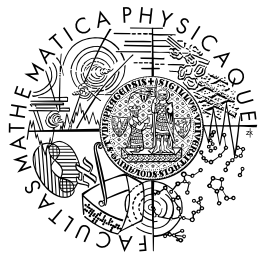


FACULTY
OF MATHEMATICS
AND PHYSICS
Charles University

BACHELOR THESIS

Samuel Jankovych

Quark/Gluon Jet Tagging

Institute of Particle and Nuclear Physics

Supervisor of the bachelor thesis: Mgr. Vojtěch Pleskot, Ph.D.
Study programme: Physics
Study branch: Physics

Prague 2023

I declare that I carried out this bachelor thesis independently, and only with the cited sources, literature and other professional sources. It has not been used to obtain another or the same degree.

I understand that my work relates to the rights and obligations under the Act No. 121/2000 Sb., the Copyright Act, as amended, in particular the fact that the Charles University has the right to conclude a license agreement on the use of this work as a school work pursuant to Section 60 subsection 1 of the Copyright Act.

In ..... date ....

Author's signature

First and foremost, I want to thank my supervisor, Mgr. Vojtěch Pleskot, Ph.D., for his everlasting support. No matter the idea, he always helped and furbished it so that it made sense and we could implement it. I could not wish for a kinder and more dedicated supervisor. I thank the MFF for allowing me to use the GPULab with all the necessary computational infrastructure.

Title: Quark/Gluon Jet Tagging

Author: Samuel Jankovych

Institute: Institute of Particle and Nuclear Physics

Supervisor: Mgr. Vojtěch Pleskot, Ph.D., Institute of Particle and Nuclear Physics

Abstract: Distinguishing between jets initiated by quarks and gluons is a challenging problem, yet very important for detailed studies of elementary particles and their interactions. This thesis will present a novel Deep Learning approach to this problem using a neural network architecture based on the Transformer model trained on the jet constituents. We improve the existing architectures used in different tagging tasks and show their ability to accurately distinguish between quark and gluon jets. By combining techniques from different fields of Deep Learning, we propose a Dynamically Enhanced Particle Transformer (DeParT) that can surpass the state-of-the-art results in the quark/gluon jet tagging task.

Keywords: ATLAS Jet Neural Networks Transformer Deep Learning

Contents

Introduction

Particle physics is a branch of physics that studies the fundamental constituents of matter and their interactions. To understand the fundamental processes that govern physical reality, we must identify all the participants in experiments that probe them.

In this thesis, we focus on tagging jets initiated by quarks or gluons. To do the identification, theory, experimental devices, and measured data must be understood in the context of particle identification.

Two fundamental particles that we are mainly interested in are *quarks* and *gluons* [1], together called *partons*. The physical theory that describes them is *Quantum Chromodynamics* (QCD) , which is inherently extremely complicated, far more than the theory of *Electromagnetism*. The less energetic partons are, the stronger the interactions between them are. This creates a problem in their description as they tend to radiate and split into more partons, which makes it hard to trace them back to their origin. The developed spray of particles is called *jets*. We discuss the physics of partons in more detail in chapter 1.

To produce these jets (and many other particles), the *European Organization for Nuclear Research* (CERN) [2] built the *Large Hadron Collider* (LHC) [3], which is the biggest particle accelerator in the world. On the LHC is the *ATLAS* [4] detector, a giant experiment testing our knowledge of elementary interactions. The LHC accelerates two proton beams in different directions, so they can collide at the interaction points and create a variety of particles. The ATLAS detector, located at one of these interaction points, then detects these particles. The two main components of ATLAS crucial to our study are the *tracker* and the *calorimeters*. They provide enough data to reconstruct the jets and their constituents. A more detailed description of the ATLAS detector is in chapter 2.

The definition of a jet is very complicated. It depends on many aspects: the detector, collisions, jet algorithm, background events, secondary collisions (pileup), and many more. Our study uses the anti-k_t algorithm [5] that clusters several constituents into a jet. The constituent is an energy deposit in the calorimeter or, if it is charged, a track in the tracker. The algorithm that clusters energy deposits and tracks into constituents we use is the *Particle Flow* algorithm (PF algorithm)

[6]. It provides essential input to the jet algorithms and our taggers.

However, to train the neural network we also need the truth information, which cannot be measured but is rather simulated with the *Monte Carlo* (MC) simulations. The MC includes both the physical simulation of the collision and the detector response. In chapter 3, we provide more detail.

Quark-gluon taggers could be used in a variety of applications, such as the search for new physics and precise measurements of the Standard Model. To give a concrete example, the detector responses to quarks and gluons differ, so a quark/gluon tagger could be used to do separate calibrations for the two. In *Vector Boson Fusion* or *Vector Boson Scattering* analyses a well-performing quark tagger would allow for a better event selection. The SUSY gluino multijet search [7] could benefit from a quark-gluon tagger. Moreover, the techniques developed in this thesis can be used in other tagging problems, such as the tagging of jets into different classes [8].

Our goal, telling apart quark/gluon jets, is non-trivial as traces left by these particles in the detector are very similar. In the past, hand-crafted variables (such as a number of hadron tracks detected in the jet) describing the jets were used [9]. Nowadays, modern *Machine Learning* techniques allow the utilization of more variables. The most adopted and successful is the *Boosted Decision Trees* (BDT) algorithm [10], which is still being developed [11]. It uses multiple *high-level jet variables* (variables that describe the jet as a whole) as input.

In this thesis, we approach the problem using modern *Deep Learning* techniques. Deep Learning has exploded in the last decade. From the *Natural Language Processing* (NLP) [12] to image recognition [13], Deep Learning has been used to solve many problems. Most notably, the GPT models [14, 15] shook the whole world with their capabilities of text generation, as they can solve complex problems and even supersede humans in exams. Or the image-generating models such as Stable Diffusion [16], or Dall-E [17] that can generate any image from a text prompt. We utilize the Deep Learning , used in these world-changing models, to solve the problem of quark/gluon tagging. An in-depth explanation of different Deep Learning techniques and models is in chapter 4.

Different architectures are explored. Firstly the *Fully Connected Network* (FC) and *Highway Network* architectures [18, 19] are developed, which use high-level jet variables, but considerably more than BDT . Afterward, we turn to a more granular jet description, using *jet constituents*. The *Energy Flow Network* (EFN) and *Particle Flow Network* (PFN) [20] are simple architectures that use the jet constituents as an input developed by the *High Energy Physics* (HEP) community.

'Attention is all you need' [21] paper introduces a highly successful and influential architecture, called *Transformer* , used in all kinds of applications, most notably NLP [12, 14, 15, 13, 22]. It can learn long-range dependencies and generalize its knowledge to new tasks. In HEP community, its use is just starting.

For example, in [23], it is used to teach the model a jet structure as a NLP task. Or recently, it was explored by the *Compact Muon Solenoid* (CMS) [24] Collaboration, where they were tagging jets into various classes [8]. We improve on their work and apply it specifically to the problem of quark/gluon tagging. The models are trained, studied comprehensively, and their performance is compared in chapter 5.

Chapter 1

Physics of Quarks and Gluons

1.1 The Standard Model

The best widely adopted and tested description of the fundamental particles and their interactions is the *Standard Model* (SM) [25]. *Fermions* are particles

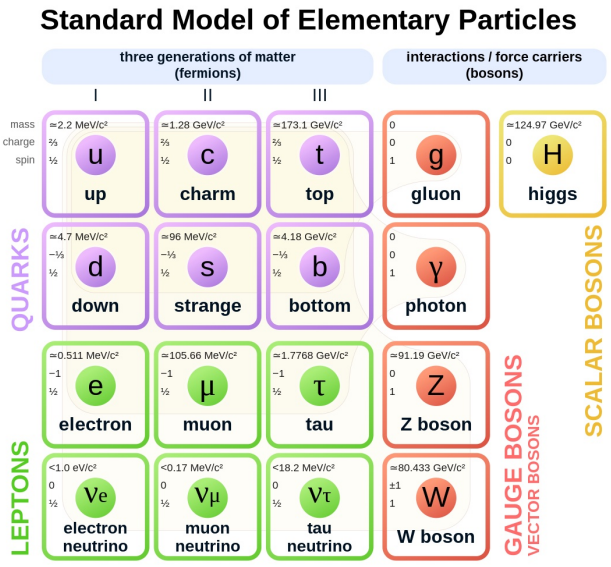


Figure 1.1 The Standard Model of particle physics. [1]

that makeup all the matter and *bosons* are force-mediating particles. All known elementary particles are displayed in figure 1.1.

[1] `https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary(P)articles.svg`

Fermions are divided into two categories, *leptons* and *quarks*. Leptons are further divided into two groups:

1.
   - *electron*, a particle that is present in all matter around us,
   - *muon*, a heavier cousin of electron naturally produced in the atmosphere,
   - *tau lepton*, the heaviest lepton.

   These three leptons can interact via the *Electromagnetic force* (they have a charge), mediated by *photons*, or via the *Weak force*, mediated by *W and Z boson*.

2. *Neutrinos*, each belonging to either electron, muon, or tau lepton. Neutrinos can interact only weakly by exchanging W and Z bosons.

Quarks are also divided into two groups:

1. With electric charge $\frac{2}{3}$ [2],
   - *up*, u quark,
   - *charm*, c quark,
   - *top*, t quark.
2. With electric charge $-\frac{1}{3}$.
   - *down*, d quark,
   - *strange*, s quark,
   - *bottom*, b quark.

All quarks can interact not only weakly and electromagnetically, but also via the *Strong force*, mediated by *gluons*. In nature, practically only the u-quark and d-quark occur. Concretely, they are the building blocks of protons and neutrons.

Fermions can be also split another way, into three families (*generations*). The mass of corresponding particles gives the order of families. The lightest is built up by a u-quark, a d-quark, an electron, and an electron neutrino. Practically all matter around us is made up of particles from this family. It should be noted that neutrino masses are not precisely known. Due to a measured process [26] called neutrino oscillations [25], they cannot be massless. Since the first discovery, estimates on the upper bound of their masses were made [25].

The bosonic particles are a photon, gluon, W, and Z boson, and *Higgs boson*. Vector bosons ('vector' comes from their mathematical description) are force carriers:

- *Electromagnetic force* - photon,
- *Strong force* - gluon,

[2] Measured in units of elementary charge $e = 1.60217663 \cdot 10^{-19}$ C.

- *Weak force* - Z and W boson.

Higgs boson is a scalar boson, giving mass to other particles. It is an artifact of the Higgs mechanism that introduces masses to particles while preserving the *local gauge invariance*. The properties of the corresponding force-carrying boson give the nature of each force. The corresponding charge of interacting particles gives the interaction strength of each force.

Particles that have electric charge can exchange photons and interact electromagnetically. The infinite range of the Electromagnetic force is due to the massless photons. On top of that, the photon has no charge, which makes them unable to scatter on other photons. [3]

Quarks are the only fermion carriers of the *strong charge* (also called *color charge*). In contrast to electromagnetism, gluons have *two strong charges*. They can interact with other gluons and exchange color charges when interacting with quarks or gluons. This differs from Electromagnetic interaction, where particles' charge does not change. Even though the Strong interaction is theoretically infinite, we do not see free gluons or quarks due to a process called *hadronization* (see section 1.5). They only occur in a bound state in nature, effectively making the color charge invisible at a larger scale (it becomes visible at the scale of approximately 1 fm).

All fermions can interact weakly, but since W and Z bosons have mass (which is significant compared to fermions), the range of interaction is short. Weak processes occur, for example, in the β-decay.

Last but not least is the Higgs boson, a scalar boson. The strength of the interaction with the Higgs boson (more precisely: with the scalar field, Higgs field) is given by the mass of the interacting particle. In other words, particles' interaction with the Higgs field gives them mass. Gluons and photons are the only particles from SM non-interacting with the Higgs boson.

1.2 Quantum Field Theory

Quantum Field Theory (QFT) is a fundamental framework in modern theoretical physics that describes the behavior of particles as fields at the quantum level. Paul Dirac did the first development by studying the electromagnetic interaction of particles and photons [27]. Massive progress in *Quantum Electrodynamics* (QED) did Richard Feynman [28], who also introduced the *Feynman diagrams*, a graphical representation of interactions. We will discuss Feynman diagrams in more detail in section 1.2.4.

[3] This has some caveats at higher energies. Classically it is forbidden due to the linearity of Maxwell's equations, but at higher orders of perturbation QFT, photons *can* scatter on another photon. However, the probability of such an event is very low.

The starting point of QFT is that particles are not fundamental objects but excitations of underlying quantum fields that permeate all of space and time. This field must satisfy the equations of motion of the corresponding field theory derived from the *Lagrangian*.

1.2.1 Free Particles

The Lagrangian of non-interacting spin-$\frac{1}{2}$ particle (fermion) of mass m, also called the *Dirac Lagrangian*, is given by

$$\mathcal{L}_{\text{Dirac}} = \bar{\psi}(i\partial\!\!\!/ - m)\psi, \tag{1.1}$$

where we abbreviate $\gamma^\mu \partial_\mu \equiv \partial\!\!\!/$; γ^μ are the Dirac matrices, and ∂_μ is the covariant derivative. $\psi(x)$ is a bispinor fermionic field of a spin-$\frac{1}{2}$ particle. It has four components, which can be split into two spinors, each corresponding to a different *chirality*

$$\psi(x) = \begin{pmatrix} \psi_L(x) \\ \psi_R(x) \end{pmatrix}. \tag{1.2}$$

Going further we will always abbreviate $\gamma^\mu V_\mu \equiv V\!\!\!/$, where V_μ is a 1-form, or a vector with lowered indices $V_\mu = \eta_{\mu\nu} V^\nu$ with the Minkowski metric $\eta_{\mu\nu} = \text{diag}(1, -1, -1, -1)$. We also automatically sum over repeated indices, utilizing the Einstein summation convention.

