




A Tree-to-Token Paradigm for Jet Physics: Bridging High-Energy Phenomenology and Language Modeling



Masahiro Morinaga

The University of Tokyo 
(ICEPP , Beyond AI )

Self-interactions

- Masahiro Morinaga (森永真央), Project Assistant Professor at ICEPP
 - Born in Tokushima
 - Kobe University → ICEPP
 - Keyboard enthusiast
- Interested in Dark Matter and BSM physics
 - Computing, software, ML
- Working on ATLAS experiments
 - SUSY chargino search: Using short tracks to detect chargino decay
 - MSSM Higgs search: High-mass Higgs (H/A) decaying to a pair of tau leptons
 - Fast Track Trigger: (Terminated) ASIC + FPGA system to reconstruct tracks at 1 MHz



High Energy Particle Physics



- After the Higgs boson discovery, all particles predicted by the Standard Model (SM) have been found.
- There are a few experimental results that cannot be explained by the SM:
 - **Dark matter:** It exists, but there are no candidates within the SM.
 - **Neutrino Mass:** The SM assumes neutrinos are massless, yet neutrinos have mass.
 - **Fine-tuning problem:** The Higgs mass (125 GeV) is obtained as the difference between two quantities of order 10^{19} GeV.
- Many types of experiments:
 - **Neutrino experiments:** Super-/Hyper-Kamiokande
 - **Underground experiments:** Huge xenon detectors for neutrino interactions
 - **High energy colliders:** Produce rare events using colliders

Particles in The Standard Model of Particle Physics

Large Hadron Collider (LHC)

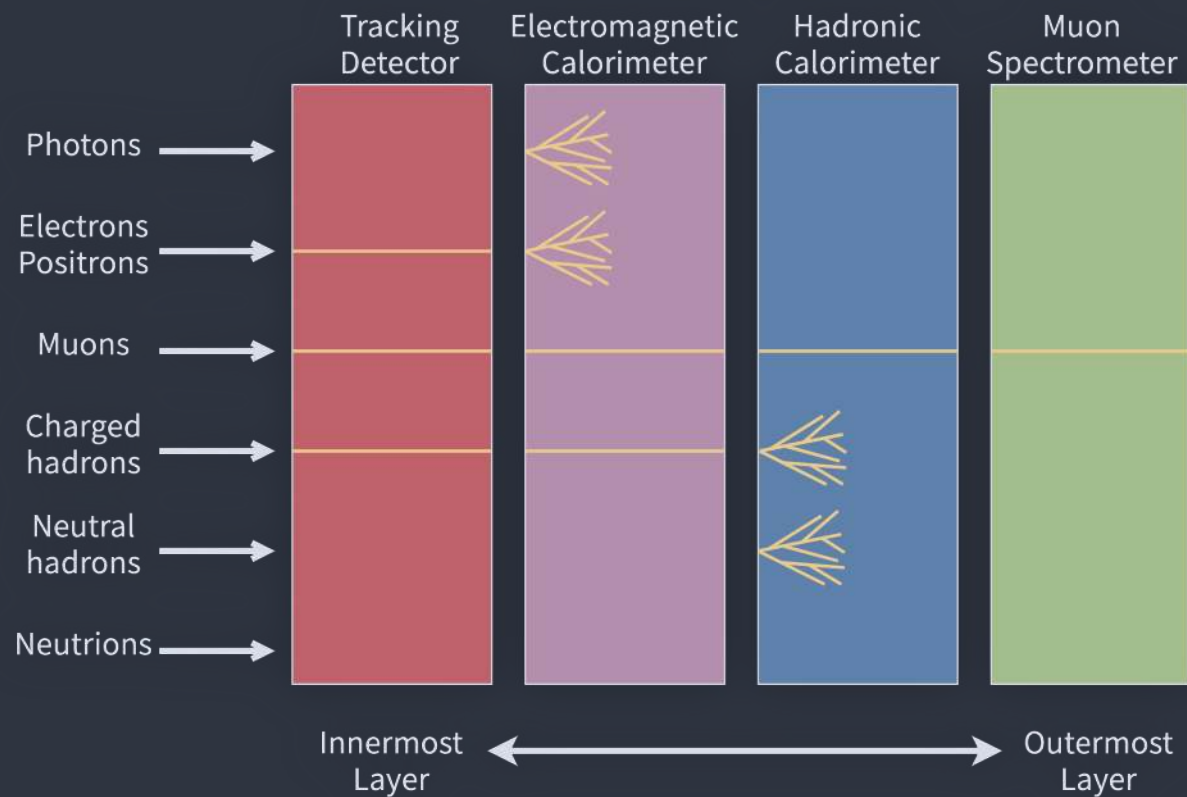


Large Hadron Collider

LHC

- Largest collider (27 km) using hadrons (protons, Au, etc.).
- Proton-proton collisions: 14 TeV (designed value, not yet reached).
- Precision measurement of SM particles.
- Higgs boson search (already discovered!).
- Searches for new particles to solve various problems:
 - Supersymmetry
 - Extra dimensions
 - etc.
- Worldwide computing grid (GRID).

Particle Detection

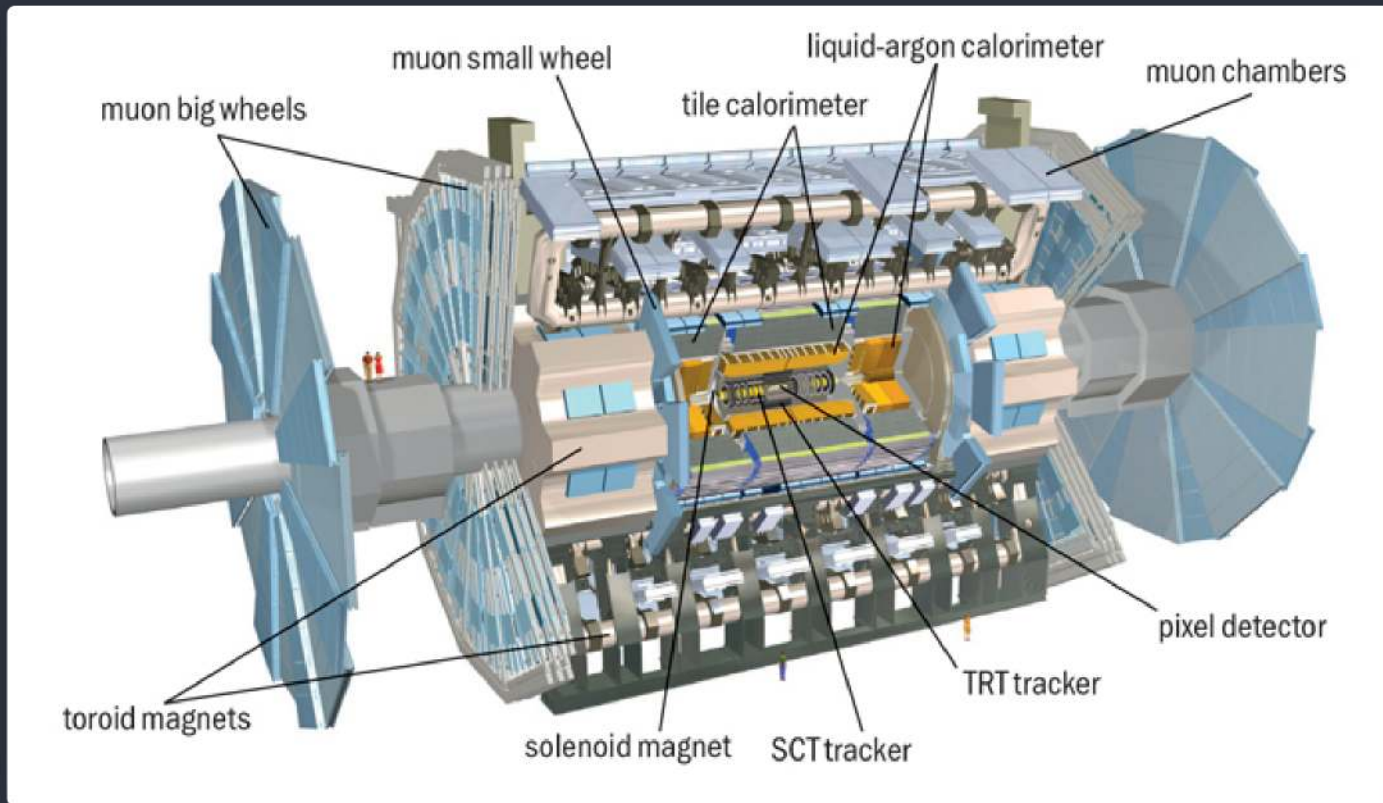


Particle Detector Concept @LHC

Particle Detector

- Highly optimized detectors for high-energy particles.
- Layered layout:
 - *Tracking*: Multiple layers to detect the tracks of charged particles.
 - *EM calorimeter*: Measures the energy of electromagnetic particles (e^\pm, γ).
 - *Hadronic calorimeter*: Measures the energy of charged/neutral hadrons.
 - *Muon spectrometer*: Detects muons.
- Neutrinos cannot be detected by the current detector concept;
 - however, the sum of their transverse momenta can be reconstructed using momentum conservation in the transverse direction.

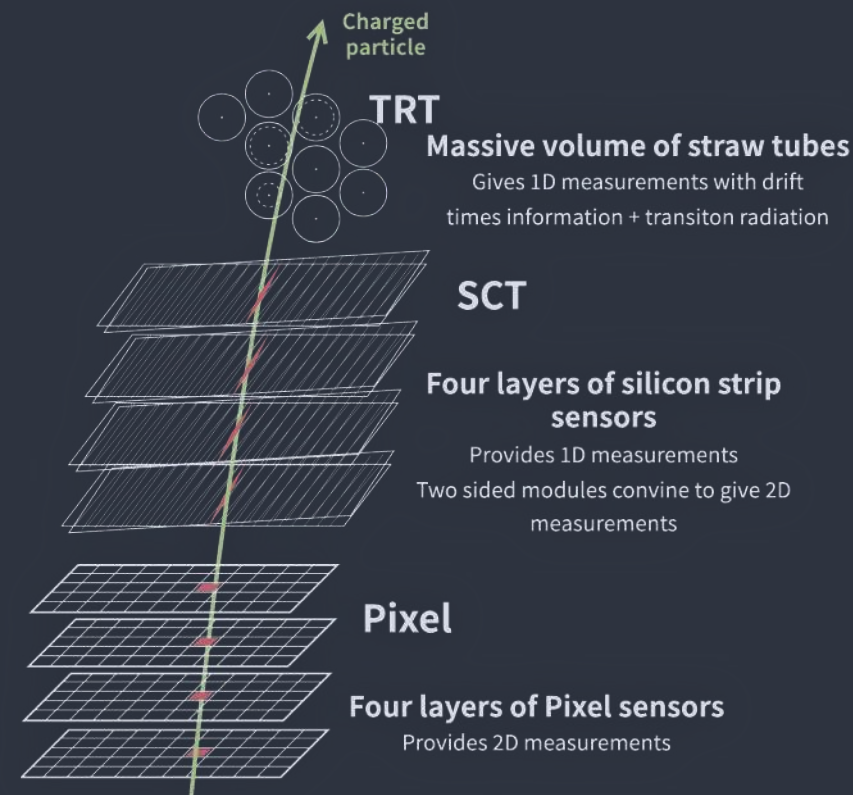
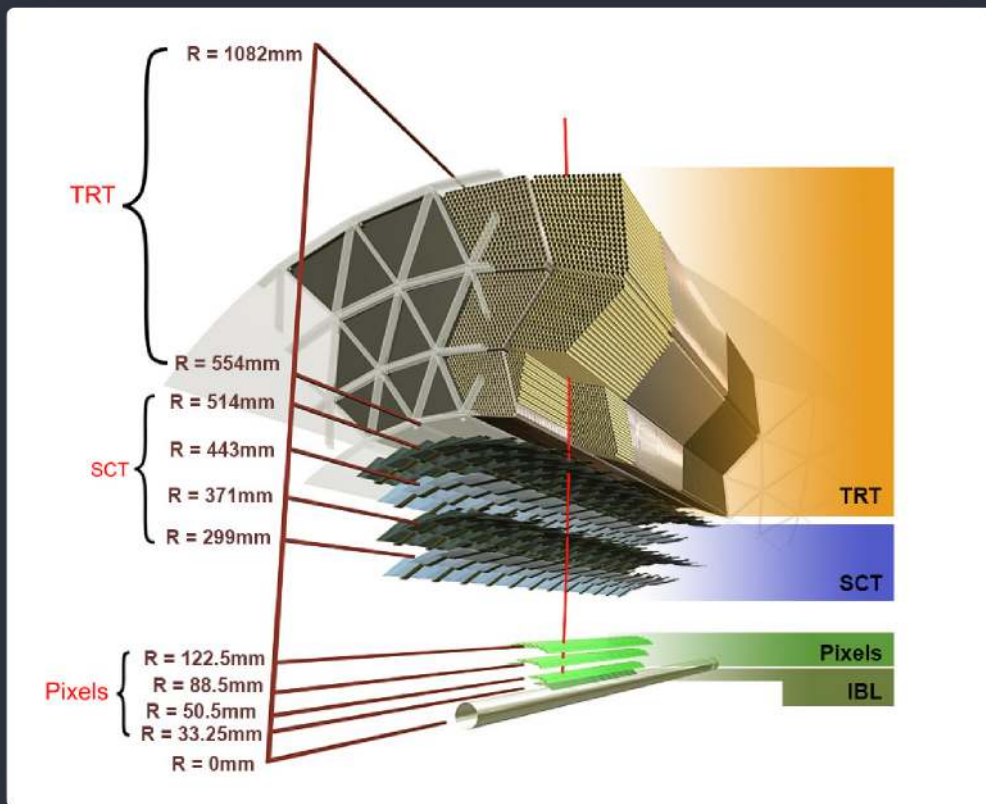
ATLAS Detector



