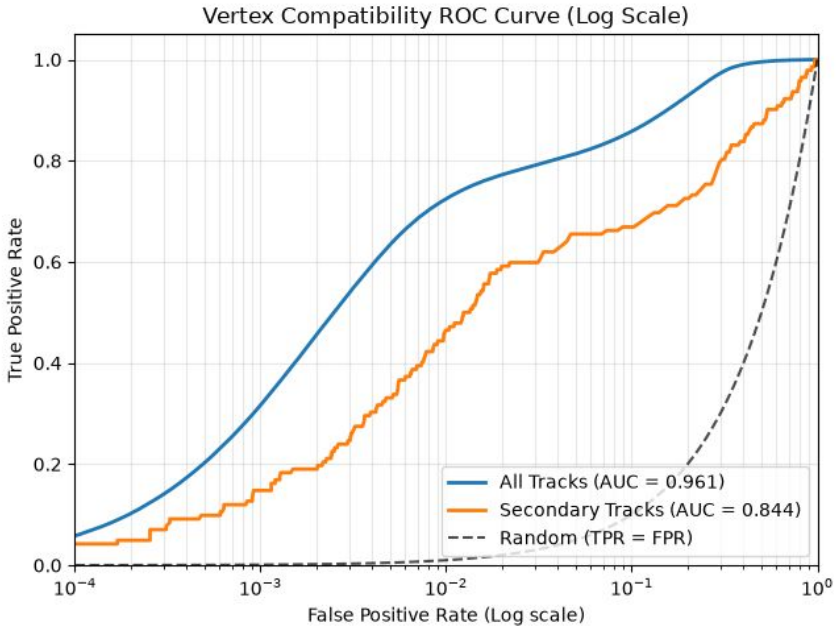




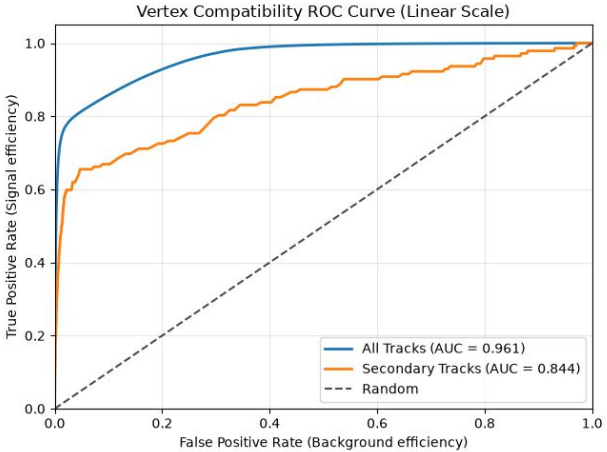


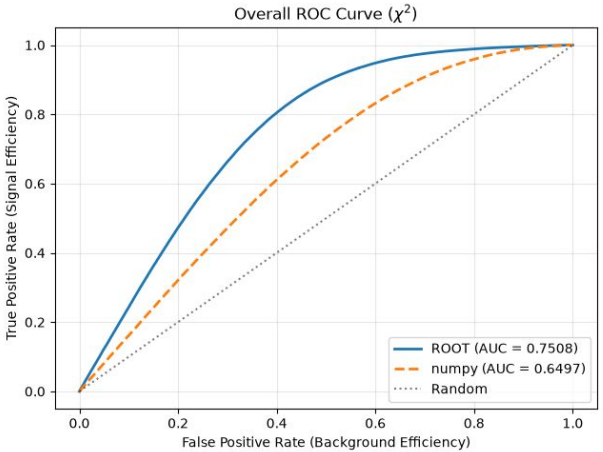
Track-pair compatibility

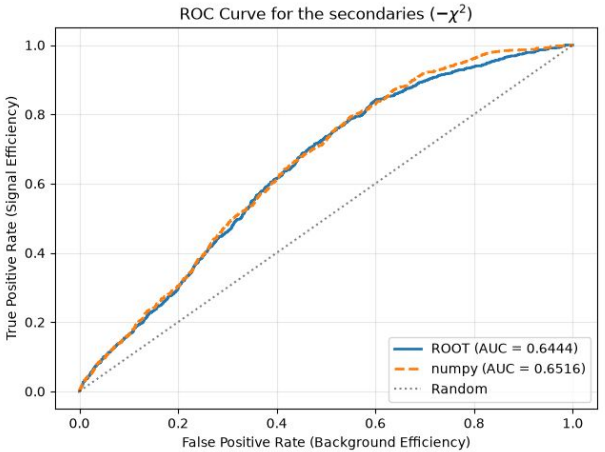




Track-pair compatibility







Method	AUC for all tracks	AUC for secondaries
Billoir fit (ROOT)	0.751	0.644
Billoir fit (numpy)	0.650	0.652
SaltModel	**0.961**	**0.844**

Summary and outlook

- Implemented Billoir fit in numpy, closely following Vadim's original implementation in ROOT.
  - ➢ Streamlined 1000+ lines of C++ code across 10+ files into ~150 lines of python code.
- The numpy implementation is then translated into pytorch using antigravity.
  - ➢ The implementation is ~ 800 lines long, and produces identical numbers.
  - ➢ Can be easily integrated into *salt* data-loader to calculate EDGE features.
- `SaltModel` predicts the track-pair compatibility matrix with a much better accuracy than the standalone Billoir fit.
  - ➢ This can be used with graph-partitioning, providing an alternate approach to MaskFormer.
  - ➢ Next step: integrate Billoir fit into *salt* data-loader and use EDGE features.