The Dirac equation is an equation of motion derived from the Lagrangian equation (1.1):

$$\left(i\gamma^\mu \partial_\mu - m\right)\psi(x) = 0. \tag{1.3}$$

It can be solved analytically. The solutions to this equation describe how fermion particles propagate in space and time without interaction.

Conversely, bosons are spin-1 particles, described by the vector field $A_\mu(x)$. The Lagrangian of a non-interacting spin-1 particle of mass m is given by a *Proca Lagrangian*

$$\mathcal{L}_{\text{Proca}} = -\frac{1}{4} F_{\mu\nu} F^{\mu\nu} + \frac{1}{2} m^2 A_\mu A^\mu, \tag{1.4}$$

where $F_{\mu\nu} = \partial_\mu A_\nu - \partial_\nu A_\mu$ is the *field strength tensor*. Note that this Lagrangian also describes massive bosons. In the case of massless bosons, the second term is omitted.

The equation of motion for the vector field, the *Klein-Gordon equation*, follows from the Lagrangian equation (1.4)

$$\left(\Box + m^2\right) A_\mu(x) = 0. \tag{1.5}$$

The solutions to this equation can also be found analytically. In the case of massless bosons, the equation reduces to

$$\Box A_\mu(x) = 0, \tag{1.6}$$

which is a simple wave equation. We can rewrite it using the field strength tensor as [4]

$$\Box A_\mu(x) = \partial_\nu \partial^\nu A_\mu(x) = \partial_\nu \partial_\nu A_\mu(x) - \partial_\nu \partial_\mu A_\mu(x) = \partial_\nu F_{\nu\mu}(x) = 0, \tag{1.7}$$

which are the *Maxwell's equations*. This means that massless non-interacting bosons propagate the same way as electromagnetic waves.

1.2.2 Interacting Particles

We introduce the interaction between fields by invoking the *local gauge invariance* of the Lagrangian. Each type of interaction (strong, electromagnetic, weak) corresponds to a different $SU(N)$ *Lie group* [29].

Elements of $SU(N)$ are special unitary NxN matrices [5]. When operating on N-dimensional space of fields, their *representations* are unitary operators in the form

$$U = \mathrm{e}^{ig \sum_a \alpha^a T^a}, \tag{1.8}$$

where T^a are *generators* of the group, and α^a and g are the parameters of the transformation. For example, in the case of a rotating spinor in classical quantum mechanics, the group is the rotation group $SU(2)$, and the generators are the Pauli matrices $T^a \equiv \frac{1}{2}\hat{\sigma}_a$ ($a = x, y, z$). g would be the angle of rotation, and $\alpha^a \equiv \vec{n}$ is the axis of rotation. More generally, T^a can be represented as NxN matrices, where N is the field space dimension. Going further, we will suppress writing the summation over a (or any other index) explicitly unless it is necessary for clarity.

The generators satisfy the commutation relations

$$[T^a, T^b] = i f^{abc} T^c, \tag{1.9}$$

where f^{abc} are the *structure constants* of the Lie group.

We can now connect the Lie group with physics; a Lagrangian describes an interacting physical system if it is invariant under the local gauge transformation

$$\psi'_j(x) = U_{ij}(x)\psi(x)_j = \mathrm{e}^{ig\alpha^a(x)T^a_{ij}}\psi(x)_j, \tag{1.10}$$

[4] We have utilized the Lorentz gauge $\partial_\mu A_\mu(x) = 0$.

[5] Matrix U is special if det$U = 1$ and unitary if $UU^\dagger = U^\dagger U = 1$

of the spinor quantum fields $\psi_i(x)$, and under the infinitesimal transformation of the vector fields $A^a_\mu(x)$

$$A^a_\mu(x)' \to A^a_\mu(x) + \frac{1}{g}\partial_\mu \alpha^a(x) - f^{abc}\alpha^b(x)A^c_\mu(x) + \mathcal{O}((f^{abc})^2), \tag{1.11}$$

where $U(x)$ is an element of the corresponding Lie group for every spacetime point x. g is the coupling constant of the interaction, and $\alpha^a(x)$ are the parameters of the transformation.

For example, the electromagnetic interaction corresponds to the $U(1)$ group (Lie group isomorphic to complex numbers), which means that the Lagrangian must be invariant under the transformations

$$\psi'(x) = \mathrm{e}^{-ie\alpha(x)}\psi(x) \quad , \quad A'_\mu(x) = A_\mu(x) + \frac{1}{e}\partial_\mu\alpha(x). \tag{1.12}$$

where the a generator of the $U(1)$ group is just a constant $T^a = 1$, and the structure constants are zero $f^{abc} = 0$.

In the general case of the non-Abelian $SU(N)$ group having N different fermions with the free Lagrangian

$$\mathcal{L}_{\text{Dirac}} = \sum_{i=1}^{N} \bar{\psi}_i(i\partial\!\!\!/ - m)\psi_i, \tag{1.13}$$

the local gauge invariant Lagrangian is given by [30]

$$\mathcal{L} = -\frac{1}{4}G^a_{\mu\nu}G^a_{\mu\nu} + \frac{1}{2}m^2A^a_\mu A^a_\mu, + \sum_{i,j=1}^{N} \bar{\psi}_i(i\delta_{ij}\partial\!\!\!/ + gA\!\!\!/^a T^a_{ij} - m\delta_{ij})\psi_j. \tag{1.14}$$

where the field strength tensor is modified, as a result of the non-commutativity of the generators, to

$$G^a_{\mu\nu} = F^a_{\mu\nu} + gf^{abc}A^b_\mu A^c_\nu = \partial_\mu A^a_\nu - \partial_\nu A^a_\mu + gf^{abc}A^b_\mu A^c_\nu, \tag{1.15}$$

such that it preserves the local gauge invariance. This additional term significantly impacts the system's dynamics because the last term of 1.15 corresponds to boson-boson interaction. We shall see in section 1.3 the consequences of this interaction.

If we go back to the example of $U(1)$, the corresponding Lagrangian has the form [6]

$$\mathcal{L} = -\frac{1}{4}F_{\mu\nu}F_{\mu\nu} + \bar{\psi}(i\partial\!\!\!/ - m)\psi - e\bar{\psi}\gamma^\mu\partial_\mu\psi A_\mu. \tag{1.16}$$

The equations of motion of the Lagrangian (1.16) ultimately determine the system's dynamics.

[6] $G^a_{\mu\nu} \equiv F^a_{\mu\nu}$, because $f^{abc} = 0$

1.2.3 Cross Section

In particle physics, we are not interested in calculating the actual state $\psi(x)$ as a solution to the equations of motion of the Lagrangian (1.16), but rather in the probability of particles interacting. The physical quantity describing this probability is the cross section σ. With the probabilistic nature of quantum mechanics, we need to define this quantity based on multiple interactions. Suppose we have a flux of incoming particles scattered off target particles. Then the probability that a given particle will scatter off of a target particle is given as

$$\sigma = \frac{\text{number of scattered particles}}{\text{incoming flux of particles}}, \tag{1.17}$$

where the flux has units m^{-2} so the cross section has units m^2.

If we want to specify the infinitesimal probability of particles scattering to a space angle $\mathrm{d}\Omega$, we can define it as

$$d\sigma = \frac{\text{number of scattered particles to an space angle } d\Omega}{\text{incoming flux of particles}}. \tag{1.18}$$

Or, if we want the cross section in terms of the momentum of the scattered particles $\mathrm{d}p$, we can define it as

$$d\sigma = \frac{\text{number of scattered particles with outgoing momentum } dp}{\text{incoming flux of particles}}. \tag{1.19}$$

Quantum mechanically, we expect the cross section to be given by the probability of the system being in a final state $|f\rangle$ given that it evolved from the initial state $|i\rangle$. We can express this as

$$\sigma \sim |\langle f|\mathrm{e}^{-i\hat{H}(t_{out}-t_{in})}|i\rangle|^2, \tag{1.20}$$

where $\hat{H}$ is the Hamiltonian of the system, and t_{in} and t_{out} are the initial and final times of the process.

To be more precise, for two interacting particles with energies $E_{1,\text{in}}$, $E_{2,\text{in}}$ and velocities $\vec{v}_{1,\text{in}}$, $\vec{v}_{2,\text{in}}$, and N particles in the final state with energies $E_{j,\text{out}}$ ($j = 1, 2, ..., N$), we can write [30]

$$d\sigma = \frac{|\mathcal{M}|^2}{4E_{1,\text{in}}E_{2,\text{in}}|\vec{v}_{1,\text{in}} - \vec{v}_{2,\text{in}}|}(2\pi)^4\delta^4(p^\mu_{\text{in}} - p^\mu_{\text{out}})\prod_{j=1}^{N}\frac{d^3p_j}{(2\pi)^3 2E_{j,\text{out}}}, \tag{1.21}$$

where p^μ_{in} is the total 4-momentum of the incoming particles and p^μ_{out} is the total 4-momentum of the outgoing particles, the $\delta^4(p^\mu_{\text{in}} - p^\mu_{\text{out}})$ function is the 4-dimensional Dirac delta function enforcing 4-momentum conservation, and the

$|\mathcal{M}|^2$ is the squared matrix element of the *scattering amplitude* $\mathcal{M}$. The matrix element $\langle f|\mathcal{M}|i\rangle$ (where $|\mathcal{M}|^2 \equiv |\langle f|\mathcal{M}|i\rangle|^2$) is given by (assuming $|f\rangle \neq |i\rangle$)

$$\langle f|S|i\rangle = i(2\pi)^4\delta^4(p^\mu_{\text{in}} - p^\mu_{\text{out}})\langle f|\mathcal{M}|i\rangle, \tag{1.22}$$

where S is the S-matrix of the scattering process. We can calculate the S-matrix utilizing the Dyson series expansion [30]

$$\langle f|S|i\rangle = \left\langle f \left| \sum_{n=0}^{\infty} \frac{(-i)^n}{n!} \int dx_1^4 \cdots \int dx_n^4 T\left[\mathcal{H}\left(t_1\right) \cdots \mathcal{H}\left(t_n\right)\right] \right| i \right\rangle, \tag{1.23}$$

where $T\left[\mathcal{H}\left(t_1\right) \cdots \mathcal{H}\left(t_n\right)\right]$ is the time-ordered product of the Hamiltonian $\mathcal{H}$ of the system at times $t_1, t_2, ..., t_n$. Equation (1.20) is a particular case of equation (1.23) where the Hamiltonian is time-independent.

1.2.4 Feynman Diagrams

Calculating the expansion (1.23) is practically impossible analytically, so R. Feynman developed a graphical method to calculate individual terms [28]. The diagrams consist of the following:

- **External lines** represent the incoming and outgoing particles, which are *real* particles,
- **Internal lines** represent the intermediate particles, which are *virtual* particles,
- **Vertices** are the points of interaction.

Virtual particles are not real particles but a mathematical construct that allows us to visualize the scattering process. Each part contributes a multiplicative factor to the amplitude $-i\mathcal{M}$. The *order of the diagram*, the order of the corresponding term in the Dyson series expansion 1.23, is given by the number of interaction vertices.

To give an example, we consider electron-muon scattering. The second-order Feynman diagram for this process, shown in figure 1.2, is the lowest-order diagram that can contribute to the cross section.