ATLAS Detector

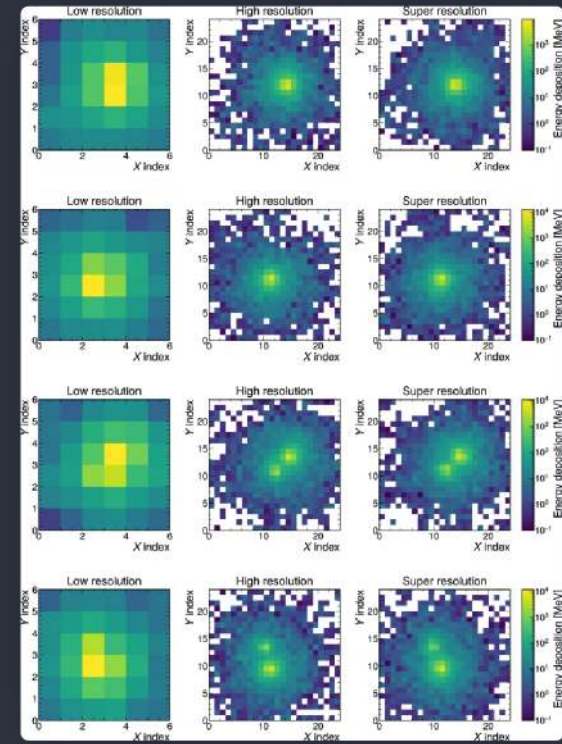
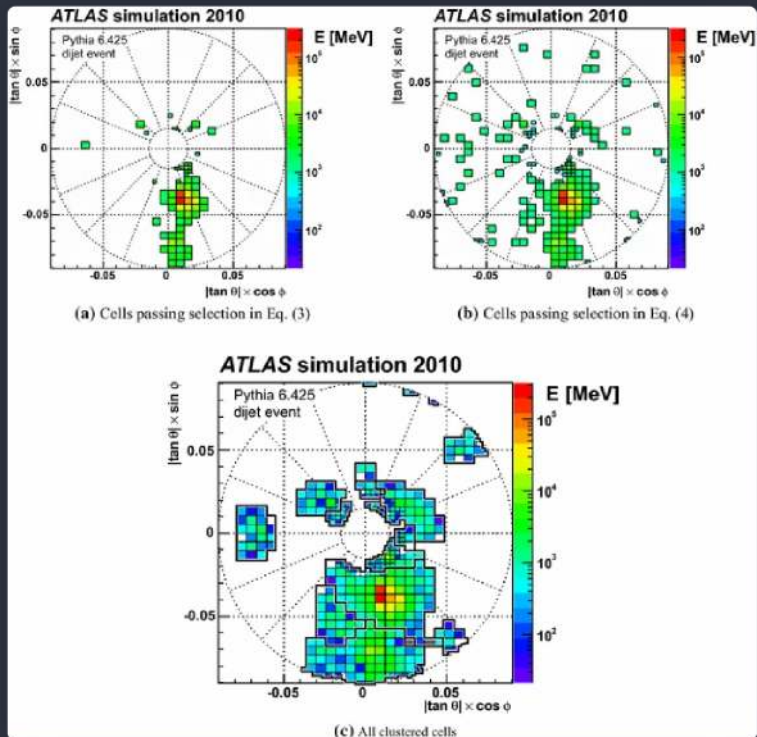
- *Inner tracker*: (Very) expensive, consisting of three different types of detectors
 - Pix: 4 layers of 2D silicon detectors
 - SCT: 6–8 layers of 1D silicon detectors
 - TRT: 20 layers of 1D gas detectors
- *Solenoid Magnet*: Extremely thin to measure the momentum of tracks.
- *EM Calo*: Liquid argon absorber.
- *H Calo*: Steel + plastic scintillator.
- *MS*: Cheap gas detectors covering most of the detector volume.
- *Toroid magnet*: Used to measure muon momentum.

Track Reconstruction



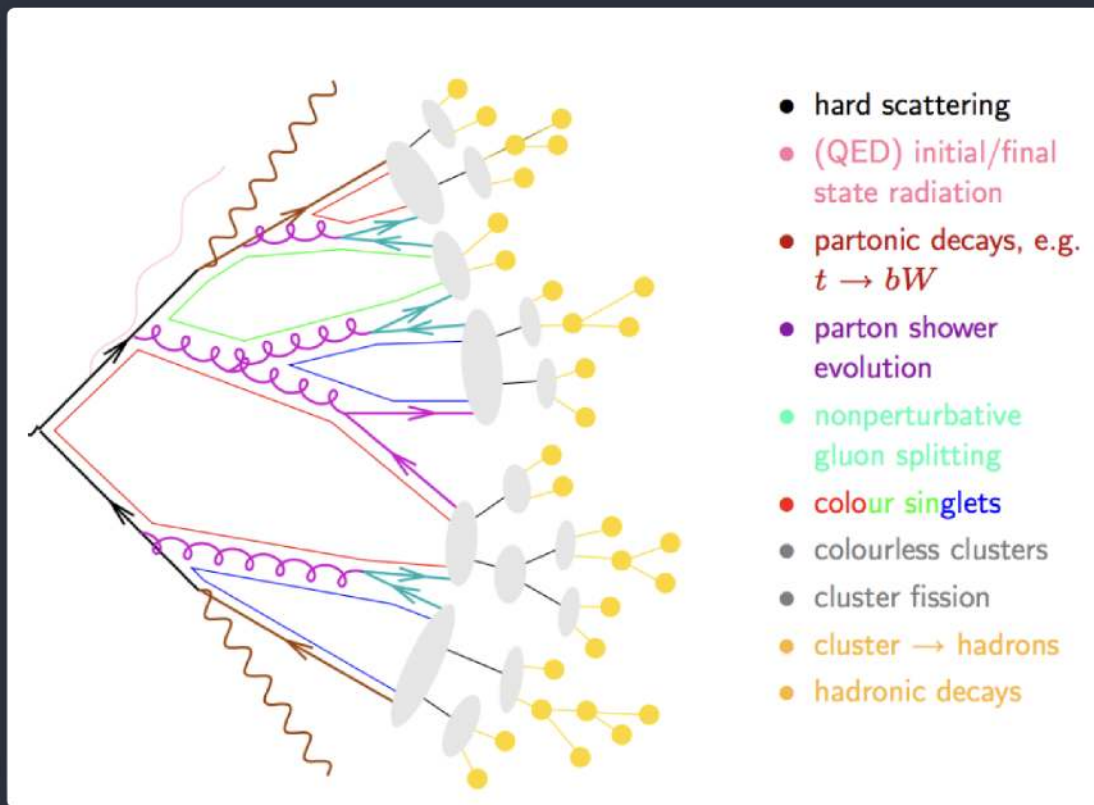
- Connect each 2D measurement to form individual positions.
- Fit using the perigee coordinates $(q/p, \phi, \theta, d_0, z_0)$.
 - Usually, a Kalman filter is used to search for 3D space-points along the current trajectory.
- There are studies reconstructing tracks with neural networks (most likely GNNs) and even quantum algorithms.

Clustering



- The electromagnetic and hadronic calorimeters have many cells to detect energy deposits.
- "Clustering" refers to grouping together cells that originate from the same particle.
 - For example, by selecting a seed cell (the cell with the highest signal-to-noise ratio) and then adding neighboring cells with $S/N > 2$.
- Many studies explore topological clustering using neural networks:
 - *Super resolution*: Improving the effective detector cell size.
 - *GNN clustering*: Treating each cell as a node with edges representing connections.

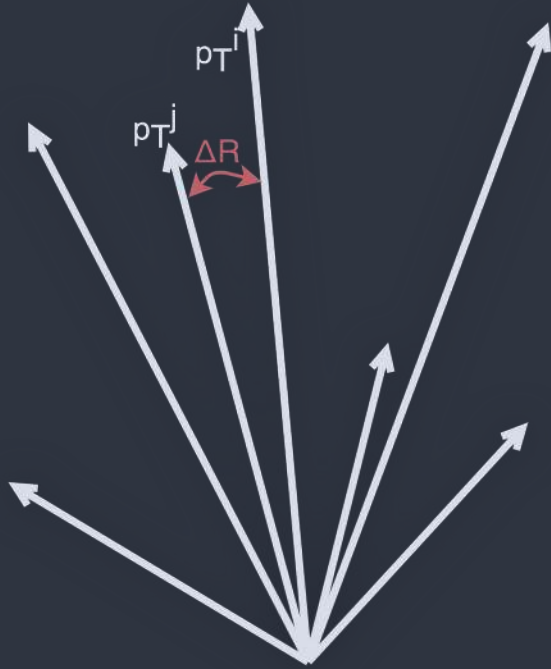
Jets



Jet production

- From hard scattering, jets are formed by several processes:
 - Decay: Particles have individual decay patterns.
 - Parton shower.
 - Gluon splitting.
 - ...
- In the final stages, we observe:
 - Charged particles: Tracks.
 - Neutral particles: Clusters.
 - (MIP: Muons)
- The original particle must be reconstructed using the hadrons observed in these final stages (tracks and clusters).
 - This is the main topic of this talk.

Anti- k_T Clustering

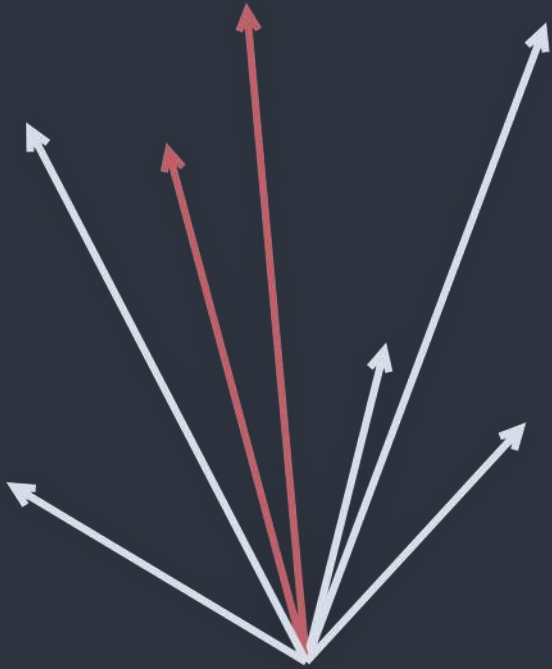


- List of jet constituents, each with four-momentum : (p_x, p_y, p_z, E) .
- For the pair with the minimum distance defined by

$$d_{ij} = \Delta R_{ij} \times \min(p_{T,i}^\alpha, p_{T,j}^\alpha)$$

- $p_T = \sqrt{p_x^2 + p_y^2}$: Transverse momentum.
 - ΔR_{ij} : Lorentz invariant distance between particles.
- Delete the pair and insert the merged new constituent into the list.
- Repeat deletion and merging until ΔR is greater than the R parameter or the list becomes empty.
 - Typically, $R = 0.4$ for lower momentum jets and $R = 1.0$ for higher momentum jets.
- The choice of the α parameter determines the appearance of the jet:
 - $\alpha = +1$: k_T algorithm, based on the minimum p_T pair.
 - $\alpha = 0$: C/A algorithm, based on the minimum ΔR pair.
 - $\alpha = -1$: anti- k_T algorithm, based on the maximum p_T pair.

Anti- k_T Clustering

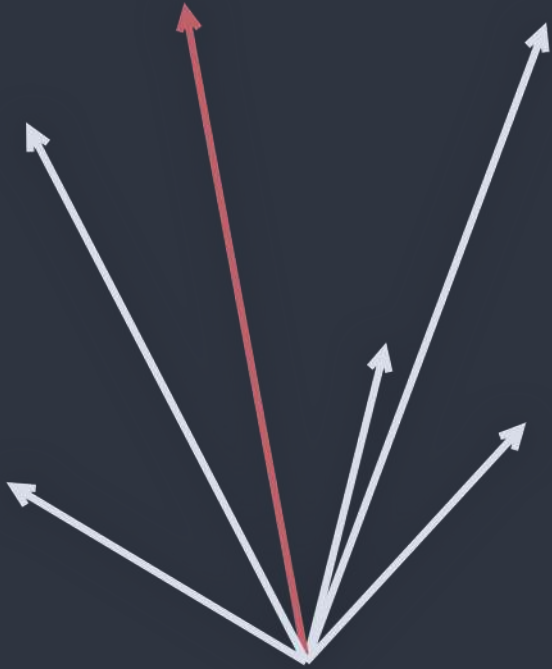


- List of jet constituents, each with four-momentum : (p_x, p_y, p_z, E) .
- For the pair with the minimum distance defined by

$$d_{ij} = \Delta R_{ij} \times \min(p_{T,i}^\alpha, p_{T,j}^\alpha)$$

- $p_T = \sqrt{p_x^2 + p_y^2}$: Transverse momentum.
 - ΔR_{ij} : Lorentz invariant distance between particles.
- Delete the pair and insert the merged new constituent into the list.
- Repeat deletion and merging until ΔR is greater than the R parameter or the list becomes empty.
 - Typically, $R = 0.4$ for lower momentum jets and $R = 1.0$ for higher momentum jets.
- The choice of the α parameter determines the appearance of the jet:
 - $\alpha = +1$: k_T algorithm, based on the minimum p_T pair.
 - $\alpha = 0$: C/A algorithm, based on the minimum ΔR pair.
 - $\alpha = -1$: anti- k_T algorithm, based on the maximum p_T pair.