The time flows from left to right, so we have an incoming electron and muon. They exchange a virtual photon, momentum, and scatter. The incoming particles, with momenta p_1, p_2, contribute factors $u(p_1)$, $u(p_2)$, which are solutions to the free Dirac equation (1.3) in momentum space. The outgoing particles, with momenta p_3, p_4, contribute factors $\bar{u}(p_3)$, $\bar{u}(p_4)$, which are complex conjugates

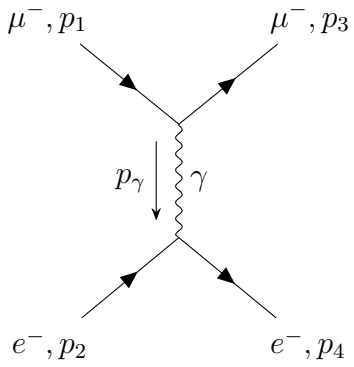


Figure 1.2 Second-order Feynman diagram of electron-muon scattering.

of the free Dirac equation solutions. The vertex is where the interaction happens, and the corresponding term $ie\gamma^\mu$ is given by the interaction Lagrangian $\mathcal{L}_{\text{int}} = -e\bar{\psi}\gamma^\mu\partial_\mu\psi A_\mu$. Virtual photon, with momentum p_γ, is an internal line, its contribution is $\frac{-i\eta_{\mu\nu}}{p_\gamma^2}$. We can now write the amplitude as

$$-i\mathcal{M} = \bar{u}(p_3)(ie\gamma^\mu)u(p_1)\frac{-i\eta_{\mu\nu}}{p_\gamma^2}\bar{u}(p_4)(ie\gamma^\nu)u(p_2) \tag{1.24}$$

Utilizing the conservation of momentum, we can write $p_\gamma = p_1 - p_3$ and rewrite

$$\mathcal{M} = -e^2\bar{u}(p_3)\gamma^\mu u(p_1)\frac{1}{(p_1-p_3)^2}\bar{u}(p_4)\gamma_\mu u(p_2). \tag{1.25}$$

After multiplying by the complex conjugate, summing over all the possible spins, and neglecting masses of electron and muon (we assume relativistic scattering), we can write the matrix element as

$$|\mathcal{M}|^2 = 2e^4\frac{(p_1+p_2)^2 + (p_1-p_4)^2}{(p_1-p_3)^2}. \tag{1.26}$$

We can now evaluate the cross section (1.21) in the centre of mass frame as

$$\frac{d\sigma}{d\Omega} = \frac{\alpha^2}{4(p_1+p_2)^2}(1+\cos^2\theta), \tag{1.27}$$

where θ is the scattering angle. Integrating over the solid angle, we get the total cross section

$$\sigma = \frac{4\pi\alpha^2}{3(p_1+p_2)^2}. \tag{1.28}$$

This calculation was an example of the lowest-order Feynman diagram. In general, many more diagrams contribute to the amplitude, for example in figure 1.2

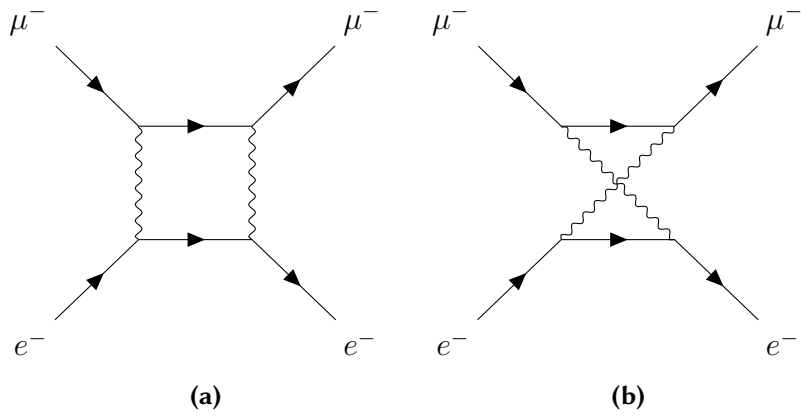


Figure 1.3 Fourth order Feynman diagrams for electron-muon scattering.

is a fourth-order diagram. All of these contributions are added together to get the full amplitude $\mathcal{M}$.

The *Feynman rules* for the general QED Feynman diagram can be found in [31]. Other interactions have slightly different Feynman rules, but the general idea is the same.

1.3 Quantum Chromodynamics

So far, we have introduced QED interactions as examples. In this section, we will introduce QCD , the theory of Strong interactions. The underlying Lie group of QCD is $SU(3)$. We can derive the following from the general theory introduced in section 1.2.2. $N = 3$, which means the underlying field space is three-dimensional. Physically, we have **3 quark fields** for each quark in the SM . They correspond to 3 color states $\psi_i(x)$, $i = 1, 2, 3$. $SU(3)$ has 8 generators T^a, which means we have **8 gluon fields** $A^a_\mu(x)$, $a = 1, \ldots, 8$. The full Lagrangian of QCD is [32] (we also suppress the summation over quark fields i, j)

$$\begin{aligned}
\mathcal{L}_{\text{QCD}} &= -\frac{1}{4} G^a_{\mu\nu} G^a_{\mu\nu} + \bar{\psi}_i (i\partial\!\!\!/ - m)\psi_i + g\bar{\psi}_i A\!\!\!/^a T^a_{ij} \psi_j \\
&= -\frac{1}{4} F^a_{\mu\nu} F^a_{\mu\nu} + \bar{\psi}_i (i\partial\!\!\!/ - m)\psi_i + g\bar{\psi}_i A\!\!\!/^a T^a_{ij} \psi_j + \\
&+ g f^{abc} \left[\left(\partial_\mu A^a_\nu - \partial_\nu A^a_\mu \right) A^\mu_b A^\nu_c + \left(\partial_\mu A^{a\nu} - \partial_\nu A^{a\mu} \right) A_{b\mu} A_{c\nu} \right] + \\
&+ g^2 f^{abc} f^{ade} A^\mu_b A^\nu_c A_{d\mu} A_{e\nu},
\end{aligned} \tag{1.29}$$

where we have expanded $G^a_{\mu\nu}$ as in (1.15). We can now see that the two last terms correspond to three and four-gluon coupling, as they contain the product of three or four, vector fields A, respectively.

1.3.1 Feynamn Diagrams of QCD

The complete list of Feynman rules for QCD can be found in [32]. However, we will introduce some of the essential Feynman diagrams here.

Quark-quark scattering

Lowest order diagrams for quark-quark scattering are shown in figure 1.4. Quarks have the same type of 'fermionic' line as leptons discussed in section 1.2. Gluons are represented with 'curly' lines to distinguish the Strong interaction from the Electromagnetic. The possibility of the initial quarks having different color charges adds magnitude to the overall amplitude because we need to sum over all possibilities. To illustrate this, we take a look at a fraction of cross section of $e^+e^- \to$ hadrons and $e^+e^- \to \mu^+\mu^-$

$$R = \frac{\sigma(e^+e^- \to \text{hadrons})}{\sigma(e^+e^- \to \mu^+\mu^-)}, \tag{1.30}$$

if we do not sum over the color charges, the fraction is $2/3$, but if we do sum over the color charges, the fraction is 2

$$R_{\text{no color}} = \frac{2}{3}, \quad R_{\text{color}} = 2. \tag{1.31}$$

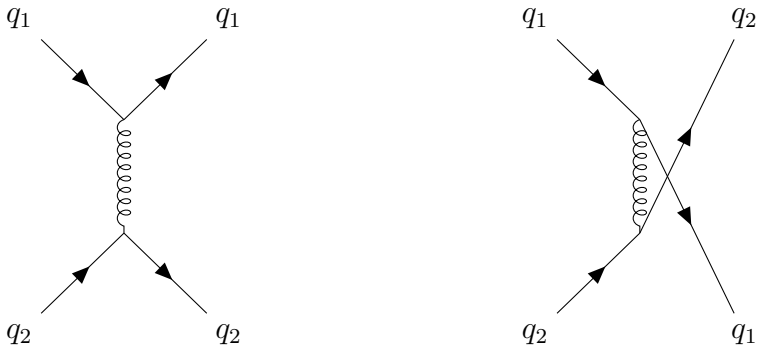


Figure 1.4 Lowest-order Feynman diagrams of quark-quark scattering.

Quark-gluon scattering

The lowest order diagrams for quark-gluon scattering are shown in figure 1.5. Due to the gluon self-interaction, there are more diagrams than in QED .

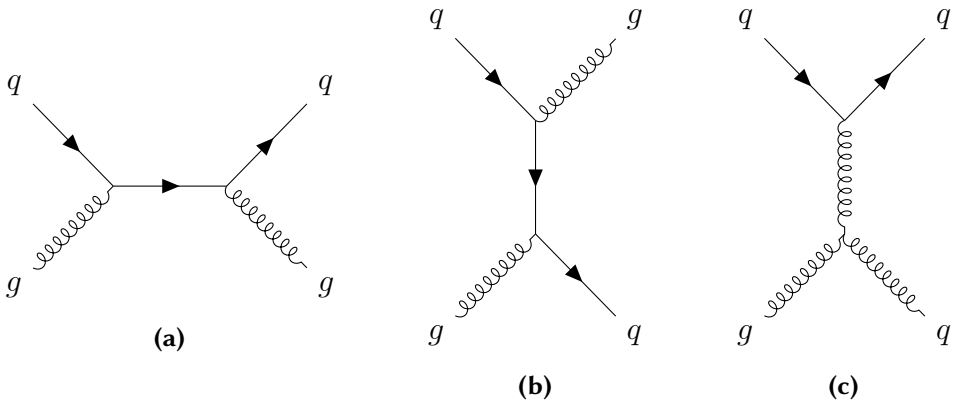


Figure 1.5 Lowest-order Feynman diagrams of quark-gluon scattering.

Gluon-gluon scattering

The lowest order diagrams for gluon-gluon scattering are shown in figure 1.6. In QCD , gluons can interact directly with each other. These interactions are non-trivial. A theory called *gluodynamics* is developed to describe just gluon-gluon interactions. In figure 1.6a and figure 1.6b, we can see the 3-gluon vertex, and in figure 1.6c, we can see the 4-gluon vertex.

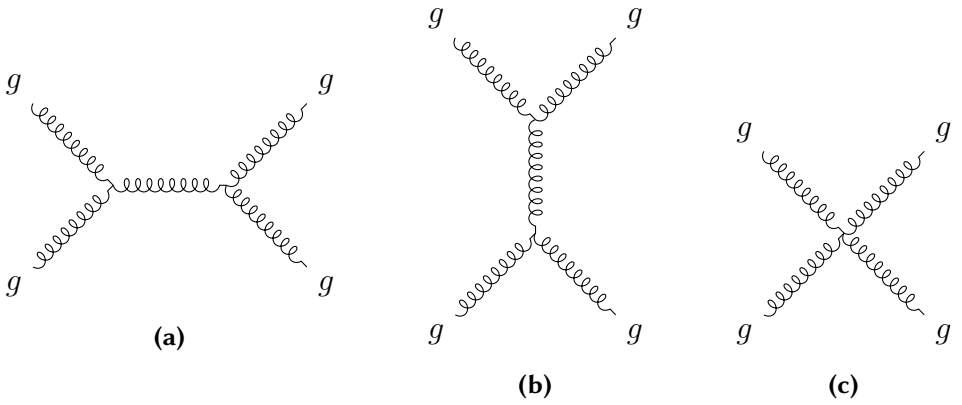


Figure 1.6 Lowest-order Feynman diagrams of gluon-gluon scattering.

Gluon Emission

Let's consider a e^+e^- annihilation process producing two quarks, and one of the quarks emits a real gluon. We will only look at the right part of the diagram after

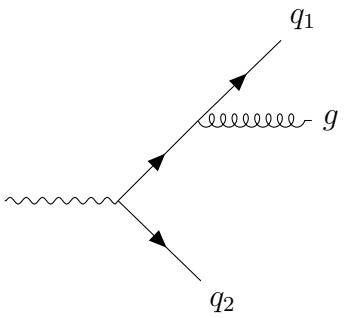


Figure 1.7 Lowest-order Feynman diagram of quark emitting a gluon.

the annihilation, as seen in figure 1.7. The scattering amplitude is given by [32]

$$\mathcal{M}^{\mu}_{q\bar{q}g} = \bar{u}\left(p_1\right)\left(-igT^a \not{\varepsilon}\right) \frac{i\left(\not{p}_1 + \not{k}\right)}{\left(p_1 + k\right)^2}\left(-ie\gamma^{\mu}\right) v\left(p_2\right), \tag{1.32}$$

where p_1, p_2 are the momenta of the quarks q_1, q_2, k is the momentum of the gluon, and ε is the solution of a free gluon. The gluon vertex contributes the factor $-igT^a \not{\varepsilon}$, corresponding to the term $g\bar{\psi}_i \not{A}^a T^a_{ij} \psi_j$ in the Lagrangian. The construction of $\mathcal{M}$ is similar to QED .

1.4 Infrared and Collinear Divergences

The perturbation expansion (1.23) is done in the constant of interaction g. However, the assumption of g being a constant is not valid. It changes with the energy of the particles, hence the distance between them. In QED , the coupling grows when we force particles to be closer to each other by smashing them together with high enough energy. In QCD , the coupling goes to zero at very high energies and it grows with the distance between the quarks. The difference arises because, QED is Abelian (the generators commute), while QCD is non-Abelian. The structure given by the Lie group $SU(3)$ dramatically changes the behavior of the coupling constant. Another way to formulate this is that the probability of a quark emitting a gluon, as it losses energy, goes to infinity.

We can see this directly from the scattering amplitude (1.32), wherein the dominator we have

$$(p_1 + k)^2 = 2E\omega(1 - \cos\theta), \tag{1.33}$$

where we consider $p_1 = (E, 0, 0, E)$, a *hard quark* (energetic quark, whose mass can be neglected), and $k = \omega(1, 0, \sin\theta, \cos\theta)$, θ being the angle between the quark and the gluon. If this term goes to zero, the scattering amplitude goes to infinity. There are two ways this can happen:

- **Infrared singularity**: gluon becomes low-energetic $\omega \to 0$
- **Collinear singularity**: gluon is emitted in the direction of the quark $\theta \to 0$

These infinities are **not physical**, but a consequence of not considering all the possible Feynman diagrams. The KLN theorem [33, 34] states: *When all the possible interactions are considered, the scattering amplitude is finite*, i.e., is **infrared and collinear (IRC) safe**.

The exact process happens for gluons. They can emit another gluon, and the scattering amplitude goes to infinity. Moreover, since gluons have two color charges, the coupling is approximately 2.25 times stronger than for quarks [35]. This is called *gluon radiation* and is why the coupling constant grows with the distance between the quarks.