Anti-kT Clustering

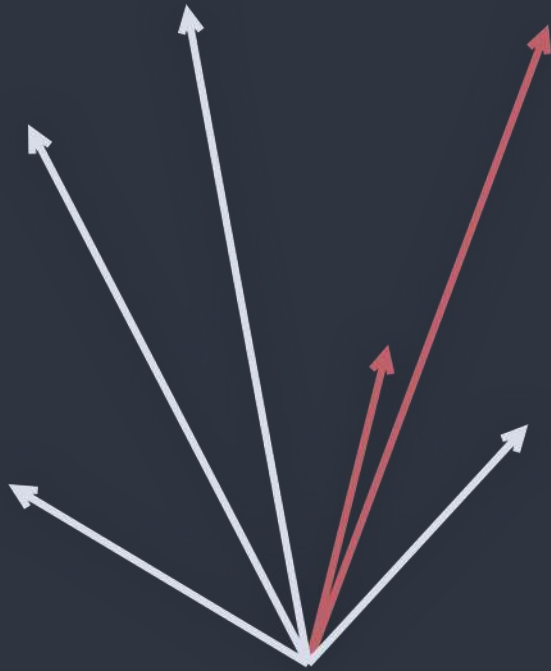


- List of jet constituents, each with four-momentum : (p_x, p_y, p_z, E) .
- For the pair with the minimum distance defined by

$$d_{ij} = \Delta R_{ij} \times \min(p_{T,i}^\alpha, p_{T,j}^\alpha)$$

- $p_T = \sqrt{p_x^2 + p_y^2}$: Transverse momentum.
 - ΔR_{ij} : Lorentz invariant distance between particles.
- Delete the pair and insert the merged new constituent into the list.
- Repeat deletion and merging until ΔR is greater than the R parameter or the list becomes empty.
 - Typically, $R = 0.4$ for lower momentum jets and $R = 1.0$ for higher momentum jets.
- The choice of the α parameter determines the appearance of the jet:
 - $\alpha = +1$: kT algorithm, based on the minimum p_T pair.
 - $\alpha = 0$: C/A algorithm, based on the minimum ΔR pair.
 - $\alpha = -1$: anti-kT algorithm, based on the maximum p_T pair.

Anti- k_T Clustering



- List of jet constituents, each with four-momentum : (p_x, p_y, p_z, E) .
- For the pair with the minimum distance defined by

$$d_{ij} = \Delta R_{ij} \times \min(p_{T,i}^\alpha, p_{T,j}^\alpha)$$

- $p_T = \sqrt{p_x^2 + p_y^2}$: Transverse momentum.
 - ΔR_{ij} : Lorentz invariant distance between particles.
- Delete the pair and insert the merged new constituent into the list.
- Repeat deletion and merging until ΔR is greater than the R parameter or the list becomes empty.
 - Typically, $R = 0.4$ for lower momentum jets and $R = 1.0$ for higher momentum jets.
- The choice of the α parameter determines the appearance of the jet:
 - $\alpha = +1$: k_T algorithm, based on the minimum p_T pair.
 - $\alpha = 0$: C/A algorithm, based on the minimum ΔR pair.
 - $\alpha = -1$: anti- k_T algorithm, based on the maximum p_T pair.

Anti- k_T Clustering



- List of jet constituents, each with four-momentum : (p_x, p_y, p_z, E) .
- For the pair with the minimum distance defined by

$$d_{ij} = \Delta R_{ij} \times \min(p_{T,i}^\alpha, p_{T,j}^\alpha)$$

- $p_T = \sqrt{p_x^2 + p_y^2}$: Transverse momentum.
- ΔR_{ij} : Lorentz invariant distance between particles.
- Delete the pair and insert the merged new constituent into the list.
- Repeat deletion and merging until ΔR is greater than the R parameter or the list becomes empty.
 - Typically, $R = 0.4$ for lower momentum jets and $R = 1.0$ for higher momentum jets.
- The choice of the α parameter determines the appearance of the jet:
 - $\alpha = +1$: k_T algorithm, based on the minimum p_T pair.
 - $\alpha = 0$: C/A algorithm, based on the minimum ΔR pair.
 - $\alpha = -1$: anti- k_T algorithm, based on the maximum p_T pair.

Anti- k_T Clustering

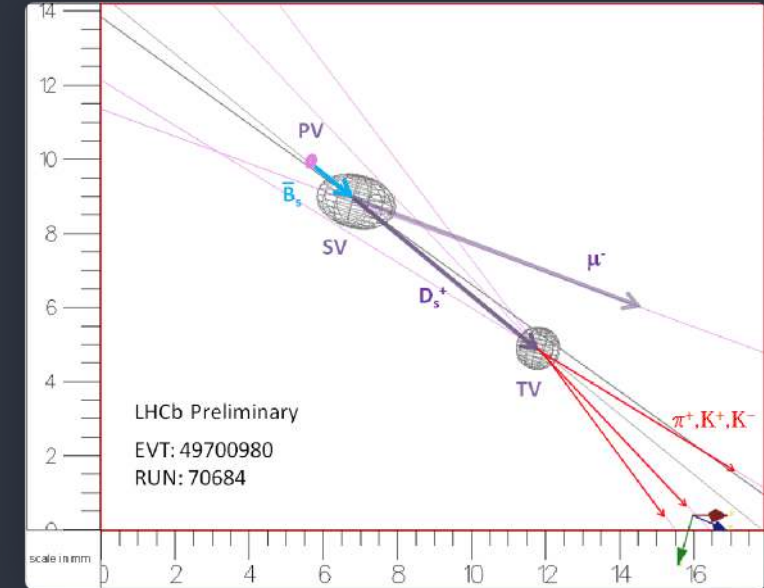
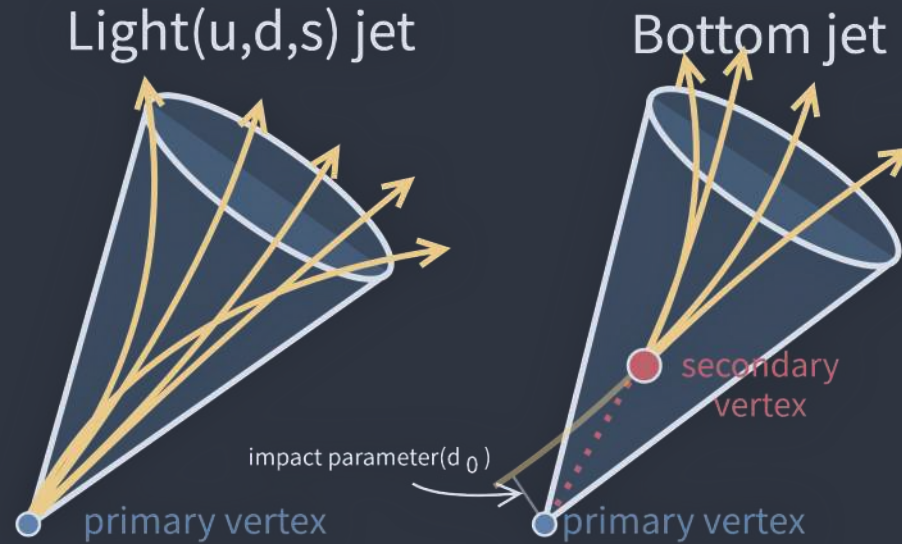


- List of jet constituents, each with four-momentum : (p_x, p_y, p_z, E) .
- For the pair with the minimum distance defined by

$$d_{ij} = \Delta R_{ij} \times \min(p_{T,i}^\alpha, p_{T,j}^\alpha)$$

- $p_T = \sqrt{p_x^2 + p_y^2}$: Transverse momentum.
- ΔR_{ij} : Lorentz invariant distance between particles.
- Delete the pair and insert the merged new constituent into the list.
- Repeat deletion and merging until ΔR is greater than the R parameter or the list becomes empty.
 - Typically, $R = 0.4$ for lower momentum jets and $R = 1.0$ for higher momentum jets.
- The choice of the α parameter determines the appearance of the jet:
 - $\alpha = +1$: k_T algorithm, based on the minimum p_T pair.
 - $\alpha = 0$: C/A algorithm, based on the minimum ΔR pair.
 - $\alpha = -1$: anti- k_T algorithm, based on the maximum p_T pair.

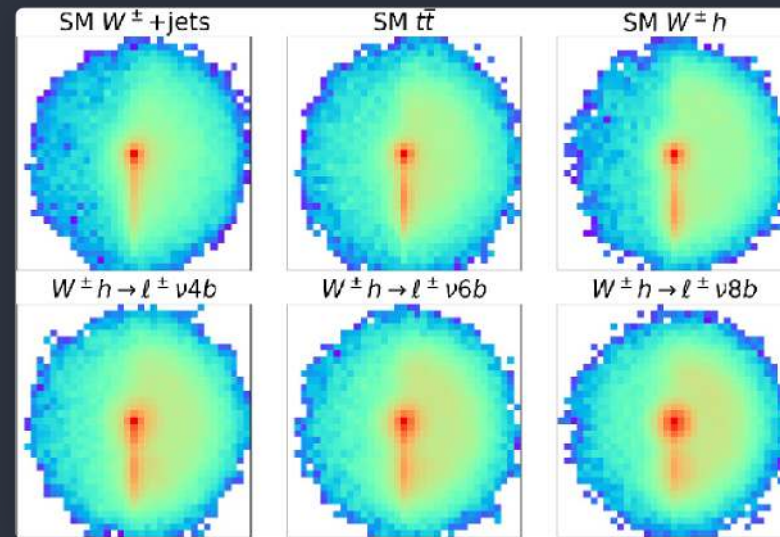
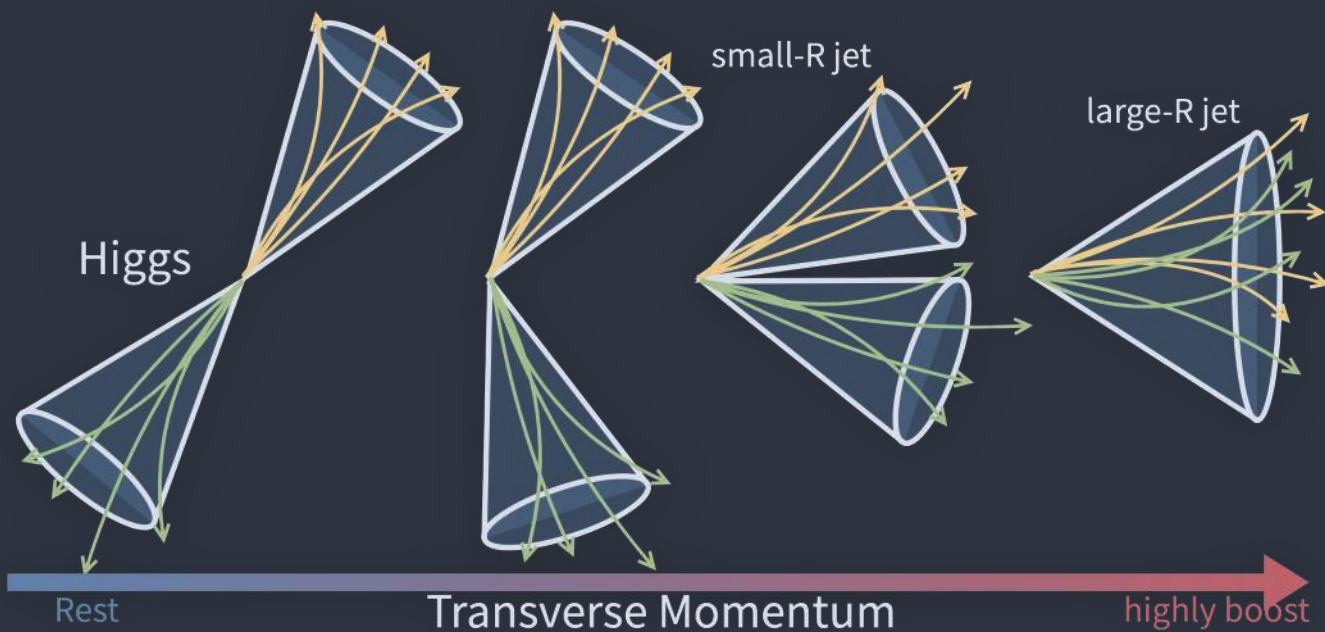
Jets Application: Flavor Tagging



Flavor Tagging

- Distinguishing jets originating from heavy flavor quarks (bottom or charm) from other jets.
- Heavy flavor quark decays are nearly identical; for example, a bottom quark decays via $b \rightarrow B \text{ meson} \rightarrow D \text{ meson}$, involving two additional vertices.
- Tracks from these decays typically have larger impact parameters.
 - The original b-tagging algorithms relied on tracks with large impact parameters.
- Recently, GNNs or Transformers have replaced older algorithms (e.g., BDTs, RNNs, CNNs).

Jets Application: Large- R Jet Tagging

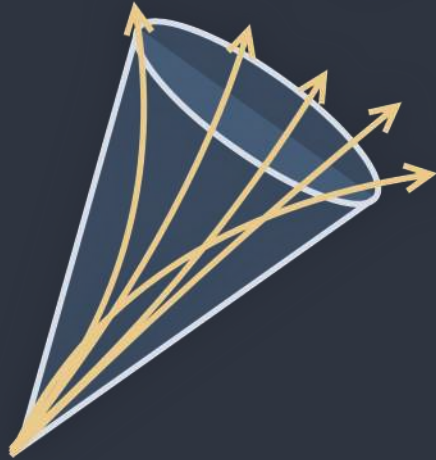


Boosted Jet

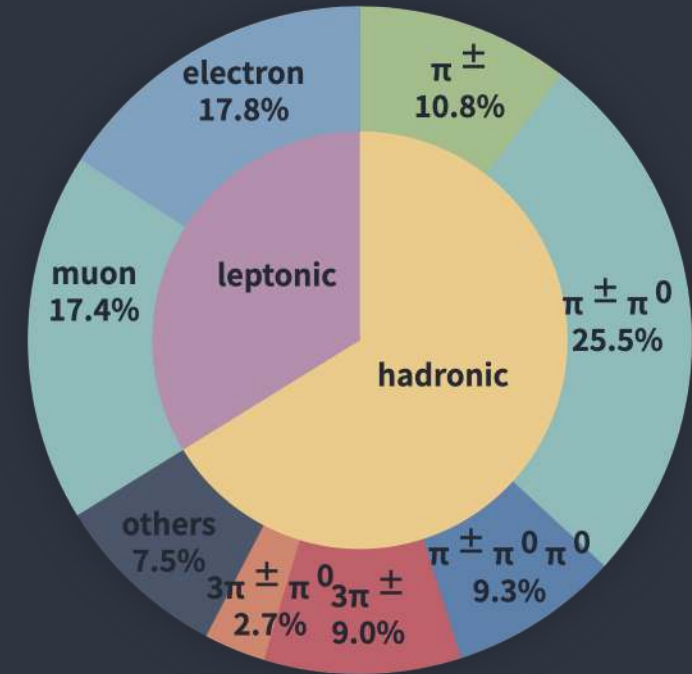
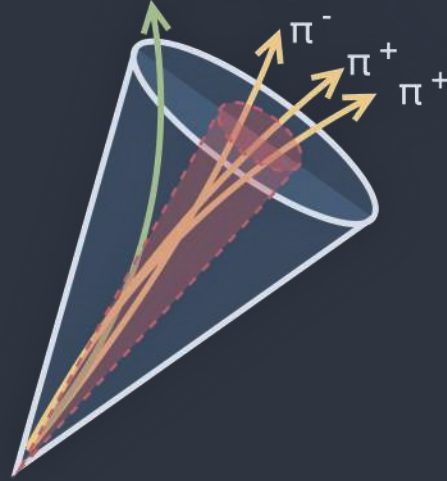
- When particles like the Higgs/W/Z boson or the top quark are boosted, their decay products become close and collimated.
- To reconstruct such objects, a larger R jet is used—typically $R \sim 1.0$.
- These objects exhibit internal structures (e.g., two- or three-pronged).
- CNNs (or more recently, Transformers) have been used to tag these objects.

Jets Application: Tau Identification

Normal jet



Tau jet

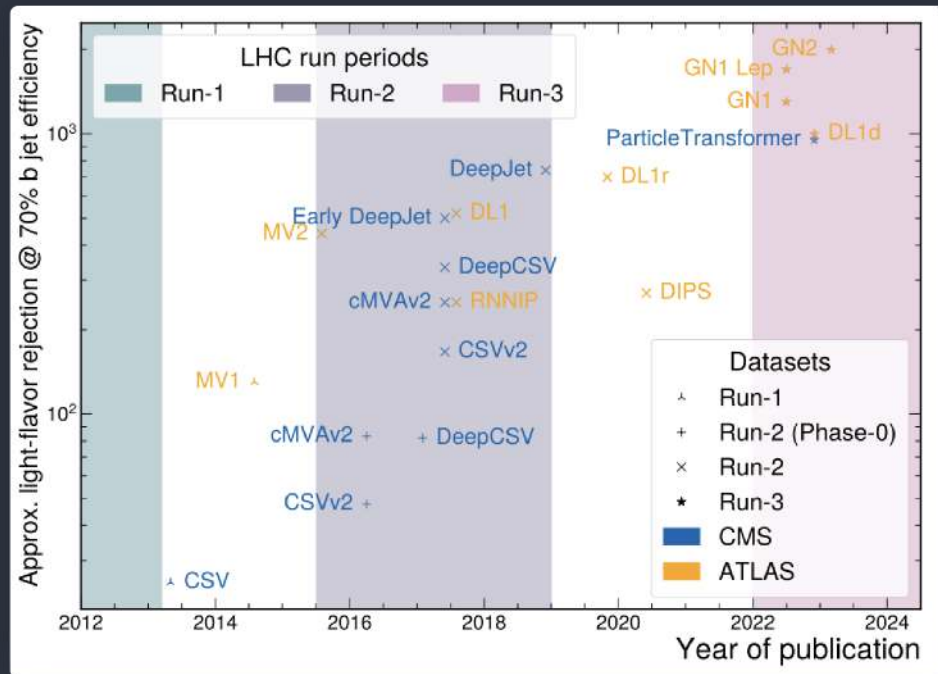


Hadronic Tau Decay

- Produces a narrow jet with a characteristic decay pattern due to tau lepton decay:
 - Since the tau lepton mass ($m_\tau \sim 1.777$ GeV) is negligible at the LHC energy scale, the tau is highly boosted and produces a narrow jet.
 - Typically, one or three charged tracks are observed, with a total charge of ± 1 .
 - There may also be neutral pions.
- Due to these features, hadronic tau tagging is highly optimized for this purpose.
 - Originally, BDTs were the best taggers, but currently tuned GNNs (or Transformers) are being utilized.

Bridging Jet and Language Model

Machine Learning in Jet Physics



[review paper](#)

Machine Learning in Jet Physics

- Jet physics has evolved alongside the development of ML:
 - CSV, MV1/2: Based on track impact parameters, (non-deep) neural networks, BDTs.
 - JetImage: Processes images of jet constituents using CNNs.
 - DeepSets: Uses invariant operations to yield identical outputs.
 - RNNs: Can process variable-length inputs, useful since jets have a varying number of tracks.
 - Graph NNs: Consist of nodes, edges, and global attributes; well adapted for jet data.
 - Transformers: The most common architecture at the moment.
- Most current models treat jet constituents as point clouds; GNNs can directly use the graph structure as input.

Current SoTA Model

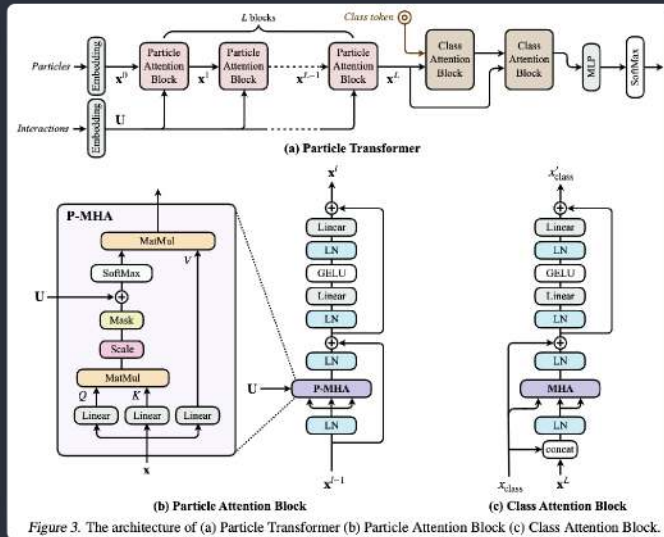
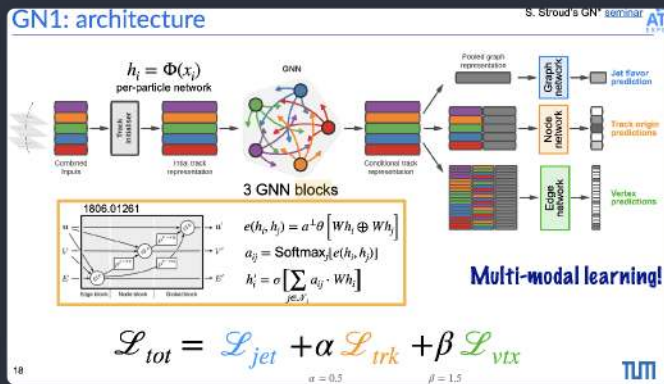


Figure 3. The architecture of (a) Particle Transformer (b) Particle Attention Block (c) Class Attention Block.

ParticleTransformer

Machine Learning in Jet Physics

- Current state-of-the-art (or near) models include:
 - Transformer**: Treats jet constituents (tracks or clusters) as point clouds.
 - Graph-based models**: Treat jet constituents as a fully connected graph.
- Both Transformers and GNNs treat inputs as fully connected graphs—the Transformer does this via its attention layer.
- Here's a question: In what order should the inputs to the model be arranged?
 - In the latest models, they are often sorted by p_T before being input.
- And why is a fully connected structure necessary?
 - Because we do not know which components are connected in terms of particle decay.
- Let's take a closer look at particle decay.



ATLAS GN2

Jets as Tree: Truth Particle

Particle Decay

- In principle, particle decay can be represented as a binary tree, e.g.:
 - $W^- \rightarrow \ell^- + \nu$
 - $Z \rightarrow q\bar{q}$
- However, after showering and hadronization, this structure collapses and one-to-many decays occur, e.g.:
 - $\tau^- \rightarrow \pi^- + \pi^0 + \nu$

Example:

- Bottom quark decay (ignoring the d from the W):
 - $b \rightarrow \pi^+ + \pi^- + 2\gamma$
- This is an abbreviated version; a detailed diagram would be longer.

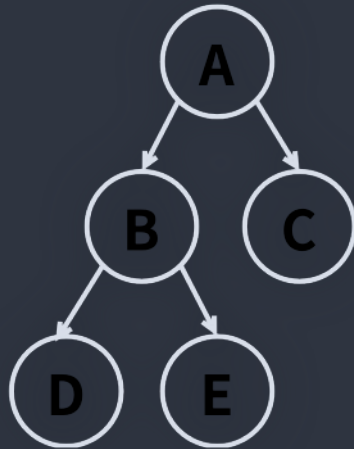
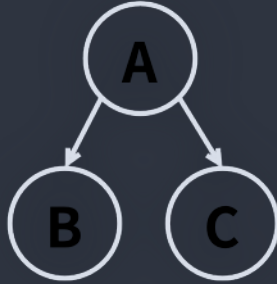
$$b(\rightarrow c + W^-) \rightarrow D^0 \rightarrow K^- + \pi^+$$

$$\hookrightarrow \pi^0 + \pi^-$$

$$\hookrightarrow \gamma + \gamma$$

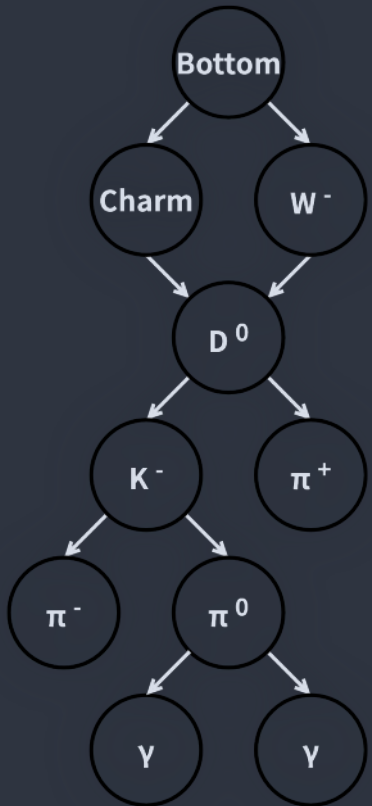
- This is what we want to understand.

Jets as Tree: Tree Structure



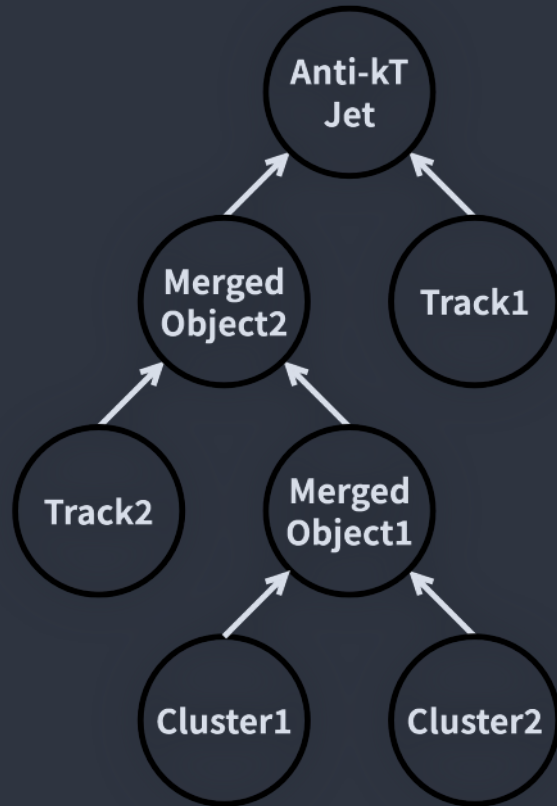
- Bottom quark decay (ignoring the d from the W) can be represented as:
 - $b \rightarrow c + W^- \rightarrow D^0 \rightarrow K^- + \pi^+ \rightarrow \pi^- + \pi^0 \rightarrow \gamma + \gamma \rightarrow \pi^+ + \pi^- + 2\gamma$
- This can be written using a *Parenthesis Representation* similar to Lisp:
 - Top example: `(A (B) (C))`
 - Bottom example: `(A (B (D) (E)) (C))`
- Thus, the bottom quark decay can be represented as:
 - `(b (c W^- (D^0 (K^- (\pi^-) (\pi^0 (\gamma) (\gamma)))) (\pi^+)))`

Jets as Tree: Truth Particle Decay as Sequence



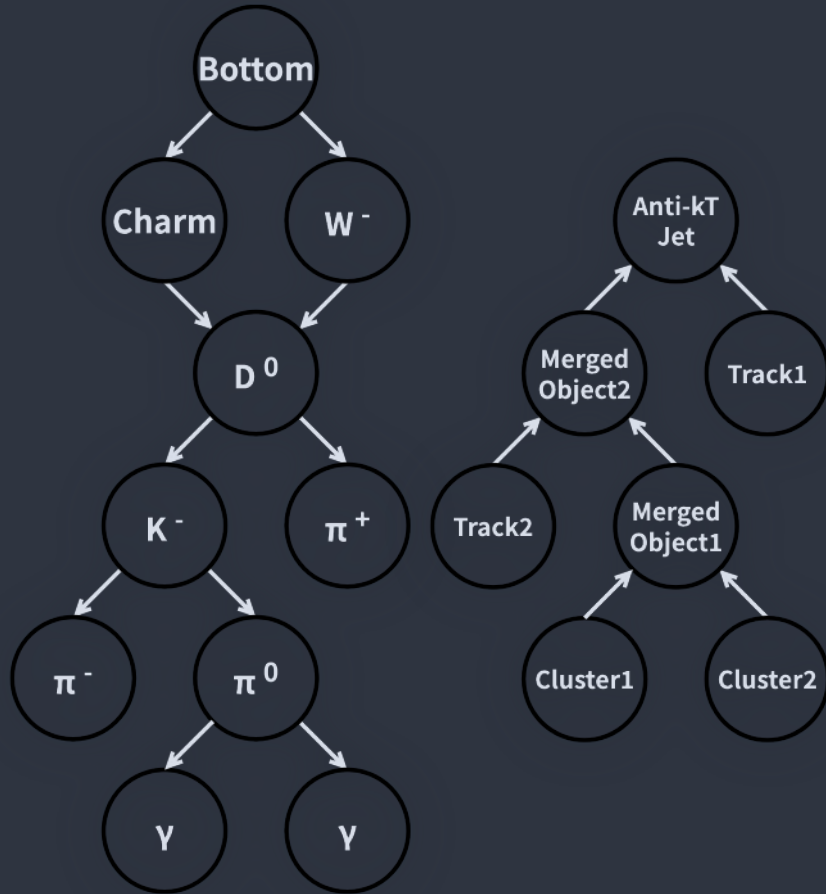
- Bottom quark decay (ignoring the d from the W) can be written as:
 - $b \rightarrow c + W^- \rightarrow D^0 \rightarrow K^- + \pi^+ \rightarrow \pi^- + \pi^0 \rightarrow \gamma + \gamma \rightarrow \pi^+ + \pi^- + 2\gamma$
- Using a Parenthesis Representation (like Lisp):
 - `(b (c W^- (D^0 (K^- (\pi^+) (\pi^0 (\gamma) (\gamma)))) (\pi^+)))`
- If we interpret the above expression as a form of language, it can be tokenized as:
 - `[0, 2, 4, 2, 5, 10, 2, 30, 2, 19, 2, 49, 3, 2, 48, 2, 6, 3, 2, 6, 3, 3, 3, 2, 47, 3, 3, 3, 3, 1]`
- This sequence can be used to train a Transformer (or another language model).
 - **Note:** This expression includes its decay structure in a “brace-brace” pattern; the Transformer will learn how particles decay into other particles.
- OK, then how about the reconstructed jet?