The physical variable that describes the multiplication of quarks and gluon is *multiplicity* N_g for gluons and N_q for quarks. From the discussion above, we conclude that the *multiplicity of gluons is higher than the multiplicity of quarks.*

1.5 Hadronization

Hadronization is a process where free quarks and gluons become bound. The bound states of quarks are called *hadrons.* In this context, it is good to introduce the word *parton*, a generic term for a quark or a gluon. As we have seen in section 1.4, the less energetic the parton is, the more likely it will emit gluons. This means that when partons reach some energy scale, they become bound, effectively making them almost immediately bound after hard scattering processes. This inevitable binding is called *confinement.*

A phenomenological description of this process is given by the *Lund model* [36]. The soft gluonic interaction is modeled as a *string* connecting the hard partons with a **linear binding potential**. We are going to illustrate this on a hard scattered quark-antiquark pair:

- Quark and antiquark are emitted from a **hard scattering** process, each going in a different direction.
- A **string is created** between the quark and the antiquark.
- As they are moving away from each other, the string is **stretched**.
- Once the tension has enough energy, the string splits and creates a **new quark-antiquark pair**.
- This process is repeated until the initial **energy** of the quark-antiquark pair **is exhausted**.

- As a result, the created quarks and antiquarks rearrange to **color-neutral mesons or baryons**.

Mesons are hadrons of one quark and one antiquark, whereas *baryons* are three quarks. The lightest mesons are the *pions*, which are made up of a quark and an antiquark from the first generation (see section 1.1) of quarks. The lightest baryons are the *protons* and *neutrons*, which are made up of three quarks from the first generation (see section 1.1).

Another phenomenological model is called *cluster fragmentation* [37]. Here the gluons are split into quarks and antiquarks, then rearranged to *clusters* with zero net color charge. Both approaches are illustrated in figure 1.8.

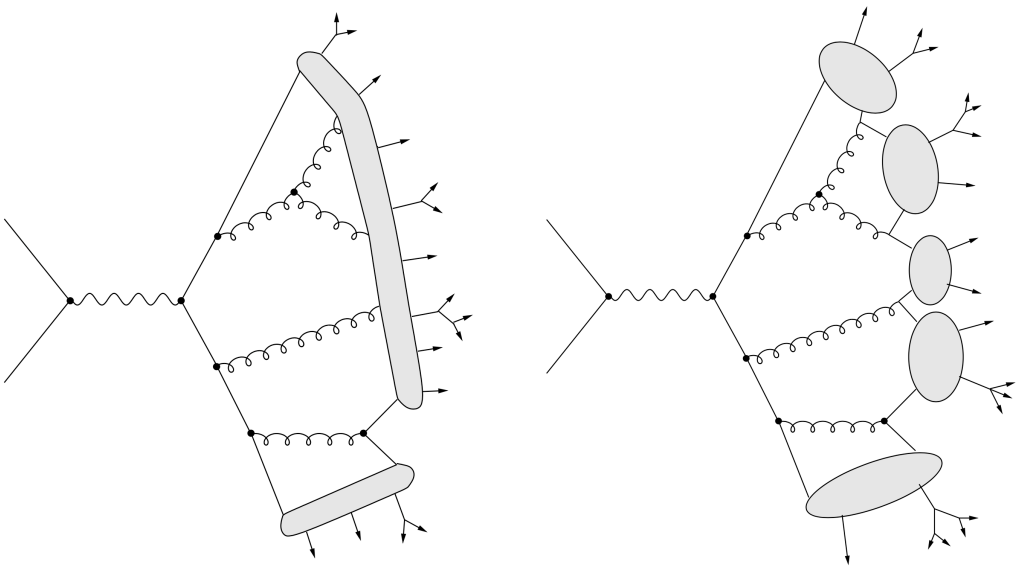


Figure 1.8 Illustration of the hadronization models. The left is the Lund model, and the right is the cluster fragmentation. [7]

The scattering of partons bound in a hadron cannot be described as hard interaction as it can be for an electron bound in an atom. As we described earlier the parton has a much higher probability to emit gluons at low energies. In a hadron, they create the so-called *sea of partons* on top of the three or two valence quarks. This parton mess is described by the *parton distribution function* (PDF) $f_i(x)$ [38], which is a probability distribution of the parton i carrying momentum fraction x of the hadron. An example is shown in figure 1.9, where the PDF $f(x, Q^2)$ for proton constituents is also a function of the squared energy scale Q^2. We can see that the dominant partons contributing to the proton momentum

[7] `https://arxiv.org/abs/1011.5131`

are u-quarks and d-quarks, the valence quarks, but the contribution from other partons is not negligible.

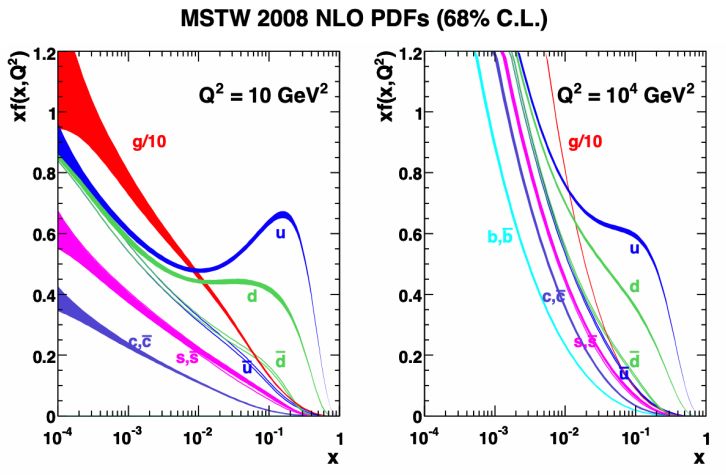


Figure 1.9 Example of the parton distribution function of a proton. The PDF $f(x, Q^2)$ is a function of the momentum fraction x and the squared energy scale Q^2. Different colors represent a different parton type. On the y-axis is the PDF $f(x, Q^2)$ multiplied by the fraction x. [8]

1.6 Jets

As seen in section 1.4, partons emit soft and collinear gluons. The initial momentum of the primary parton is split into more partons, creating a shower called *jet*. The main processes in proton-proton collisions that produce a jet are shown in figures 1.4 to 1.6.

Experimentally it is not straightforward to define a jet from the measured decays. For example, a quark initially radiates one collinear gluon, having approximately the same momentum as the quark. How do you decide whether the gluon is part of the initial jet or creates a new one? Or, if two jets exchange a soft gluon that emits another one, how do you decide where the new gluon belongs? This is important because if the jets are not defined correctly, the infinities in the scattering amplitude will not be canceled out, as is stated in the KLN theorem [33, 34]. This makes comparing the data with theory very difficult.

There are complex algorithms that use unique physical variables to enable the reconstruction of the jets [5]. In the first approximation, we can say that the jet

[8] `https://arxiv.org/pdf/0901.0002.pdf`

is a *cone*, where the apex corresponds to the initial parton and all other partons created are inside the cone. We will discuss this in more detail in section 3.5. The illustration of a jet definition is shown in figure 1.10.

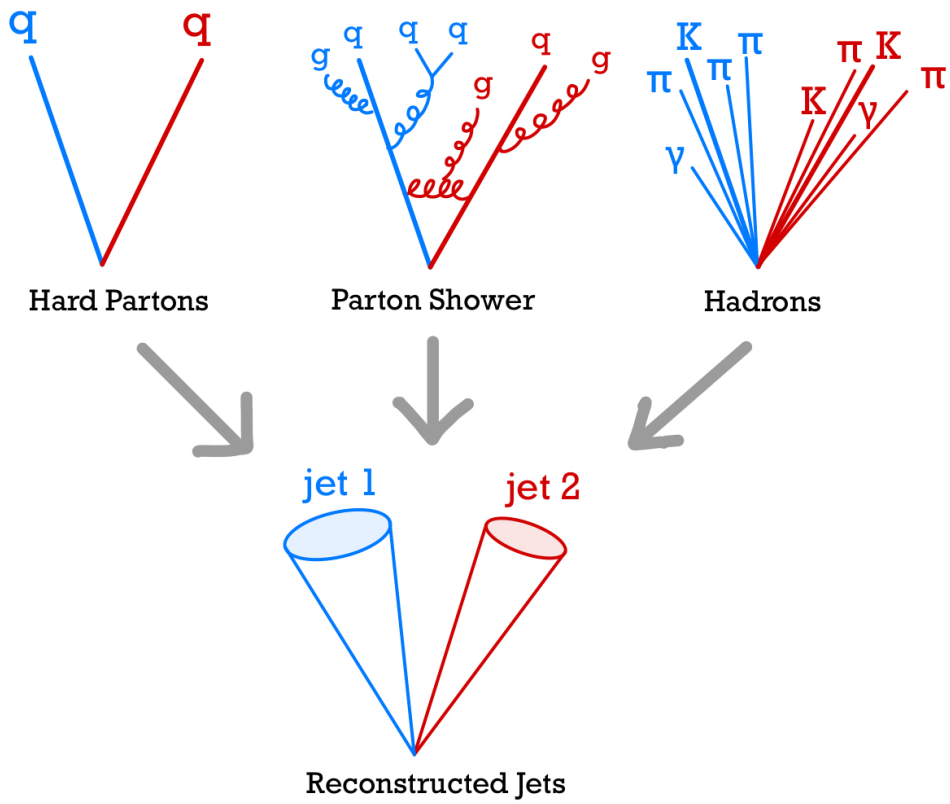


Figure 1.10 Jet definitions on various levels. Good reconstructed jet should be the same at all levels.

Another aspect of jets is the definition of their energy. The most common approach to defining the jet 4-momentum is by summing the 4-momenta of all the partons inside the jet [5]. The problem is that it introduces a mass to the jet, but the initial parton migth be massless.

1.6.1 Infrared and Collinear Safety

One way of making sure that the jet algorithm is reflecting the physics of partons is to make sure that it is *infrared and collinear safe*:

- **Collinear safety** means that if the initial parton (or any other in the shower) radiates a *collinear* gluon, it is always part of the same jet.

- **Infrared safety** means that the jet is unchanged if the initial parton (or any other in the shower) radiates a *soft* gluon.

We can now define *Infrared and Collinear safe* (IRC) observable $\mathcal{O}$ as:

Definition 1 (Infrared and Collinear Safe Observable). *Let $\mathcal{O} = \mathcal{O}(p_1^\mu, p_2^\mu, \ldots, p_N^\mu)$ be observable on the momentum space of N partons with 4-momentum p_i^μ matched to a jet by a jet algorithm. Then $\mathcal{O}$ is :*

- ***Infrared safe** wrt. the jet algorithm if*

$$\mathcal{O}(p_1^\mu, p_2^\mu, \ldots, p_N^\mu) = \mathcal{O}(p_1^\mu, p_2^\mu, \ldots, p_N^\mu, p_{N+1}^\mu)$$

  where $p_{N+1}^\mu \approx 0$ is 4-momentum of a soft parton.

- ***Collinear safe** wrt. the jet algorithm if*

$$\mathcal{O}(p_1^\mu, p_2^\mu, \ldots, p_N^\mu) = \mathcal{O}(p_1^\mu, p_2^\mu, \ldots, p_{N-1}^\mu, p_{N_1}^\mu, p_{N_2}^\mu)$$

  where $p_{N_1}^\mu \parallel p_{N_2}^\mu$ are 4-momenta of collinear partons and $p_{N_1}^\mu + p_{N_2}^\mu = p_N^\mu$.

In figure 1.11, we can see that an IRC unsafe jet algorithm initially finds two jets, but when one jet radiates a soft gluon, they are put together or when a jet radiates a collinear gluon, the jet is split.

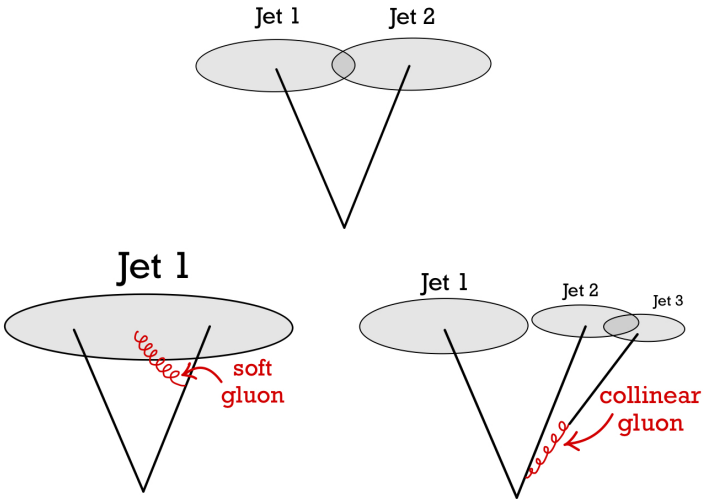


Figure 1.11 IRC unsafe jet algorithm. Initially, there are two jets (*top*), but they are put together if one jet radiates a soft gluon (*bottom left*), or one jet is split if it radiates a collinear gluon (*bottom right*).