Jets as Tree: Reconstructed Jet



- For the reconstructed jet, such as anti- k_T jets, we can use the history of clustering.
 - **History:**
 - Merge cluster1 and cluster2 \rightarrow Merged object1
 - Merge Track2 and merged object1 \rightarrow Merged object2
 - Merge Merged object2 and track1 \rightarrow Reconstructed jet!
 - We can write this history as a merged chain:
 - $Jet \leftarrow T1 + MO2 \leftarrow T2 + MO1 \leftarrow C1 + C2$
 - Then, this history can be translated into a *Parenthesis Representation*, for example:
 - `(Jet (MO2 (T2) (MO1 (C1) (C2))) (T1))`
 - And tokenized as:
 - `[0, 2, 71, 2, 81, 2, 93, 3, 2, 82, 2, 101, 3, 3, 3, 2, 104, 3, 3, 1]`
 - This sequence also includes structural information from anti- k_T clustering.
- There might be a better clustering algorithm for this dedicated study.

Jets as Tree: Truth and Reconstructed Jet



- It is not clear if the anti- k_T clustering history matches (or includes) the true decay chain.
 - They have different information levels and different clustering structures.
- Points of concern:
 - Ordering of sub-trees: (Top (W (q1) (q2)) (b)) versus (Top (b) (W (q1) (q2)))
 - Non-binary tree: The anti- k_T history is always a binary tree, but the true decay chain is not.
 - Number of tokens: There is an $\mathcal{O}(10)$ difference between point clouds and this expression.
 - Handling measurement uncertainties in discrete structures.
 - Defining evaluation metrics.
- Seq2seq learning (reco \rightarrow truth) should theoretically work:
 - Reco tokens \rightarrow Transformer (encoder-decoder) \rightarrow truth tokens (target, cross-entropy loss).

Example from Pythia

```

idx=-5 pid=0(status=-1), v=( 77.16, -0.68, -2.20, 96.43)
idx=-5 pid=0(status=-1), v=( 70.80, -0.66, -2.20, 87.09)
idx=-5 pid=0(status=-1), v=( 61.53, -0.65, -2.17, 75.32)
idx=-5 pid=0(status=-1), v=( 54.68, -0.66, -2.17, 67.30)
idx=-5 pid=0(status=-1), v=( 37.86, -0.68, -2.21, 47.06)
idx=-5 pid=0(status=-1), v=( 31.53, -0.68, -2.21, 39.18)
idx=-5 pid=0(status=-1), v=( 23.70, -0.69, -2.20, 29.66)
idx=10 pid=-211(status=-1), v=( 15.99, -0.68, -2.20, 19.90)
idx=11 pid=22(status=-1), v=( 7.72, -0.71, -2.22, 9.76)
idx=-5 pid=0(status=-1), v=( 7.82, -0.65, -2.22, 9.52)
idx=7 pid=211(status=-1), v=( 6.55, -0.64, -2.22, 7.96)
idx=8 pid=22(status=-1), v=( 1.28, -0.65, -2.21, 1.56)
idx=-5 pid=0(status=-1), v=( 6.34, -0.68, -2.25, 7.88)
idx=6 pid=22(status=-1), v=( 3.91, -0.69, -2.26, 4.88)
idx=9 pid=22(status=-1), v=( 2.43, -0.67, -2.24, 3.00)
idx=-5 pid=0(status=-1), v=( 16.93, -0.62, -2.08, 20.24)
idx=3 pid=22(status=-1), v=( 14.85, -0.62, -2.08, 17.79)
idx=4 pid=22(status=-1), v=( 2.08, -0.59, -2.09, 2.45)
idx=5 pid=0(status=-1), v=( 6.85, -0.58, -2.17, 8.02)
idx=2 pid=0(status=-1), v=( 9.40, -0.70, -2.35, 11.77)
idx=-5 pid=0(status=-1), v=( 6.37, -0.91, -2.28, 9.33)
idx=0 pid=2212(status=-1), v=( 3.70, -0.95, -2.25, 5.60)
idx=1 pid=22(status=-1), v=( 2.68, -0.86, -2.32, 3.73)

```

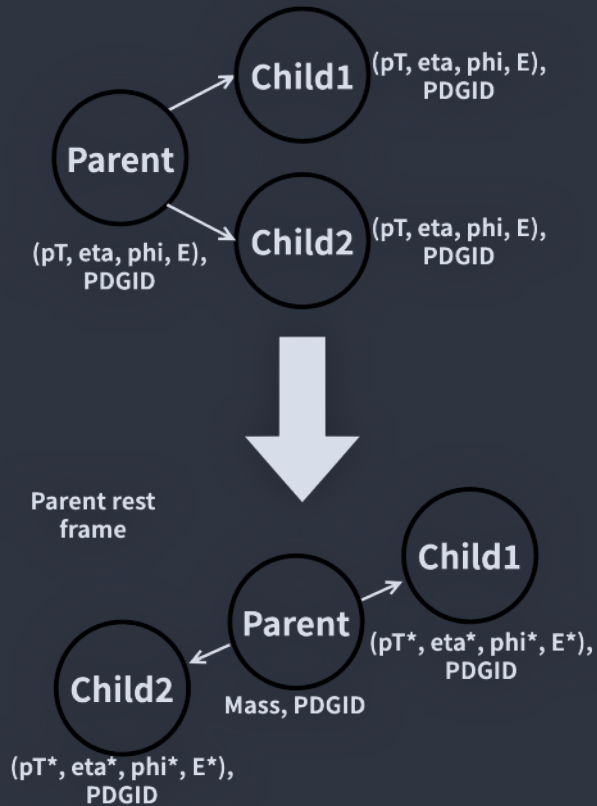
```

idx=568 pid=5(status=52), v=( 71.14, -0.66, -2.17, 87.55)
idx=635 pid=-523(status=2), v=( 66.62, -0.66, -2.17, 81.64)
idx=1041 pid=-521(status=2), v=( 66.55, -0.66, -2.17, 81.55)
idx=1300 pid=413(status=2), v=( 28.20, -0.66, -2.22, 34.61)
idx=1424 pid=411(status=2), v=( 27.82, -0.66, -2.22, 34.20)
idx=1459 pid=310(status=2), v=( 14.40, -0.70, -2.24, 18.08)
idx=1474 pid=111(status=2), v=( 7.86, -0.71, -2.22, 9.93)
idx=1479 pid=22(status=1), v=( 7.50, -0.71, -2.22, 9.49)
idx=1480 pid=22(status=1), v=( 0.36, -0.64, -2.18, 0.44)
idx=1473 pid=111(status=2), v=( 6.55, -0.69, -2.27, 8.15)
idx=1477 pid=22(status=1), v=( 4.04, -0.69, -2.28, 5.05)
idx=1478 pid=22(status=1), v=( 2.51, -0.67, -2.25, 3.10)
idx=1441 pid=323(status=2), v=( 13.43, -0.62, -2.20, 16.13)
idx=1472 pid=130(status=1), v=( 6.88, -0.60, -2.18, 8.16)
idx=1458 pid=211(status=1), v=( 6.55, -0.65, -2.23, 7.96)
idx=1425 pid=22(status=1), v=( 0.37, -0.44, -2.21, 0.41)
idx=1303 pid=111(status=2), v=( 17.44, -0.61, -2.08, 20.81)
idx=1426 pid=22(status=1), v=( 14.81, -0.62, -2.08, 17.70)
idx=1427 pid=22(status=1), v=( 2.63, -0.59, -2.09, 3.11)
idx=1302 pid=-211(status=1), v=( 16.60, -0.69, -2.19, 20.67)
idx=1301 pid=-211(status=1), v=( 4.42, -0.67, -2.14, 5.46)
idx=1042 pid=22(status=1), v=( 0.07, -0.61, -2.48, 0.09)
idx=637 pid=-2112(status=1), v=( 2.17, -0.82, -2.13, 3.10)
idx=636 pid=2212(status=1), v=( 1.86, -0.93, -2.25, 2.88)

```

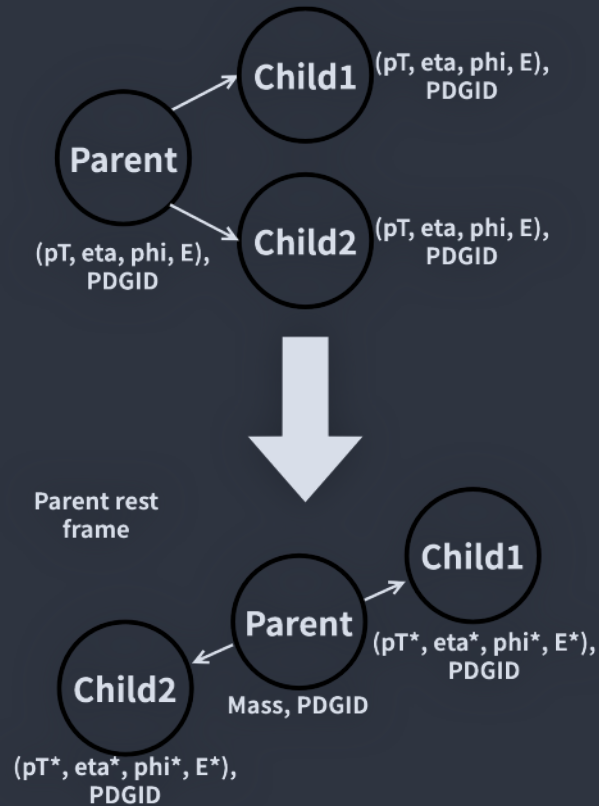
- Almost raw output from some jet, `v=(pt, eta, phi, energy)`
 - Left : Reconstructed jet, Right : Truth jet
- We need to tokenize these lines

How to Tokenize Jet Constituents: Parent



- Which information should be embedded into the token?
- Let's start with a basic structure: `(Parent (Child1) (Child2))`
 - **Parent**: Four-momentum and PDGID (or PID).
 - **Child**: The kinematic differences between the child and the parent (including PDGID and its four-momentum).
- To suppress ambiguity in tree order, sort the children by energy (or p_T).
 - Any other good idea?
- Now the basic structure looks like:
 - `[17, 4, 3, 16, 119, 179, 321, 325, 1050, 1, 4, 3, 13, 798, 975, 933, 1, 4, 3, 13, 798, 999, 857, 1, 4, 3, 14, 510, 579, 681, 1050, 1067, 1, ...]`
 - Including start/end tokens.
- Let's process with a Transformer!

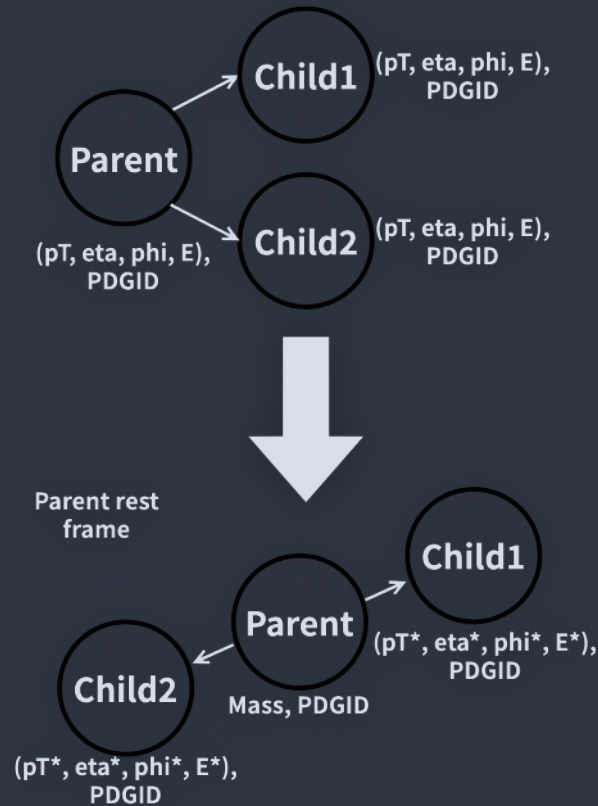
How to Tokenize Jet Constituents : Parent



- Which information is useful to embed into token?
- Let's start with a basic structure `(Parent (Child1) (Child2))`
 - **Parent** : 4-momentum and PDGID(or PID)
 - **Child** : Kinematic difference between parent, PDGID and its 4-mom
- To suppress ambiguity of tree order, sort children by its energy(or pT)
 - Any other good idea?
- Now the basic structure looks like :


```
(<Parent rPt=XX rEta=XX rPhi=XX rE=XX>
  (<Child1 ...> (...))
  (<Child2 ...> (...))
)
```
- **rPt, rEta, rPhi, rE** : relative Pt, Eta, Phi and Energy like ratio or difference
 - A ratio of energy or pT between this and parent(parent's parent)
 - Difference of eta or phi between this and parent(parent's parent)
- Keep variable aornalized quantity
 - There might be better way

How to Tokenize Jet Constituents : Children



- Boost back to parent rest frame, and normalize it using parent's quantity

-

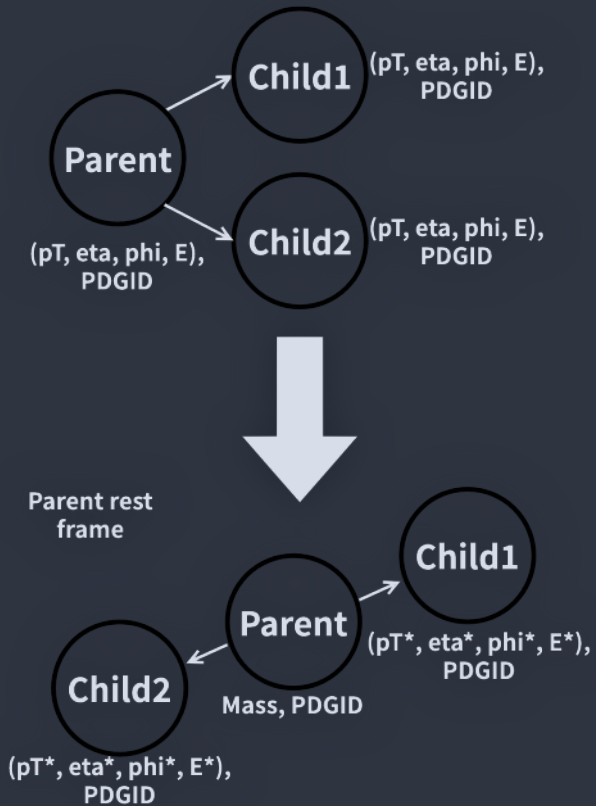
```
(<Parent Pt=XX Eta=XX Phi=XX E=XX>
  (<Child1 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child2 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child3 rEnergy=XX rEta=XX rPhi=XX> (...))
)
```

- Energy conservation : children should satisfy energy conservation
 - Sum of children's momentum and energy should identical value of parent
- Then, e.g. satisfy $rEnergy1 + rEnergy2 + rEnergy3 = 1$, this is a problem
- Network doesn't know which part of children should satisfy energy conservation
 - Here is a simple and strong solution : `left`

```
(<Parent Pt=XX Eta=XX Phi=XX E=XX>
  (<Child1 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child2 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child3 rEnergy=left rEta=left rPhi=left> (...))
)
```

- `left` will make everything fit together.
- Here next problem, if model output $rEnergy1 + rEnergy2 > 1$?

How to Tokenize Jet Constituents : Children



- Here next problem, if model output $rEnergy1 + rEnergy2 > 1$?

```
(<Parent Pt=XX Eta=XX Phi=XX E=XX>
  (<Child1 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child2 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child3 rEnergy=left rEta=left rPhi=left> (...))
)
```

- Leftover Energy**: energy over left energy of parent, $child_i_E / leftover_i_E$

- $leftover_i_E : E_{parent} - \sum_{j=0}^{i-1} E_{child,j}$

- Example : $E_{parent} = 100\text{GeV}, E_{child,0,1,2} = 50, 30, 20\text{GeV}$

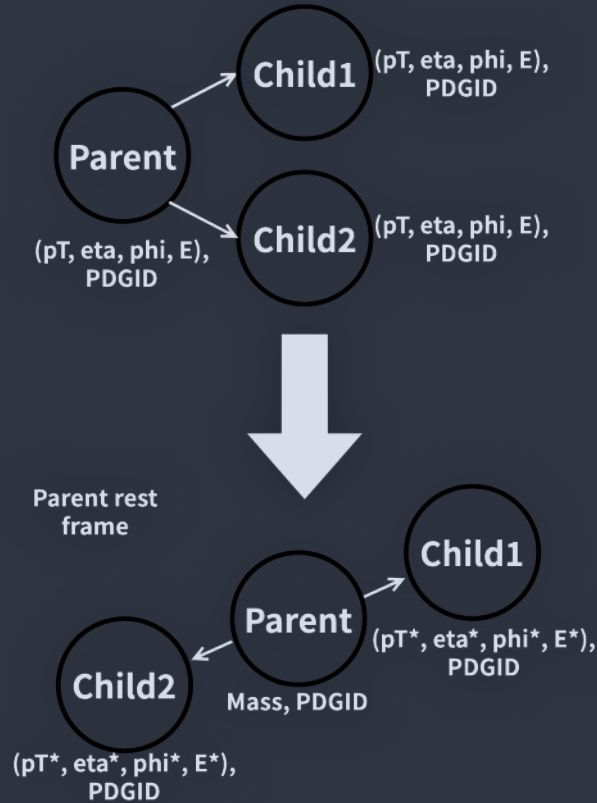
- $leftover_0 = 50 / (100 - 0) = 0.5$

- $leftover_1 = 30 / (100 - 50) = 0.6$

- $leftover_2 = 20 / (100 - 50 - 30) = 1.0 = leftover$

- Using this definition, it is possible to output values in the range 0 to 1 regardless of order.
 - The degree of freedom of the output is very high.

Details : How to Tokenize Jet Constituents



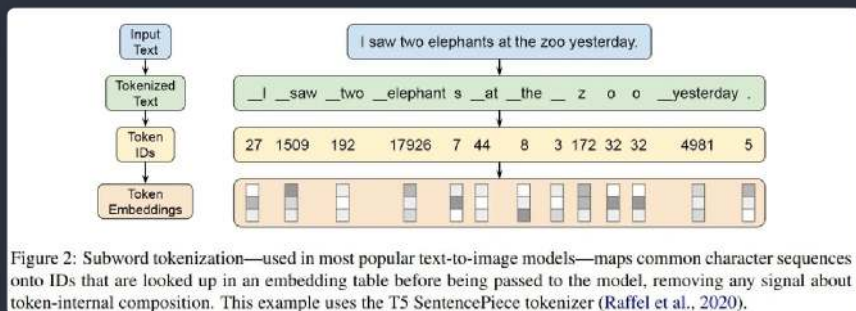
- Current tokenization looks like :

```
(<Parent Pt=XX Eta=XX Phi=XX E=XX>
  (<Child1 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child2 rEnergy=XX rEta=XX rPhi=XX> (...))
  (<Child3 rEnergy=left rEta=left rPhi=left> (...))
)
```

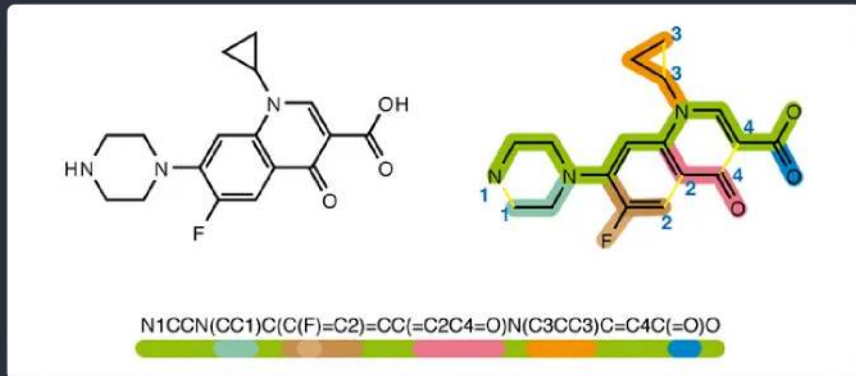
- $PDGID$ (or PID in reconstructed jet) is important part
 - We can directly use $PDGID$ as token, but with different tokenID
 - PID are similar situation
 - (electron, muon, photon, charged/neutral hadron in JetClass)
- Charge is also important for tracks
- Then, textized particle decay looks like

```
(<Parent rPt=.. rEta=.. rPhi=.. rE=.., PDGID=pi0, Ch=0>
  (<Child1 lEnergy=0.5 lEta=.. rPhi=.. PDGID=gamma, Ch=0> (...))
  (<Child2 lEnergy=left lEta=.. rPhi=.. PDGID=gamma, Ch=0> (...))
)
```


How to Divide into Token : Learn from Other Fields



SentencePiece example



SMILES example

SentencePiece :

- Current most popular way to tokenize natural language
- Subword tokenizer that splits text without relying on whitespace, effectively handling languages with unclear boundaries and rare words.
- Statistically divide word into subwords.

SMILES :

- SMILES represents molecular structures as “one-dimensional character strings”, it is possible to treat them like a type of natural language.
- Molecular structure into 1D texts

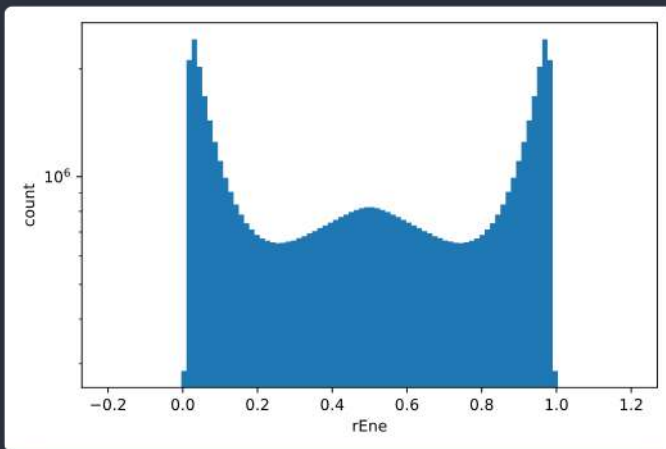
Let's Tokenize

- Example from bottom jet for reconstructed jet

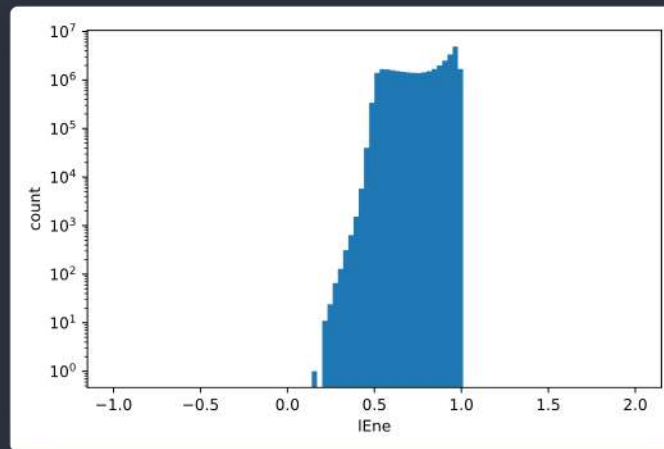
```
[('(', '<', 'T', 'Pt=23.9344', 'Eta=0.408515', 'Phi=3.12396', 'Ene=26.4747', 'Ch=0', '>', '(', '<', 'C0', 'lEne=0.961521', 'lEta=-0.905909', 'lPhi=3.07515', '>', '(', '<', 'P', 'rEne=0.961521', 'rEta=-0.0125', 'rPhi=-0.000453628', 'Ch=0', 'PID=5', '>', '(', '<', 'C0', 'lEne=0.963138', 'lEta=0.961912', 'lPhi=-1.78703', '>', '(', '<', 'P', 'rEne=0.963138', 'rEta=0.00853066', 'rPhi=0.00873275', 'Ch=0', 'PID=5', '>', '(', '<', 'C0', 'lEne=0.926714', 'lEta=-0.709824', 'lPhi=2.0613', '>', '(', '<', 'P', 'rEne=0.926714', 'rEta=-0.0130085', 'rPhi=-0.0126826', 'Ch=0', 'PID=5', '>', '(', '<', 'C0', 'lEne=0.862889', 'lEta=-0.422096', 'lPhi=-2.23227', '>', '(', '<', 'P', 'rEne=0.862889', 'rEta=-0.0104727', 'rPhi=0.0133601', 'Ch=1', 'PID=4', '>', '(', '<', 'C0', 'lEne=0.615834', 'lEta=0.692287', 'lPhi=-1.72699', '>', '(', '<', 'P', 'rEne=0.615834', 'rEta=0.0727701', 'rPhi=0.117426', 'Ch=0', 'PID=5', '>', ')', ')', '(', '<', 'C1', 'lEne=left', 'lEta=-0.692287', 'lPhi=1.4146', '>', '(', '<', 'P', 'rEne=0.384166', 'rEta=-0.123453', 'rPhi=-0.177901', 'Ch=1', 'PID=4', '>', '(', '<', 'C0', 'lEne=0.569539', 'lEta=0.146375', 'lPhi=2.57418', '>', '(', '<', 'P', 'rEne=0.569539', 'rEta=-0.0059779', 'rPhi=-0.0238852', 'Ch=0', 'PID=3', '>', ')', ')', '(', '<', 'C1', 'lEne=left', 'lEta=-0.146375', 'lPhi=-0.567412', '>', '(', '<', 'P', 'rEne=0.430461', 'rEta=0.00782522', 'rPhi=0.0322349', 'Ch=1', 'PID=4', '>', ')', ')', ')', ')', ')', ')', '(', '<', 'C1', 'lEne=left', 'lEta=0.422088', 'lPhi=0.909315', '>', '(', '<', 'P', 'rEne=0.137111', 'rEta=0.0655412', 'rPhi=-0.087044', 'Ch=-1', 'PID=4', '>', ')', ')', ')', ')', '(', '<', 'C1', 'lEne=left', 'lEta=0.709819', 'lPhi=-1.08029', '>', '(', '<', 'P', 'rEne=0.0732861', 'rEta=0.162563', 'rPhi=0.171277', 'Ch=0', 'PID=3', '>', ')', ')', ')', ')', '(', '<', 'C1', 'lEne=left', 'lEta=-0.96191', 'lPhi=1.35457', '>', '(', '<', 'P', 'rEne=0.0368616', 'rEta=-0.221852', 'rPhi=-0.211725', 'Ch=0', 'PID=3', '>', ')', ')', ')', ')', '(', '<', 'C1', 'lEne=left', 'lEta=0.905922', 'lPhi=-0.0664389', '>', '(', '<', 'P', 'rEne=0.0384792', 'rEta=0.337128', 'rPhi=0.0133072', 'Ch=0', 'PID=3', '>', ')', ')', ')']
```