Chapter 2

Detector ATLAS

2.1 Overview

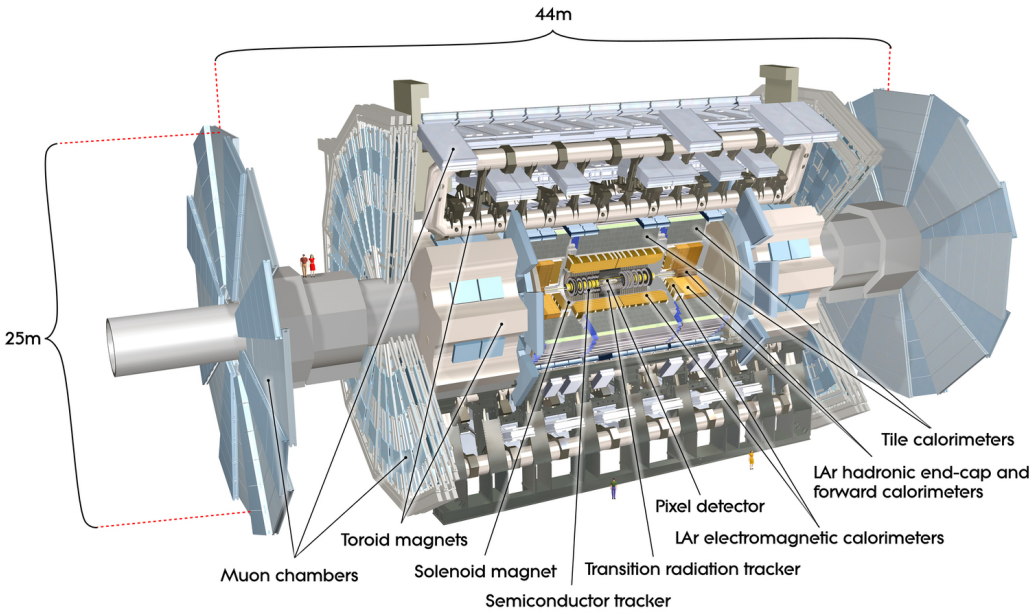


Figure 2.1 The ATLAS detector

LHC is a circular accelerator with a circumference of around 27.6km located on the Swiss-French border near Geneva, at CERN [2]. It accelerates particles, mainly protons and occasionally heavy ions (dominantly lead), to almost the speed of light, reaching a center-of-mass energy $\sqrt{s} = 14$ TeV. On the accelerating ring are four *interaction points*, where the particles collide and are detected by the

ATLAS, CMS , ALICE, and LHCb detectors. ATLAS and CMS are multi-purpose detectors, while ALICE is designed to study heavy-ion collisions, and LHCb is designed to study the physics of b-quarks.

The ATLAS detector [4] is a general-purpose particle detector located in a cavern 100m underground. The detector is a multi-layered cylindrical structure with a diameter of 46m and a height of 25m. The main detecting components are the *silicon pixel* and *strip trackers*, the *electromagnetic calorimeter*, the *hadronic calorimeter*, and the *muon spectrometer*. In figure 2.1, we show a schematic of the ATLAS detector, in figure 2.2 the cross section of the detector and its interactions with the particles in individual components, and figures 2.5 to 2.8 the individual components of the detector.

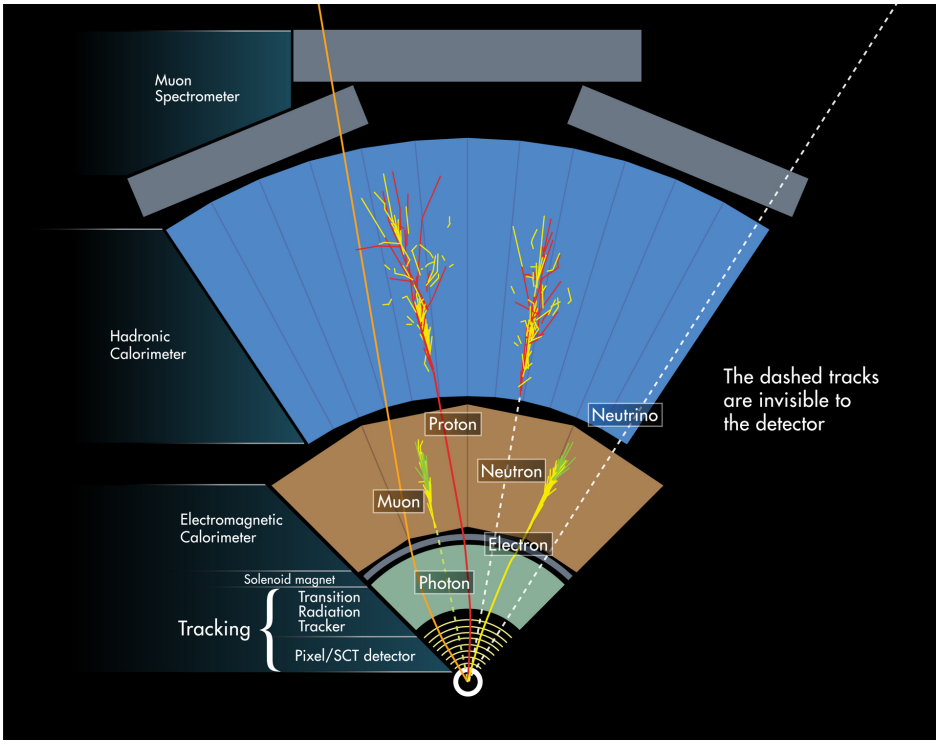


Figure 2.2 The ATLAS detector cross section and its interactions with the particles.

In the following section 2.2, we will discuss the LHC and in sections 2.3 to 2.7 the individual components of the detector. Before doing so, we will introduce the ATLAS coordinate system and space-orienting notation in section 2.1.1.

2.1.1 ATLAS Coordinate System

The ATLAS coordinate system is defined as follows:

1. The *origin of the coordinate* system is the nominal interaction point.
2. the x-y *plane* is perpendicular to the beam direction.
3. The positive x *axis* is pointing towards the centre of LHC .
4. The positive y *axis* is pointing upwards.
5. The z *axis* is along the beam direction, such that it forms a right-handed coordinate system with the x and y axes.
6. The *azimuthal angle* ϕ is measured around the z axis, being zero on the positive x axis.
7. The *polar angle* θ is measured from the z axis, where it is zero.

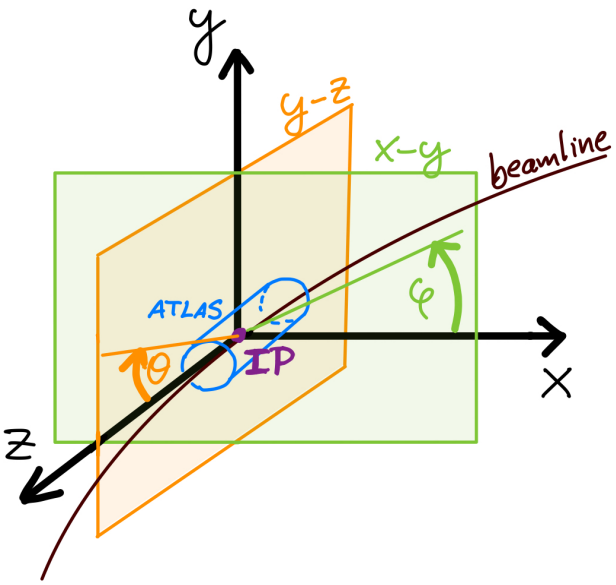


Figure 2.3 The ATLAS coordinate system.

We also define the **pseudorapidity** η as

$$\eta = -\ln(\tan(\theta/2)), \tag{2.1}$$

and the **transverse momentum** p_{T} as momentum in the x-y plane. For massive objects, we also define the **rapidity** as

$$y = \frac{1}{2}\ln\left(\frac{E + p_z}{E - p_z}\right), \tag{2.2}$$

where E is the energy and p_z is the momentum along the z axis of the particle. The forward region is defined as $|\eta| > 2.5$, and the central region as $|\eta| < 2.5$.

Simple sketch of the ATLAS coordinate system is shown in figure 2.3.

2.2 Large Hadron Collider

LHC is the biggest particle accelerator in the world, operated by CERN. It is built underground, up to 175m below the surface [1]. To keep the particles in a circular motion, it uses *superconducting NbTi magnets*, reaching up to 8.33 T, submerged in a liquid Helium of temperature 1.9 K. Particles are pre-accelerated in a sequence of linear and circular accelerators. The system of all accelerators is shown in the figure 2.4.

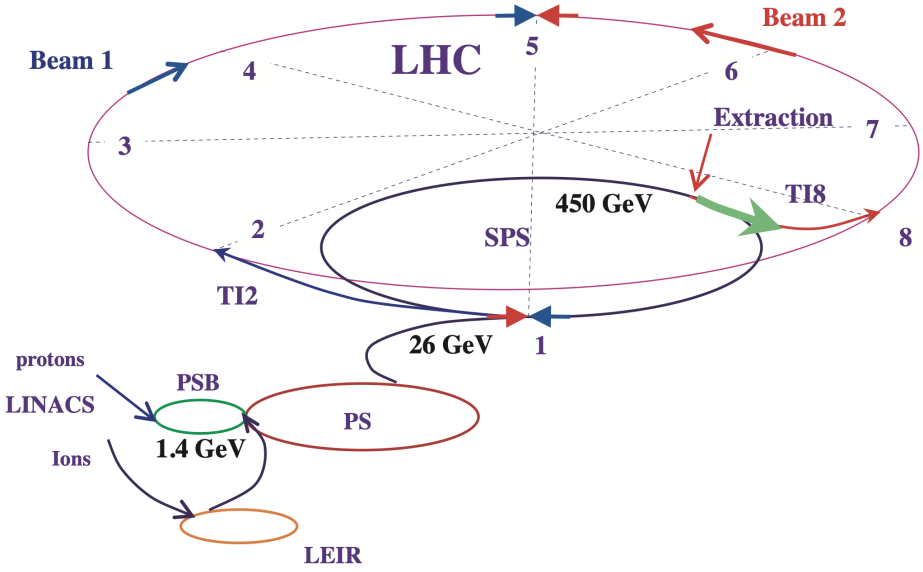


Figure 2.4 The Large Hadron Collider, with its pre-accelerators.

At a given time, two beams of protons are circulating in opposite directions in the accelerator. Beams are separated into *bunches*, each containing approximately $1.3 \cdot 10^{11}$ protons. At maximum, 2800 bunches are present in each beam, with a typical size of 200-300 μm and a spacing of 25 ns (about 7.5m). The beams collide at four specific points where the detectors are. Before the collision, the beams are focused to a size of 16 μm.

[1] Due to the high prices of land in Geneva.

The most important physical variable of the accelerator is the *luminosity* L, defined as

$$L = \frac{N_1 N_2 n_b f_{\text{rev}}}{A}, \tag{2.3}$$

where N_1, N_2 are the number of particles in each beam, n_b is the number of bunches, f_{rev} is the revolution frequency, and A is the effective beam overlap cross section of the collision. If we assume the Gaussian distribution of the beams in the transverse plane, the overlap cross section is given by

$$A = 4\pi\sigma_x\sigma_y, \tag{2.4}$$

where σ_x and σ_y are the standard deviations of the Gaussian distribution. LHC has a luminosity of $L = 10^{34}$ cm^{-2}s^{-1} translating to billion collisions per second.

Another related variable is the *integrated luminosity* $\mathcal{L}$, expressing the amount of data collected in a given time period T

$$\mathcal{L} = \int_T L dt, \tag{2.5}$$

expressed in the units of inverse femtobarns (fb^{-1}). For example, the ATLAS detector collected 140 fb^{-1} of data.

2.3 Inner Detector

The *inner detector* consists of three parts: *Pixel Detector*, *Semiconductor Tracker*, and *Transition Radiation Tracker*, figure 2.5. They are the first detection points that are hit after the collision. The primary purpose of the inner detector is to reconstruct the tracks of the charged particles. As we can see in figure 2.2, only charged particles are visible by the inner tracking system.

Most interesting collisions are produced particles within $|\eta| < 2.5$, so the tracker is only in this region. Even more, the inner detector is much denser and has a higher granularity in the central region, as seen in figure 2.5.

2.3.1 Pixel Detector

The pixel detector, the first detection point, has the highest resolution and must withstand the highest radiation dose. It consists of four layers of silicon pixels of size 50 x (250-400) μm^2. With an accuracy of about 10 μm, it can locate a charged particle's origin and measures its momentum.

The primary physical detection process is ionization. As charged particles pass through the semiconducting silicon, they create electron-hole pairs, which are collected by the readout electronics.

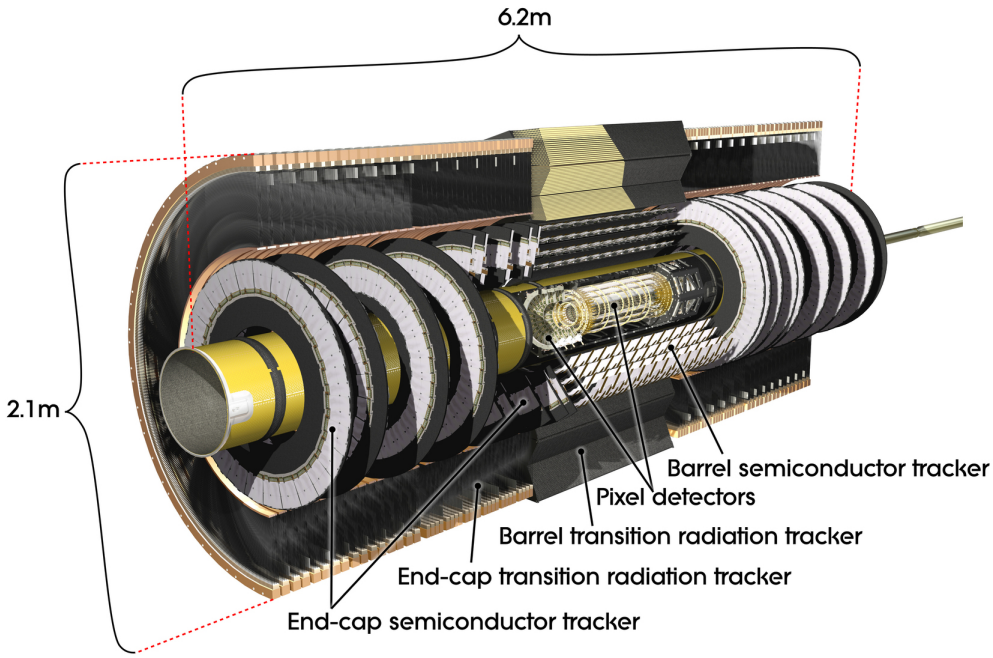


Figure 2.5 Tracker system of the ATLAS detector.

2.3.2 Semiconductor Tracker

Semiconductor Tracker (SCT) is similar to the pixel detector but with a different type of position measuring. Instead of pixels, it uses *strips* to determine the particle's position. It covers way more volume than the pixel detector, with a precision of up to 25 μm. Determination of coordinates is done by precise alignment of the strips.

The physical process behind the detection is the same as in the pixel detector.

2.3.3 Transition Radiation Tracker

Transition Radiation Tracker (TRT) is the last part of the inner detector, being the least precise. Instead of silicon, it uses a gas mixture of Xe, CO_2, and O_2 to provide the ionizable material. The gas is put into *straws* with a diameter of 4 mm and a length of up to 144 cm. Along the straw, in the middle, is a gold-plated tungsten wire acting as an anode, while the cathode is the tube wall.

When a charged particle passes through the gas, it produces ion pairs. The anode is at ground potential attracting negative ions, while the cathode wall is at -1530V attracting positive ions. The straws are interleaved with polypropylene fibers or foils to provide a material with different index of refraction. Relativistic

particles, with speeds above the speed of light in polypropylene (mainly electrons), generate transition radiation when moving from and into the polypropylene. Additional information about the particle is inferred from this transition radiation, which is used to reconstruct the track and identify the particle.

2.4 Calorimeters

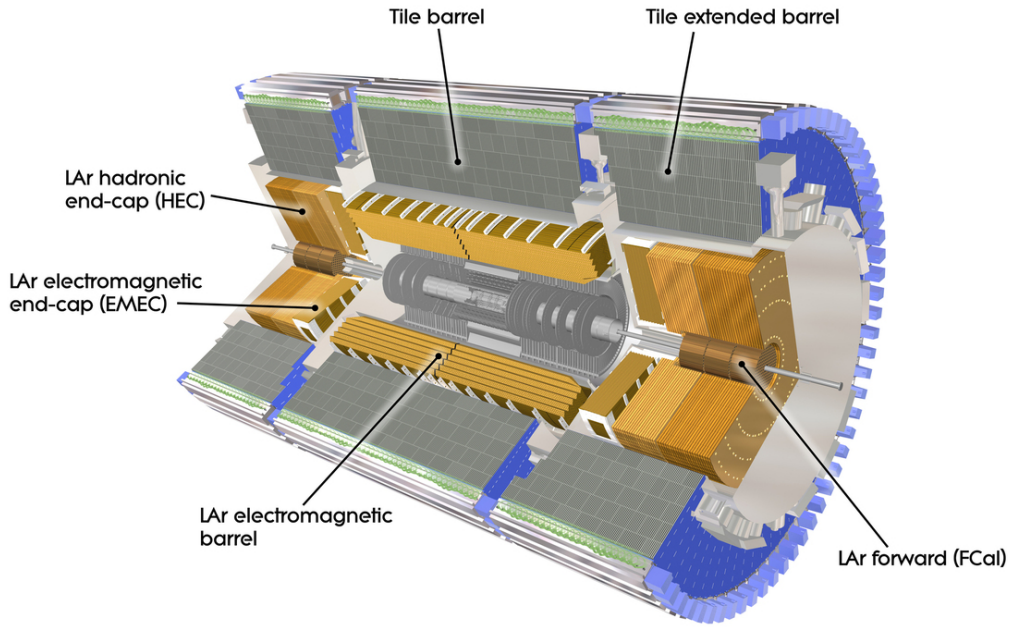


Figure 2.6 Calorimeter system of the ATLAS detector.

The primary purpose of calorimeters is to stop particles. More precisely, they are designed such that particles deposit most of the energy in them. That is why they are much larger than the tracking system, as seen in figure 2.2. They are built up by alternating absorbing layers and energy-measuring layers.

In figure 2.2, we can see two types of calorimeters:

- **Electromagnetic calorimeter** is designed to stop electrons, positrons, and photons.
- **Hadronic calorimeter** is designed to stop hadrons, i.e., composites of quarks and gluons.

The hadronic calorimeter is the most important for our study because it provides data for the jet reconstruction.

2.4.1 Electromagnetic Calorimeter

Another name for the electromagnetic calorimeter is *liquid argon* (LAr) calorimeter because it uses liquid argon as a detecting material. The LAr is kept at -186°C to maintain the liquid state. As an absorber, the electromagnetic calorimeter uses lead. Particles ionize the LAr when passing the layers, creating a measurable current, which is then detected by accordion-shaped electrodes.

2.4.2 Hadronic Calorimeter

The hadronic calorimeter uses mainly *Tile Calorimeter* (TileCal) , an envelope of the electromagnetic calorimeter, and LAr at the endcaps and forwards part. The LAr part is similar to the electromagnetic calorimeter described above, but instead of lead, it uses copper and tungsten as an absorber.

The TileCal uses steel as an absorber and scintillating tiles as a detector. As particles pass through the plastic scintillators, they excite the molecules inside, which emit light as they return to their ground state. This visible light is then collected by optical fibers and converted into an electrical signal with photomultiplier tubes. The intensity of the light is proportional to the energy deposited in the scintillator.

2.5 Muon Spectrometer

The Muon Spectrometer is the furthest measuring system of the ATLAS detector, as seen in figure 2.7. Muons have bigger mass than electrons and do not interact with the strong interaction as hadrons, so they are not stopped in the calorimeters (see figure 2.2)

Since stopping the muons is not feasible, we measure their momentum and flight direction, utilizing the muon track's bending in a magnetic field, which we will explain in the next section 2.6. Muon spectrometers surround the entire ATLAS detector to ensure that all muons are measured. The tracker provides one measurement point, while the spectrometer offers the other to reconstruct the muon track with the precision of 10% at 1 TeV.

2.6 Magnet systems

In the whole ATLAS detector, there are two main systems of magnets, toroidal (blue field lines in figure 2.8) and solenoidal (green field lines in figure 2.8). All magnets are superconducting, operated at -268°C, to reach necessary field strengths.

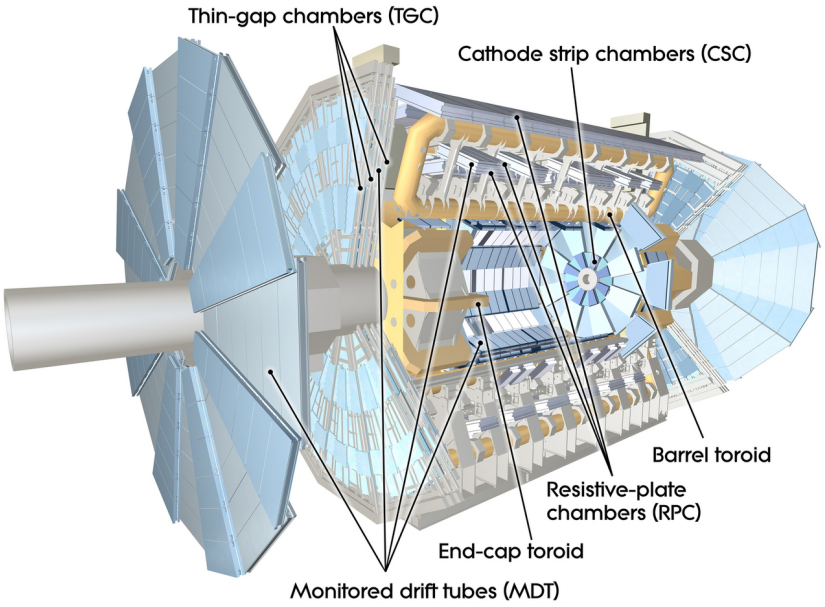


Figure 2.7 Muon spectrometer system of the ATLAS detector.

The main purpose of magnets is to bend the trajectories of charged particles to measure their transverse momentum p_{T} according to [2]

$$p_{\mathrm{T}}[\mathrm{GeV}] = 0.3 \cdot q \cdot B[\mathrm{T}] \cdot R[\mathrm{m}], \tag{2.6}$$

where q is the particle's charge (expressed in units of the elementary charge), B is the magnetic field strength, and R is the radius of the trajectory.

The inner NbTi magnets are solenoidal and provide coverage for the tracking system. They produce a 2T magnetic field in the axial direction. To give a perspective, if we assume an electron with a momentum of 10 GeV, the bending radius is 17 m. This number is enormous compared to the radius of the solenoid magnet, which is just 1.2m. From this example, we can see how precise the tracking system is.

On the other hand, particles with momentum less than circa 360 MeV wind inside the solenoid magnet and do not reach the colorimeters. One such example is low energetic pions, particularly important in our study because they are trapped inside the solenoid magnet, not reaching the calorimeters, and are not used in the jet reconstruction.

The toroidal magnets are composed of two parts: barrel and endcap. The barrel magnets consist of eight 25.3m long, rectangular-shaped coils, the biggest toroidal magnets in the world. They are supported by the endcap magnets, which

[2] We assume that the magnetic field is constant and the particle's mass is zero.

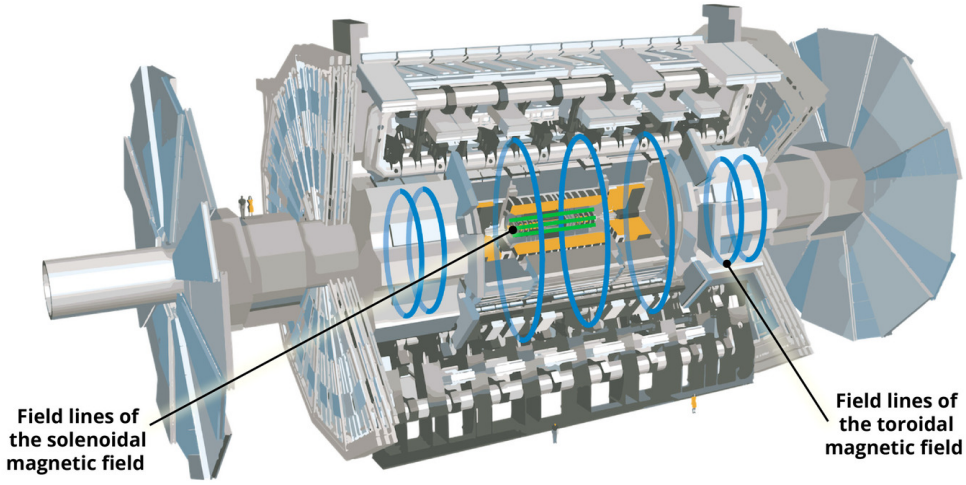


Figure 2.8 Magnet system of the ATLAS detector.

cover particles going in the forward direction, as seen in figure 2.8. Together they provide a field of approximately 0.5-1T for bending the trajectories of muons, measured by the muon spectrometer.

2.7 Trigger

The high luminosity that LHC provides results in an interaction rate of 40 MHz, equivalent to 60 TB/s of data throughput [39]. This is a massive amount of data, impossible to store and process. Most of the data contains no interesting physics, so it is filtered by the *trigger system*.

The trigger system consists of two levels, *L1* and *High Level Trigger* (HLT) [39]. L1 uses data from calorimeters and muon spectrometers defining a *Region-of-Interest* (RoI) , where it decides whether to keep the event or not, reducing the interaction rate to 100 kHz. Fast custom electronic hardware with a latency of 2.5μs is used.

The HLT trigger uses a CPU farm of 40k CPU cores to reduce the event rate to 1 kHz. After it, the data is prepared for long-time storage and analysis.

Chapter 3

Data

3.1 Monte Carlo Simulations

As training data for various neural network models, we do not use measured data by the ATLAS detector, but rather *simulated events*. Simulations are based on physical models, detector models, and stochastic processes involved in both the physics and the detector. Hence the name, *Monte Carlo* simulations.

Simulation can be split into two parts: the physics simulation and the detector simulation. The physical part uses the `Pythia 8.2` package [40], a state-of-the-art event generator. The detector response is made using the `Geant4` package [41] tailored to the ATLAS detector [42].

After the simulations, the MC data comes in the same format as the actual data from the detector. For the simulation to make physical sense and still allow an effective production of rare events, they assign *weight* to each event based on the cross section of the simulated process. MC differs from data in the *truth information* it contains from the physical simulation, which is used to train the neural networks. However, object reconstruction is still done the same way as for the actual data. Jet reconstruction is done in two steps: first, the jet constituents are identified, and then the jet is reconstructed from the constituents. Constituent identification will be discussed in section 3.3 and jet reconstruction in section 3.5.