- Top level parent is specially assigned as **T** (**T** as **Top level**)
- Brace (**<** , **>** , **(** , **)**) is one token
- Each variables (e.g. **eEta** or **Pt** etc...) are treated as just one token

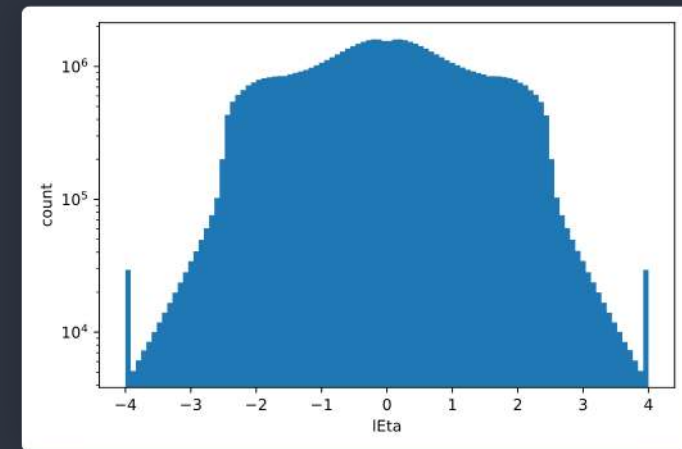
Let's Tokenize : Quantization



Relative Energy



Leftover Energy



Leftover Eta

- It's time to quantize floating variable into quantized integer
- Currently just using histogram style binning.
 - Just apply linear binning for all floating variables with fixed the number of bins
 - Currently just using 100 bins for all variables

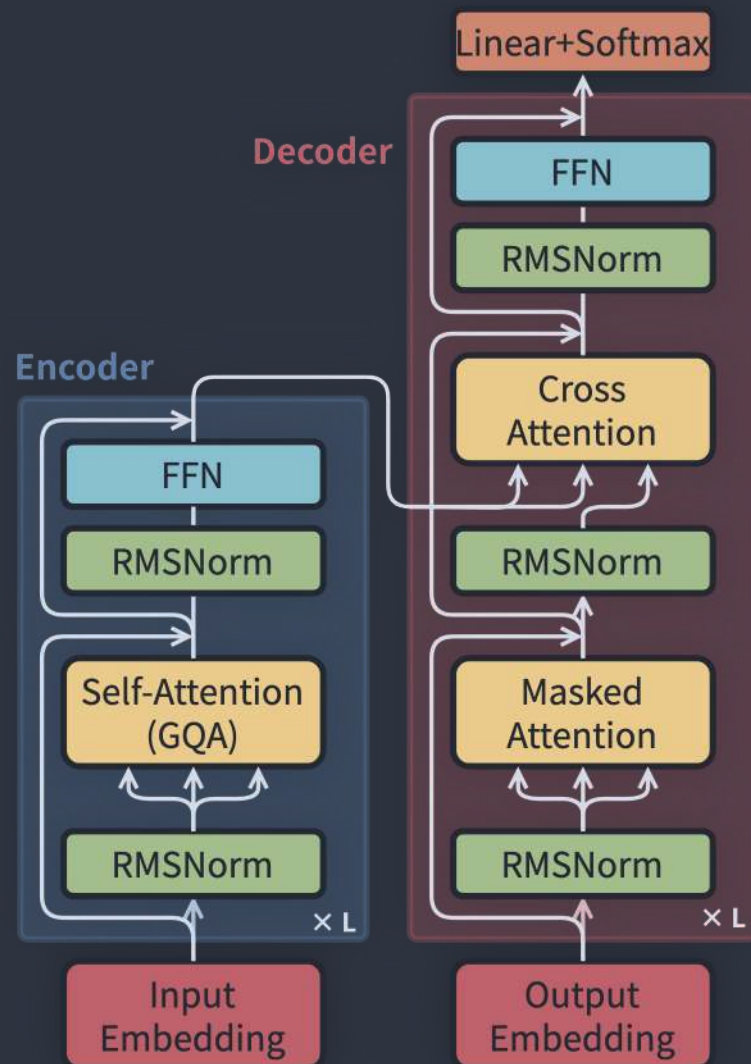
After Tokenization

- Example from bottom jet for reconstructed jet

```
[17, 4, 3, 16, 119, 179, 321, 325, 1050, 1, 4, 3, 13, 798, 975, 933, 1, 4, 3, 14, 509, 577, 680, 1050, 1067, 1, 4, 3, 1, 3, 798, 999, 857, 1, 4, 3, 14, 510, 579, 681, 1050, 1067, 1, 4, 3, 13, 797, 978, 917, 1, 4, 3, 14, 507, 577, 680, 1050, 1067, 1, 4, 3, 13, 795, 981, 850, 1, 4, 3, 14, 502, 577, 681, 1051, 1066, 1, 4, 3, 13, 786, 995, 858, 1, 4, 3, 14, 485, 586, 683, 1050, 1067, 1, 2, 2, 4, 3, 11, 0, 978, 907, 1, 4, 3, 14, 468, 566, 677, 1051, 1066, 1, 4, 3, 13, 785, 988, 925, 1, 4, 3, 14, 481, 578, 680, 1050, 1065, 1, 2, 2, 4, 3, 11, 0, 985, 876, 1, 4, 3, 14, 472, 579, 681, 1051, 1066, 1, 2, 2, 2, 2, 2, 2, 4, 3, 11, 0, 992, 899, 1, 4, 3, 14, 451, 585, 679, 1049, 1066, 1, 2, 2, 2, 2, 4, 3, 11, 0, 995, 868, 1, 4, 3, 14, 446, 595, 684, 1050, 1065, 1, 2, 2, 2, 2, 4, 3, 11, 0, 974, 906, 1, 4, 3, 14, 443, 556, 676, 1050, 1065, 1, 2, 2, 2, 2, 4, 3, 11, 0, 998, 883, 1, 4, 3, 14, 444, 612, 681, 1050, 1065, 1, 2, 2, 2, 18]
```

- Including start/end token
- Let's process with Transformer!

Seq2Seq: Reconstruct into Truth?



- Perform next-token prediction using the encoder output:
- Reconstructed jet text \rightarrow truth jet text, similar to English-to-Korean translation.
- Not sure if it will work or not... but the Transformer works well in chemical tokenization (SMILES).
- Following the original Transformer architecture, but with more modern layers:
 - **Grouped Query Attention**: Can reduce the number of attention heads.
 - **RMSNorm**: Simply normalizes using RMS.
 - **FFN (SwiGLU)**: A Swish-based gated FFN.
 - **RoPE**: Rotary Positional Encoding.

Samples and Distributions

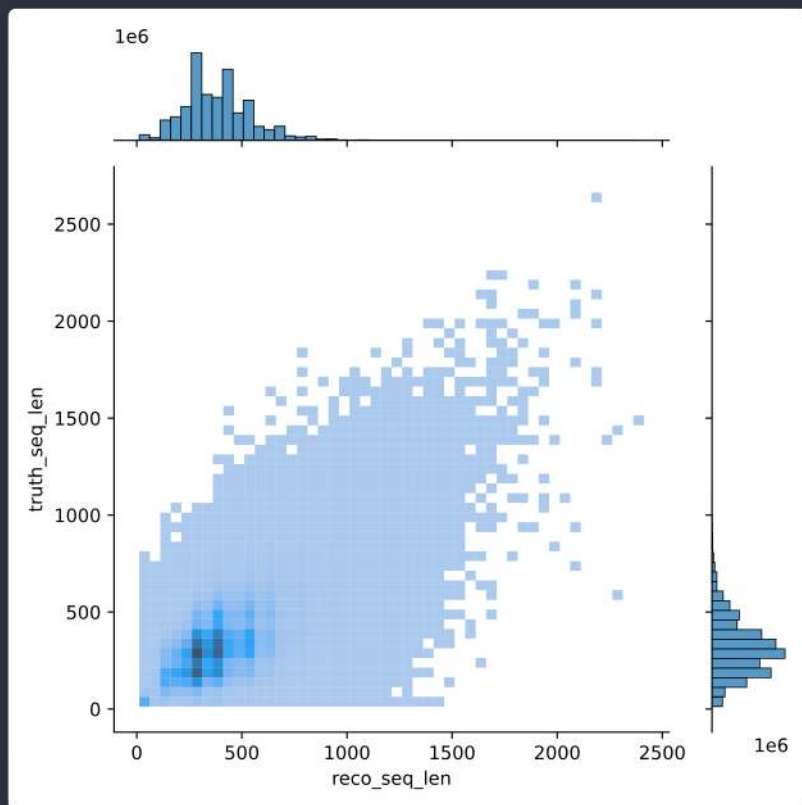
- Generate QCD jets using MG5 + Pythia8:
 - $pp \rightarrow q\bar{q}$, where $q = u, d, s, c, b$
 - $pp \rightarrow gg$
 - $pp \rightarrow Z \rightarrow \tau\tau$
 - $p_T > 20 \text{ GeV}$
- Detector smearing is applied with Delphes using a CMS card taken from the jet-universe repository.
- **Truth Particle Filtering:**
 - If you use the truth decay chain as is, you will have to deal with outputs that are either meaningless or extremely difficult to interpret; therefore, use the status code to slim down the truth particles.
 - Only the truth output that matches the reco jet axis within a certain range is used.
 - These additional selections help reduce complexity and improve learning ability.

Results....?



- I'm very sorry, but I couldn't manage to finish the training...
- This is because the sequence length is really long.
- A huge number of padding tokens consumes a lot of memory in the attention calculations—maybe the sequence length needs to be reduced?

Sequence Length



- 2D plot for the number of tokens in the reconstruction and truth sequences.
- Example: 4 tracks → (A (B (C (D)) (F (G) (H))))
 - Results in at least 21 tokens without node features.
 - Roughly 10–20 times more tokens than typical point cloud representations.
- The maximum for both reaches around 1600 tokens!
- The attention layer needs to handle sequences up to 1600 tokens!
- That's too long...
- Sequence packing is necessary.

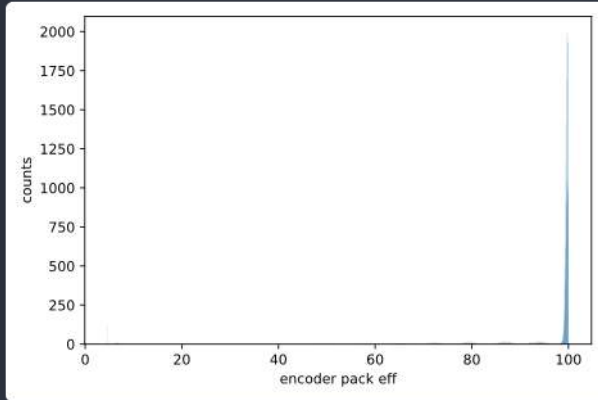
Sequence Packing



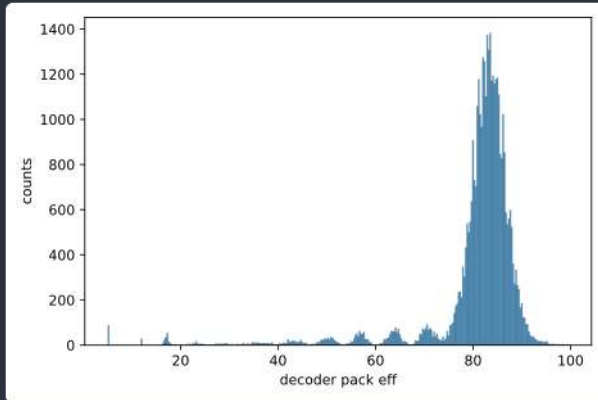
Sequence Packing

- Sequence packing concatenates multiple shorter sequences into one longer batch to reduce padding waste.
- It utilizes special tokens (e.g., [SEP]) and masks so the model can still differentiate individual sequences.
- This technique improves GPU usage and overall training efficiency.
- [PackedBERT](#) proposed a packing algorithm for sequences:
- Shortest-pack-first histogram-packing (SPFHP): Pack starting from the shortest sequence.
- Longest-pack-first histogram-packing (LPFHP): Pack starting from the longest sequence.
- However, this algorithm is designed for 1D sequences, and we need to pack sequences for both the encoder and decoder inputs.

2D Sequence Packing



Packing Efficiency at Encoder



Packing Efficiency at Decoder

2D Sequence Packing

- Test packing with:
- Maximum sequences per pack: 256
- Maximum tokens per pack: $1024 \times 64 = 65\,536$
- Shortest-first packing
- Packing results:
- Number of original jets: 6,441,499
- Number of packs: 38,679 packs
- Average number of jets per pack: 167
- Memory improvement $\rightarrow 1600 / (1024 \times 64) * 167 = 4$
- New questions:
- Can such long sequences be processed in the attention layer?
 - Yes, flash attention can efficiently handle very long sequences.
- How do you manage attention between different sequences?
 - Using attention masks (both causal and padding) along with token IDs can solve this.

Inference

Language/Chemical

- **Greedy Decoding:**
- A simple method where the most probable token is selected one at a time.
- Although it has a low computational cost, it may overlook the optimal solution.
- **Beam Search:**
- This method maintains multiple candidate sequences simultaneously and explores the sequence that has the highest overall probability.
- While it can lead to more refined results, the computational load increases.
- **Sampling Methods (Top-k, Nucleus Sampling):**
- When diversity in the output is desired, methods that incorporate randomness based on probability are also utilized.

Jet Physics

- **Maintaining Correct Syntax:**
- Since particle decay follows strict structural rules, it's important to retain multiple candidate sequences to ensure correct overall syntax.
- Greedy Decoding may lead to local optima and invalid decays.
- **Diversity and Accuracy:**
- Beam Search explores several candidate sequences, potentially yielding more diverse and accurate particle decay structures.
- **Conservation Law Constraints:**
- For decays from a parent to daughter particles, rule-based constraints—such as energy conservation—can be applied.