3.2 Event Production

`Pythia` is capable of simulating hard interactions at LO (leading order in perturbative QCD) accuracy and clustering partons into hadrons using the Lund string model (described in section 1.5). We utilize the ATLAS A14 tuned parameters [43] using the `NNPDF23LO` [44] parton density functions (see section 1.5). The decays of heavy flavor hadrons are simulated using the `EvtGen` package [45].

Cross sections of events depend highly on the p_T of scattered partons. To be able to describe well the high p_T (low probability, i.e., cross section) events, the MC `Pythia` production is split into `JZ` slices. Each slice has a different p_T range of leading (with the highest p_T) jet. In the original dataset are 13 slices, `JZ0-JZ12`, but we will only utilize `JZ1-JZ5`, which are the most interesting for physical analyses. A summary of the used `JZ` samples is shown in table 3.1. To make the physical sense of the `JZ` slices, each slice is assigned a cross section on top of the event weight, also summarized in table 3.1.

Table 3.1 Summary of the used `JZ` samples. Each slice corresponds to a different p_T range of the leading jet. The higher the p_T the lower the probability, i.e. cross section σ.

Slice	p_T range [GeV]	σ [nb]	Filter Efficiency	Size [GB]
`JZ1`	20-60	$7.8 \cdot 10^{10}$	$2.44 \cdot 10^{-2}$	78
`JZ2`	60-160	$2.4 \cdot 10^{9}$	$9.86 \cdot 10^{-3}$	29
`JZ3`	160-400	$2.6 \cdot 10^{7}$	$1.16 \cdot 10^{-2}$	70
`JZ4`	400-800	$2.5 \cdot 10^{5}$	$1.34 \cdot 10^{-2}$	39
`JZ5`	800-1300	$4.6 \cdot 10^{3}$	$1.45 \cdot 10^{-2}$	22

3.2.1 Truth Label

We use the `PartonTruthLabelID` as the target during the training. This label is the flavor of the *ghost parton* with the highest energy in the jet. Ghost partons are created from the partons in the jet by rescaling their energy with a small number (10^{-15}), adding them to the jet, and then rerunning the jet clustering algorithm. Rescaled partons matched with the original partons to a jet are called *ghost partons*. The truth label defined this way is *not collinear safe*.

3.3 Constituent Identification

Our study identifies the jet constituents using the PF algorithm [6].

The PF algorithm is based on the Topocluster algorithm [46], which uses only calorimeter (electromagnetic and hadron) information to identify energy clusters. The seed of the cluster is a signal of strength at least 4σ, where σ is the noise of the calorimeter. When such a seed is found, the algorithm searches for neighboring cells with signal strength above 2σ. The neighboring cells are added to the cluster, and the algorithm repeats the search for other neighboring cells. When no more neighboring cells with signal $> 2\sigma$ are found, all cells that form a boundary are added to the cluster. After the algorithm looks for local maxima in

the cluster and splits it into two or more, if necessary. If a cell is neighboring two or more clusters, the cell is assigned to two clusters with the highest energy. An example of clusters is shown in figure 3.1.

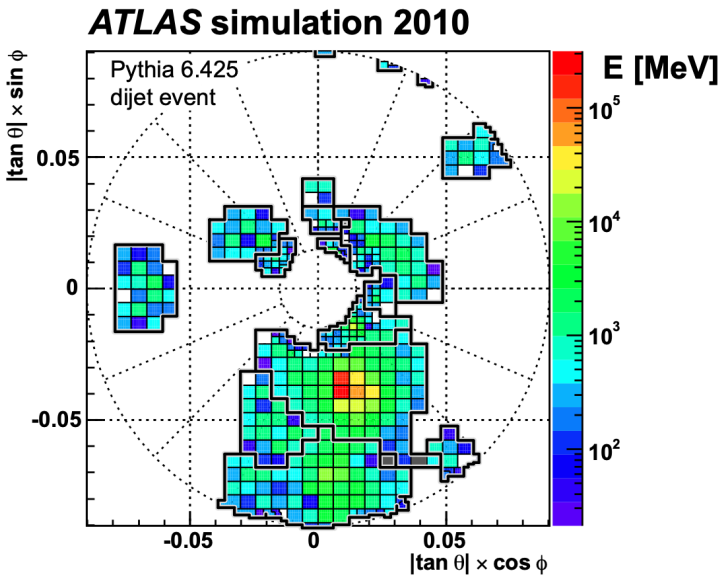


Figure 3.1 Example of clusters found by the Topocluster algorithm [46]. Black lines separate clusters.

Apart from topo-clusters, the PF algorithm also uses track information from the inner detector. The main improvement of the PF algorithm is the removal of double counting of energy in multiple clusters and putting them together as an energy shower. A flow chart of the PF algorithm is shown in figure 3.2.

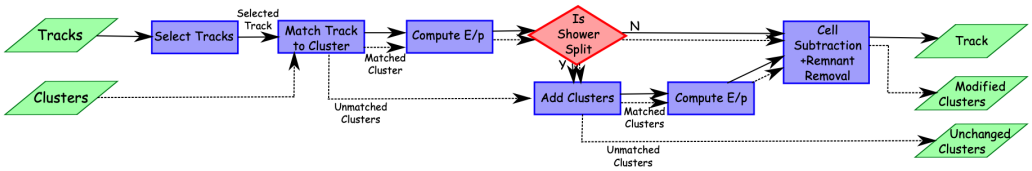


Figure 3.2 Flow chart of the Particle Flow algorithm [6].

The object construction procedure is as follows:

1. The PF algorithm **starts** with the topo-clusters and tracks.
2. Single **track is matched** to a single topo-cluster.
3. **Energy** of the track and the topo-cluster **is calculated**.

4. If the track's energy is larger than the energy of the topo-cluster, a **new topo-cluster is added** to the list of topo-clusters.

5. The **energy** of the list of topo-cluster **is updated**.

6. This is **repeated until** the **track energy matches** the energy of the list of **topo-clusters**.

7. When energy matches, the **energy** from the list **of topo-clusters is subtracted** from the whole calorimeter energy, and remnants are removed.

8. The output of the PF algorithm is a **list of topo-clusters and tracks**.

Energy showers used for jet reconstruction are an ensemble of matched topo-clusters and tracks assigned to hard-scattered objects from the primary vertex. Note that tracks are assigned only to charged particles with $|\eta| < 2.5$ (tracker position), the rest are only assigned topo-clusters. We will refer to elements of topo-cluster and track ensemble as *Particle Flow Objects* (PFOs) .

3.4 Pile-up

Pile-up refers to particles created in interactions, not from the primary, hard interaction. As particles come in bunches, scattering can happen between whole bunches, creating other particles. The important scatter tagged by the trigger is clouded by pile-up particles, creating another background on top of the electronics. The Topocluster algorithm searches for clusters on top of the pile-up, meaning that the noise σ also contains pile-up particles.

In this context, it is good to introduce variables modeling the pile-up effect on the jet [47]. Specifically, jet-vertex-fraction JVF, corrected JVT corrJVT, momentum fraction originated from primary vertex $R_{p_{\mathrm{T}}}$, jet-vertex-tagger JVT, and forward jet-vertex-tagger fJVT. The primary vertex has the largest sum of squares of transverse momenta of associated tracks.

The jet-vertex-fraction JVF is a fraction of the jet transverse momentum originating from the primary vertex (*1st vertex* is a shortcut for primary vertex, and *2nd vertices* is a shortcut for all other vertices)

$$\mathrm{JVF} = \frac{\sum_{\text{track}\in\text{1st vertex}} p_{\mathrm{T}}^{\text{track}}}{\sum_{\text{track}\in\text{1st vertex}} p_{\mathrm{T}}^{\text{track}} + \sum_{\text{track}\in\text{2nd verticies}} p_{\mathrm{T}}^{\text{track}}}, \tag{3.1}$$

where $p_{\mathrm{T}}^{\text{track}}$ is the transverse momentum of the track.

The corrected JVT corrJVT is a variable that corrects the JVF for the linear increase of pile-up p_{T} in jet wrt. the total pile-up in an event

$$\mathrm{corrJVF} = \frac{\sum_{\text{track}\in\text{1st vertex}} p_{\mathrm{T}}^{\text{track}}}{\sum_{\text{track}\in\text{1st vertex}} p_{\mathrm{T}}^{\text{track}} + \frac{\sum_{\text{track}\in\text{2nd verticies}} p_{\mathrm{T}}^{\text{track}}}{k\cdot N_{\text{pile-up}}^{\text{track}}}}, \tag{3.2}$$

where k is a constant (usually $k = 0.01$), and $N_{\text{pile-up}}^{\text{track}}$ is the total number of pile-up tracks in the event.

The variable $R_{p_{\mathrm{T}}}$ is a fraction of the transverse jet momentum originating from the primary vertex

$$R_{p_{\mathrm{T}}} = \frac{\sum_{\text{track}\in\text{1st vertex}} p_{\mathrm{T}}^{\text{track}}}{p_{\mathrm{T}}^{\text{jet}}}, \tag{3.3}$$

where $p_{\mathrm{T}}^{\text{jet}}$ is the transverse momentum of the whole jet (including topo-clusters).

The jet-vertex-tagger JVT is a discriminant calculated from corrJVF and $R_{p_{\mathrm{T}}}$ using *k-Nearest Neighbors* (kNN) algorithm in two-dimensional corrJVT-$R_{p_{\mathrm{T}}}$ plane. It assigns a number between 0 and 1, where 0 is a pile-up, and 1 is a primary vertex scatter.

The forward jet-vertex-tagger fJVT [48] is a variable assigned to forward jets that do not have any track assigned to them given as a normalized projection

$$\mathrm{fJVT} = \max_{\text{2nd verticies}} \frac{\mathbf{p}_{\mathrm{T,\ 2nd\ vertex}}^{\text{miss}} \cdot \mathbf{p}_{\mathrm{T}}^{\text{forward jet}}}{|\mathbf{p}_{\mathrm{T}}^{\text{forward jet}}|^2}, \tag{3.4}$$

where $\mathbf{p}_{\mathrm{T}}^{\text{forward jet}}$ is the two-dimensional transverse momentum vector of a forward jet (no tracks assigned), and $\mathbf{p}_{\mathrm{T,\ 2nd\ vertex}}^{\text{miss}}$ is given by

$$\mathbf{p}_{\mathrm{T,\ 2nd\ vertex}}^{\text{miss}} = -\frac{1}{2}\left(\sum_{\text{track}\in\text{2nd vertex}} k\mathbf{p}_{\mathrm{T}}^{\text{track}} + \sum_{\text{jet}\in\text{2nd vertex}} \mathbf{p}_{\mathrm{T}}^{\text{jet}}\right), \tag{3.5}$$

where k is a constant (usually $k = 2.5$), and $\mathbf{p}_{\mathrm{T}}^{\text{track}}$ and $\mathbf{p}_{\mathrm{T}}^{\text{jet}}$ are the two-dimensional transverse momentum vectors of tracks and jets assigned to a 2nd vertex, respectively.

It is necessary to emphasize that the momenta $p_{\mathrm{T}}^{\text{track}}$ are calculated from the tracks, not the PFOs because the Topocluster algorithm cannot assign a vertex. If a jet has no track assigned, JVF, corrJVF, and JVT are all set to -1. Jets with

JVT< 0.5 are considered as pile-up (passJVT=0), and similarly for fJVT< 0.5 (passfJVT=0)[1].

Another vital variable describing pile-up is the **average expected number of interactions per bunch crossing** denoted by μ.

3.5 Jet Reconstruction

There are two commonly used types of jet reconstruction algorithms: *fixed cone* and *sequential recombination* [49].

Fixed cone algorithms are based on the idea of a cone of fixed size, where all objects inside the cone make up the jet. Examples are SISCone [50], or CellJet [40]. The simple principle goes as follows:

1. Start with a given direction, usually the momentum direction of the hardest objects.
2. Construct a cone of fixed size around the direction.
3. Add all objects inside the cone to the jet and exclude them from further consideration.
4. Repeat the procedure for the remaining objects.

In case of cones overlapping, they are merged if the overlapping energy is more than a given threshold (usually 0.5 of total energy). Jet is split into two if the energy of the overlapping cone is less than a given threshold (usually 0.5 of total energy).

Sequential recombination algorithms are based on clustering jets by some metric, 'distance'. The general form of the common metric d_{ij} between two object, and reference (*beam*) distance $d_{i\mathrm{B}}$ are given by [5]

$$d_{ij} = \min\left(p_{\mathrm{T},i}^{2p}, p_{\mathrm{T},j}^{2p}\right)\frac{\Delta_{ij}^2}{R^2}, \quad d_{i\mathrm{B}} = p_{\mathrm{T},i}^{2p}, \tag{3.6}$$

where $p_{\mathrm{T},i}$ and $p_{\mathrm{T},j}$ are the transverse momenta of the objects i and j, $\Delta_{ij}^2 = (y_i - y_j)^2 + (\phi_i - \phi_j)^2$ is the squared radial distance between the constituents (coordinates are as defined in section 2.1.1), R is the radius parameter, and p is a parameter determining the type of algorithm:

- $p = -1$ is the anti-k_t [2]algorithm [5],

[1]Jets are also required to have `Timing`< 10 ns to pass the fJVT (passfJVT=1). See appendix A for the description of `Timing`.

- $p = 1$ is the k_t algorithm [51],
- $p = 0$ is the Cambridge/Aachen algorithm [52].