A lot of things need to be considered...

Conclusion

Conclusion

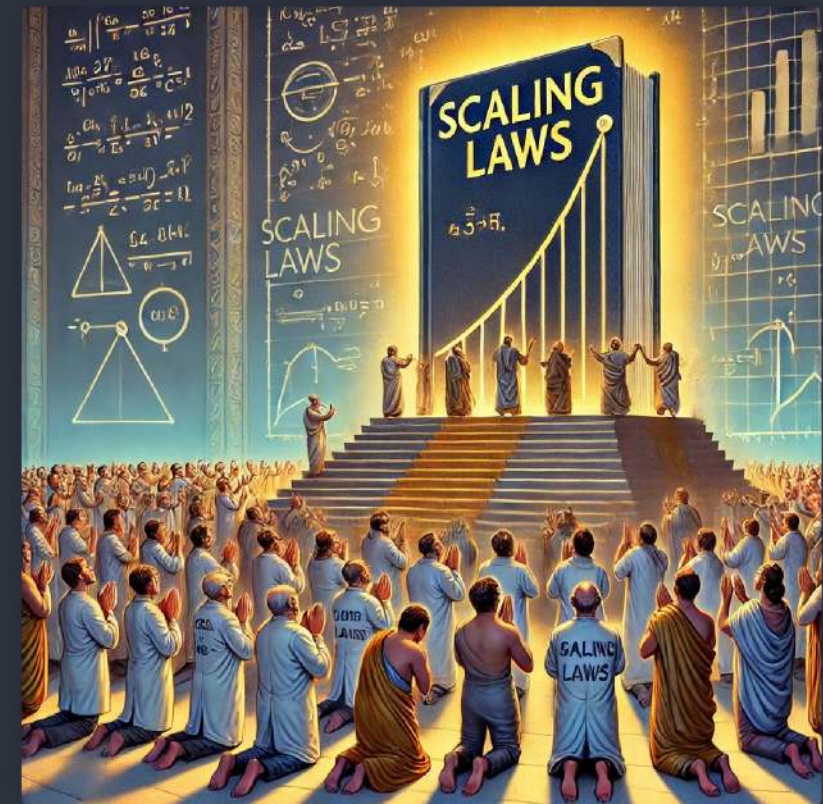
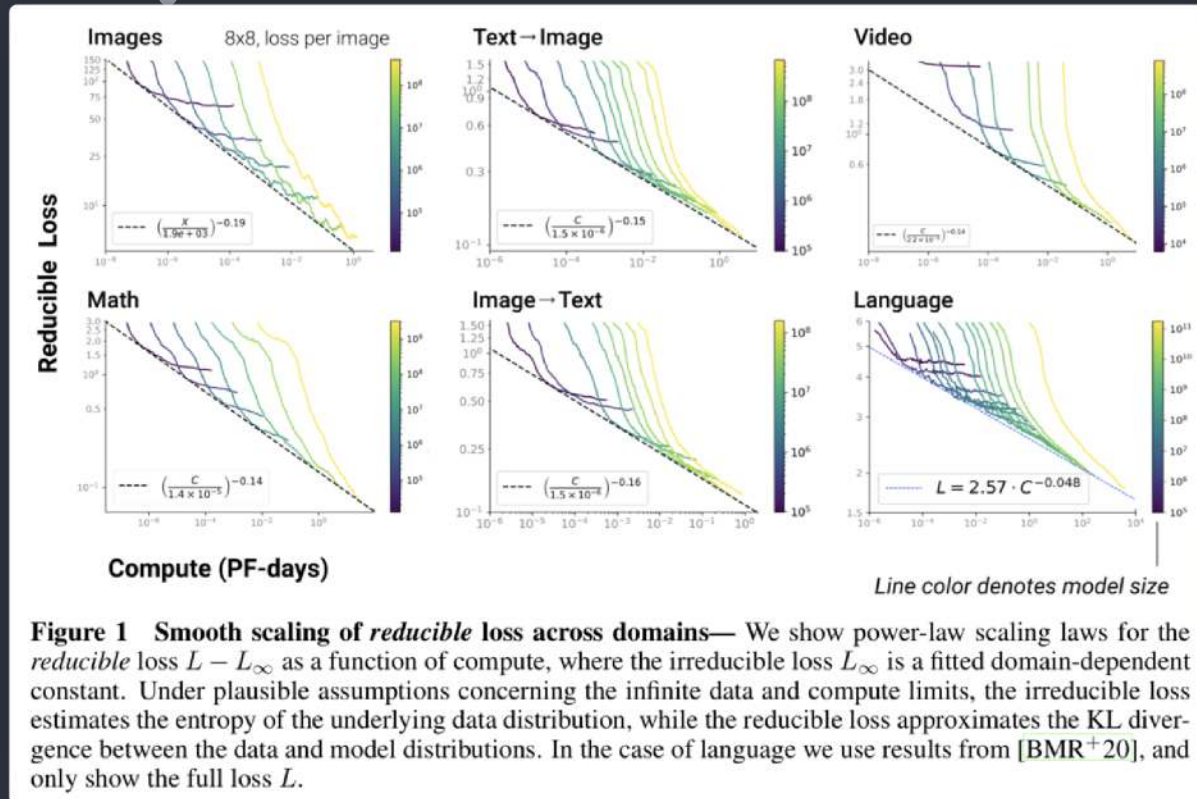
- Jets are important physics objects—not only for understanding QCD but also for searching for or measuring the Higgs boson and other (B)SM particles.
- Most current models treat jet constituents as point clouds.
- Bridging jet ML and language models seems to be a key point.
- Converting tree structures into 1D sequences appears to work.
- If this works well, we can access substructure (or decay chain) information.

Plan

- Successfully complete training.
- Inference generation:
- Beam search? Rule-based methods?
- Pythia or SHERPA...?
- Scaling the model is always the answer.

backup

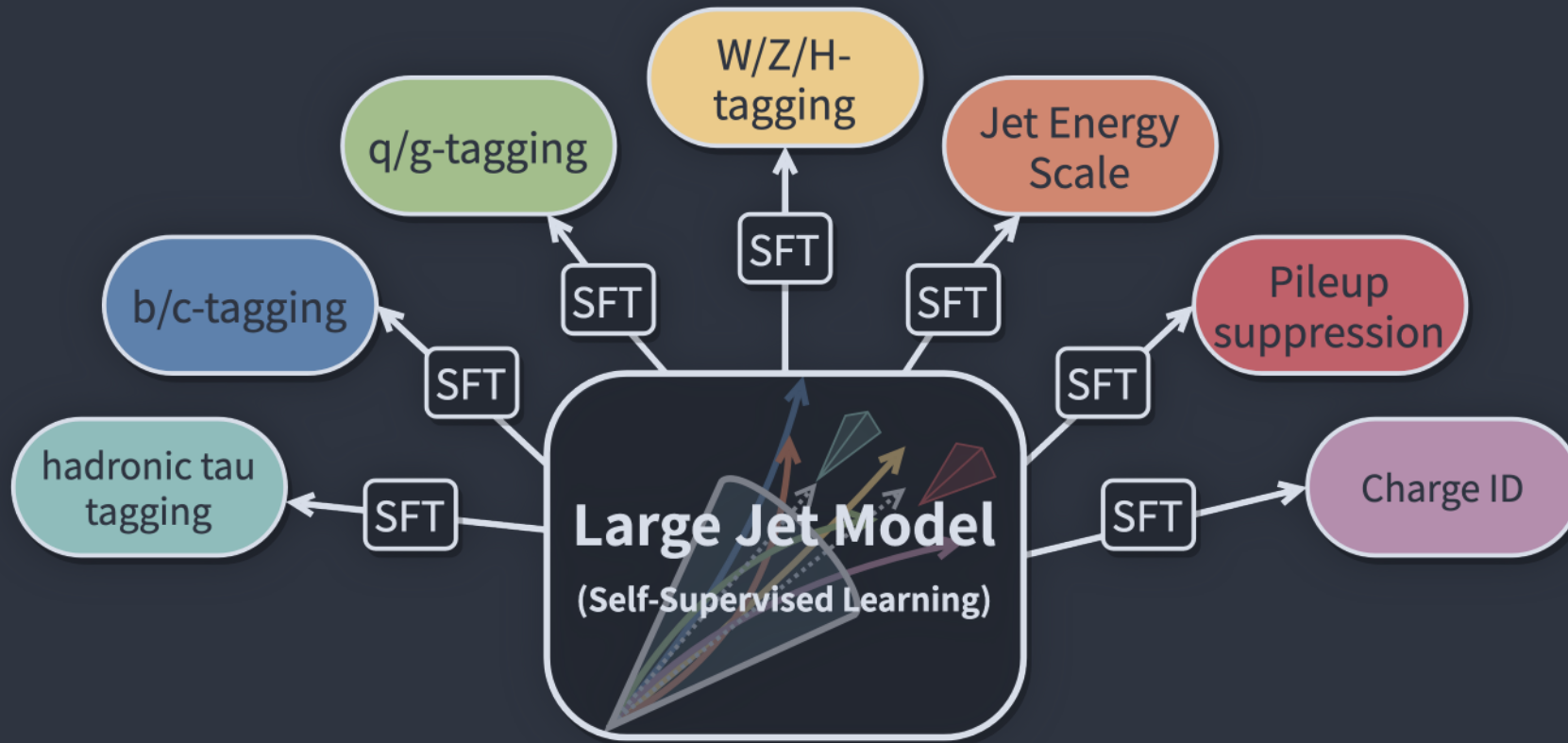
Scaling Law for Transformer



Scaling Law: Current AI Principle

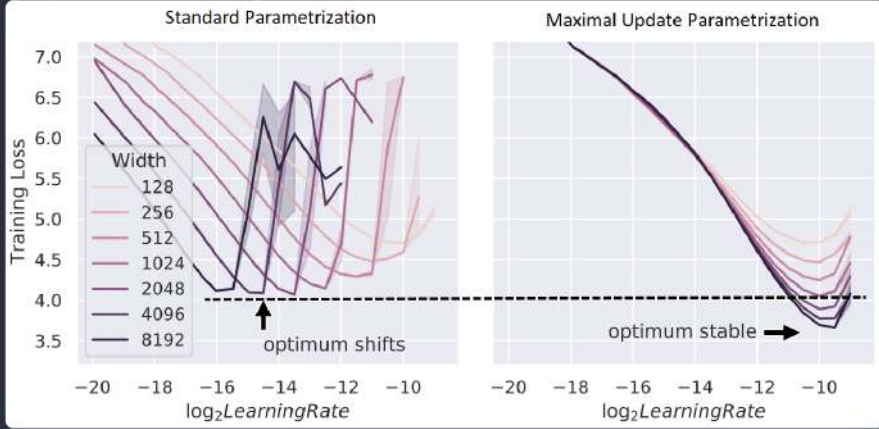
- The neural network loss will continue to decrease as the number of model parameters, the amount of training data, and the training time increase.
- This trend is observed not only in language but also in other fields—does it hold for particle physics as well?
- Do larger models yield better performance in jet physics?

(Realistic) Foundation Model for Jet-Related Tasks

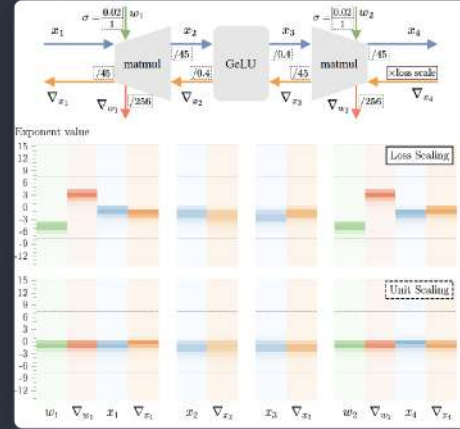


- At the LHC, there are numerous tasks related to jets, but each uses its own models and data.
- Larger models require more data, and training becomes more challenging.
- Pre-training that can generalize to various tasks is important.
- So, how can we train such a model?

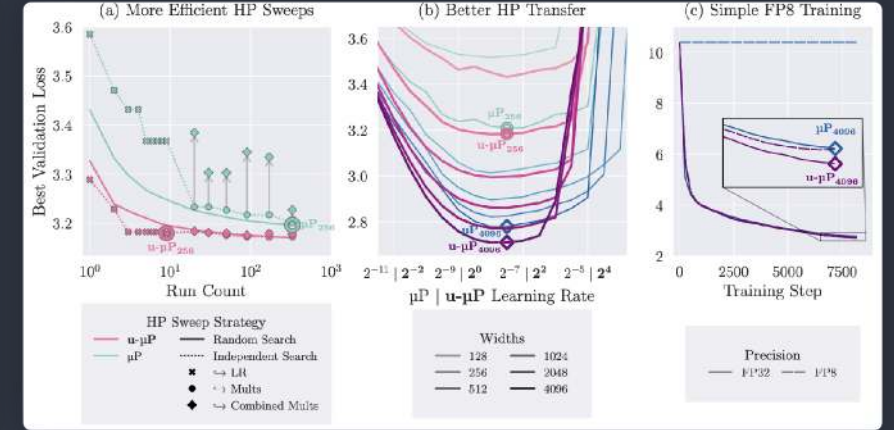
μ Transfer: μ P and u - μ P



μ P



unit-scaling



u - μ P

- **Maximal Update Parametrization (μ P):**
- Controls activations so that the optimal learning rate does not depend on model width.
- Adjusts the initial weights and introduces weight-wise learning rates.
- **Unit Scaling:**
- Controls the deviation of network activity to enable low-precision learning.
- Adjusts the initial values and the values for forward and backward propagation of gradient updates using fan-in/fan-out so that the activity falls within the valid range for FP8.
- **Unit scaling μ P (u - μ P):**
- Combines μ P and unit scaling.
- Adjusts the activation values so they fall within the valid range in FP8 and are not dependent on the width and depth of the model.
- Controls the variance of all activation values to 1 using different scaling for forward and backward propagation.