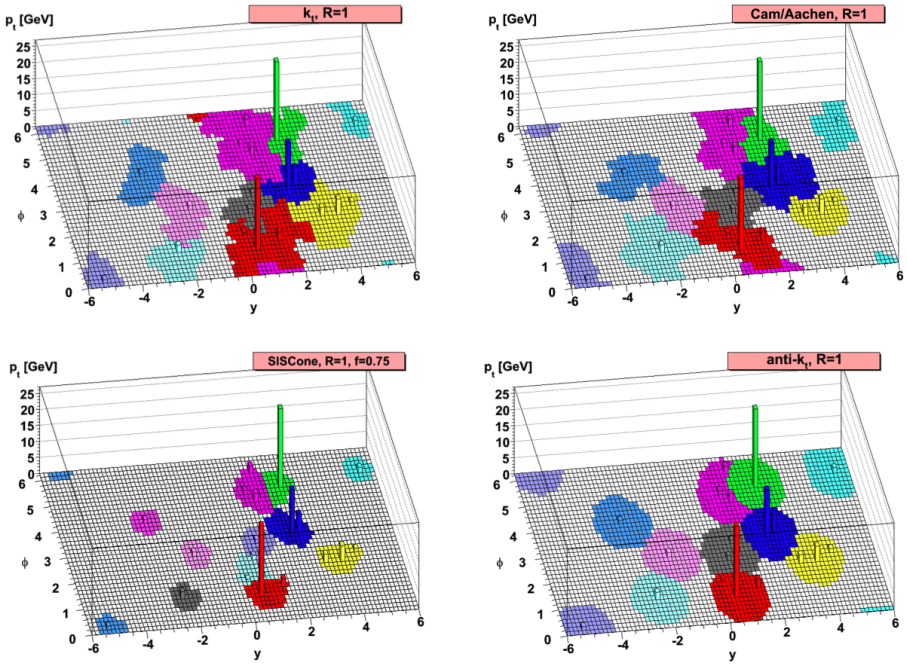


Figure 3.3 Example of clustered jets with various algorithms. The *top left* is the k_t algorithm, the *top right* is the Cambridge/Aachen algorithm, the *bottom left* is the SISCone algorithm, and the *bottom right* is the anti-k_t algorithm.

The algorithm is as follows [5]:

1. Start with the hardest object and call it an *entity*.
2. Calculate d_{ij} between entity i and object j.
3. If $d_{ij} < d_{i\mathrm{B}}$, add object j to the entity i (add their 4-momenta) and continue with the next object, going back to step 2.
4. Else call the entity i a jet and exclude it from further consideration.
5. Continue until all objects are clustered into jets.

[2]In the original paper, authors refer to the transverse momentum p_{T} as k_t. Anti-k_t is the reciprocal transverse momentum.

An example of clustered jets with various algorithms is shown in figure 3.3.

In our study, the object referred to in the algorithm descriptions are PFOs . The anti-k_t algorithm ($p = -1$) is widely used in the ATLAS community due to its IRC safety (as defined in definition 1) and symmetrical jet shapes. We shall use the anti-k_t algorithm with $R = 0.4$ for all jet reconstructions.

Chapter 4

Deep Learning Architectures

4.1 Basic Concepts

Machine Learning is a field of computer science that uses statistical techniques to allow computers to learn without being explicitly programmed. *Deep Learning* is a subfield of Machine Learning focusing on the use of neural networks [1]. The conceptual difference between Machine Learning and Deep Learning is:

***Machine Learning** is about learning general features.*
***Deep Learning** is about learning abstract concepts.*

To emphasize the difference even more, consider BDT (see for example [10]), which is an old-fashioned Machine Learning model, and Transformer (see section 4.5), a modern Deep Learning model. In the context of particle physics, BDT is trained to provide learned cuts on the input data to separate the signal from the background. On the other hand, Transformer is learned to form an *abstract representation* of the input data, from which one can extract information about the signal/background (this is even more apparent in the *Particle Transformer* (ParT) and the *Dynamicaly Enhanced Particle Transformer* (DeParT) , see sections 4.6 and 4.7). It can learn not only the differences between signal and background but also *general physical concepts* [23].

Deep Learning builds from the *Multilayer Perceptron* (MLP) [19], a multi-layered neural network able to learn a *any* non-linear function from input data. In detail, we will discuss more enhanced MLP in section 4.2.

The general idea of Deep Learning is to have some model, which is a set of parameters and functions, taking a set of inputs and producing a set of outputs.

[1] In some papers, they emphasize the difference between neural networks and *artificial neural networks* since we are not interested in studying *human neural networks*, we will only be using the term neural networks.

The model is trained by adjusting the parameters to fit the desired outputs. In this context, we are interested in the *supervised* learning [53], where the desired outputs are known (from MC simulations, for example). Some new use cases exist for *self-supervised* learning [54], where the desired outputs are unknown, but we will only briefly touch on them as a prospect.

4.1.1 Forward and Backward Passes

Let $\boldsymbol{x}$ be the input (generally a multi-dimensional tensor[2]) of a model $f(\bullet; \boldsymbol{w})$ with trainable parameters $\boldsymbol{w}$. We will call the *forward pass* of the model an application of the model on the input data $\boldsymbol{o} = f(\boldsymbol{x}; \boldsymbol{w})$, where $\boldsymbol{o}$ is the output of the model (inference on input data $\boldsymbol{x}$). The *backward pass* or *backpropagation*[3] is the process of adjusting the parameters $\boldsymbol{w}$ to fit the desired output $\boldsymbol{y}$. This is done by calculating the *loss* $L(\boldsymbol{o}, \boldsymbol{y})$ between the output $\boldsymbol{o} = f(\boldsymbol{x}; \boldsymbol{w})$ and the desired output $\boldsymbol{y}$ (*target*), and then adjusting the parameters $\boldsymbol{w}$ to minimize the loss. Loss is a measure of how far the model is from the target.

There are several ways of adjusting the parameters, but the widely used approaches are based on computing the gradient of the loss wrt. the parameters $\boldsymbol{w}$ and then updating the parameters in the direction of the gradient [53]. This can be sketched as follows

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \mathcal{O}\left[\nabla_{\boldsymbol{w}} L(f(\boldsymbol{x}; \boldsymbol{w}), \boldsymbol{y})\right], \tag{4.1}$$

where α is the *learning rate* and $\mathcal{O}$ is the optimization algorithm (function), shortly *optimizer*. The loss is usually calculated on a *batch* of data, making the process more efficient and parallelizable, which is then averaged to get the final loss

$$\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \cdot \mathcal{O}\left[\nabla_{\boldsymbol{w}} \frac{1}{B} \sum_{i=1}^{B} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{w}), \boldsymbol{y}^{(i)})\right], \tag{4.2}$$

where B is the batch size and $\boldsymbol{x}^{(i)}$ and $\boldsymbol{y}^{(i)}$ are the i-th elements of the batch.

4.1.2 Traning process

Before discussing the individual models, we will briefly touch on the training process as a whole.

[2]This is NOT the same tensor as a physical tensor, which has a prescribed way of transforming, but rather a multi-dimensional array of numbers.

[3]Term backpropagation comes from the application of the chain rule. Imagine the model as a composite of many functions, as the derivatives of the loss wrt. some parameters are computed, the chain rule is used to get *back* to that parameter.

The dataset is split into a *training set*, a *validation set*, and in our case, also a *test set*. The training set is used to train the model, the validation set is used to monitor the training process, and the test set is used to evaluate the model's performance. The training process is split into several *epochs*, each consisting of several *iterations*. An epoch is a complete pass through the training data, while an iteration through a single batch of data. The total number of *steps* is the product of the number of epochs and iterations.

After each epoch, we evaluate the model on the training set and the validation set. Validation is done to have a statistically independent measure of the model's performance during the training process.

4.1.3 Output layer

Based on the type of the target output, we can distinguish between *regression* and *classification* problems.

Classification is the task of predicting a discrete variable. The target variable is a vector of numbers $\boldsymbol{t}$, where the length of the vector is the number of classes. It is a *one-hot* encoded vector, meaning that all the inputs are zero except for the one corresponding to the correct class, which is one. To perform a classification task, we must convert the vector $\boldsymbol{v}$ into a probability distribution over the classes. This is done by applying the *softmax* function to the vector $\boldsymbol{v}$, which is defined as

$$\boldsymbol{p} = \mathrm{softmax}(\boldsymbol{v}) = \frac{\exp(\boldsymbol{v})}{\sum_{i=1}^{N} \exp(v_i)} = \frac{\exp(\boldsymbol{v})}{||\exp(\boldsymbol{v})||}, \tag{4.3}$$

where N is the number of classes. In a case of binary classification (only two classes), the vector is a single number v, and the softmax function reduces to a *sigmoid* function

$$p = \sigma(v) = \frac{1}{1 + \exp(-v)}. \tag{4.4}$$

Ideally, we would like the vector $\boldsymbol{p}$ to have a single non-zero element corresponding to the correct class. In the case of binary classification, we would like the number p to be either zero or one.

Regression is the task of predicting a continuous variable. The model's output is a vector of numbers $\boldsymbol{v}$, where the length of the vector is the number of variables to be predicted. Usually there is no output layer function for regression.

4.1.4 Loss Functions

The loss function measures how far the model is from the target. In the old days of Machine Learning , the most common loss function was the *mean square er-*

ror (MSE) , defined as

$$L(\boldsymbol{o}, \boldsymbol{y}) = \frac{1}{N} \sum_{i=1}^{N} (o_i - y_i)^2, \tag{4.5}$$

where N is the number of variables to be predicted (number of classes, in case of classification task).

In modern Deep Learning frameworks, the *cross-entropy* loss is the most common loss function, rooting from the *information theory*. Before talking about the cross-entropy loss, we will introduce some basic concepts from information theory [55]. We will denote the probability of a random variable $x \in X$ by $p(x)$, where X is the set of all possible values of the random variable.

Self-information of a random variable x is defined as

$$I_p(x) = \log \frac{1}{p(x)} = -\log p(x). \tag{4.6}$$

The self-information measures the *surprise* of obtaining random variable x when sampled from p.

Entropy of distribution p is defined as a mean self-information of the distribution

$$H(p) = \langle I_p(x) \rangle_p = -\sum_{x \in X} p(x) \log p(x). \tag{4.7}$$

The entropy measures the *average surprise* of obtaining a random variable from the distribution p. The entropy is zero if the random variables are deterministic and maximal if the random variables are uniformly distributed.

Cross-entropy is defined as a mean value of the self-information of the distribution q when sampled from the distribution p

$$H(p, q) = \langle I_q(x) \rangle_p = -\sum_{x \in X} p(x) \log q(x). \tag{4.8}$$

The cross-entropy measures the *average surprise of distribution* q when the actual distribution is p. Because of the *Gibbs inequality* $H(p, q) \geq H(p)$, [56], the cross-entropy is minimal if the distribution q is equal to the distribution p.

Kullback-Leibler (KL) divergence is a difference between the cross-entropy of the distribution p and q and the entropy of the distribution p.

$$D_{\mathrm{KL}}(p|q) = H(p, q) - H(p) = -\sum_{x \in X} p(x) \log \frac{p(x)}{q(x)}. \tag{4.9}$$

The KL divergence measures the *excess of the average surprise of distribution* p when the actual distribution is p.

We can now define the *cross-entropy loss* function

$$L(\boldsymbol{o}, \boldsymbol{y}) = -\sum_{i=1}^{N} y_i \log o_i. \tag{4.10}$$

It is important to note that cross-entropy is defined for probability distributions so that the cross-entropy loss can be used only for classification tasks.

This is an excellent place to discuss the similarities to physical entropy. In modern statistical physics, the definition of classical entropy roots exactly from Shannon [55] as in the information theory. It is the *measure of ignorance*, in the sense of missing information. In this context, we can say that the cross-entropy is the *measure of ignorance* of the model. In quantum mechanics, entropy was developed by John von Neumann [57], where the definition is slightly different, but it is the same regarding probability distributions (wave functions). The KL divergence is precisely the *measure of the distinguishability of two quantum states*.

4.1.5 Optimizers

Optimizers are algorithms that adjust the model weights to minimize the loss function. The most common optimizers are based on the *gradient descent* and its variants [58]. All of them have a common hyperparameter, the *learning rate* α, which controls the size of the steps taken by the optimizer. We introduce three common optimizers:

SGD (*Stochastic Gradient Descent*) [53] is an old-school Machine Learning algorithm.

Algorithm 1 SGD

Input: model with weights $f(\bullet; \boldsymbol{w})$, outputs of a model $f(\boldsymbol{x}^{(i)}; \boldsymbol{w})$ in batches, target values $\boldsymbol{y}^{(i)}$ in batches
Input: learning rate α
  for n = 1 **to** `number of iterations` **do**
    $\boldsymbol{g} \leftarrow \nabla_{\boldsymbol{w}} \frac{1}{B} \sum_{i=1}^{B} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{w}), \boldsymbol{y}^{(i)})$
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \boldsymbol{g}$
  end for
Output: model $f(\bullet; \boldsymbol{w})$ with optimized weights $\boldsymbol{w}$

Adam (*Adaptive Moment Estimation*) [59] is a popular Deep Learning algorithm used in almost every Deep Learning framework. As hyperparameters, apart from the learning rate α, it has the decay rate of the first moment β_1, the

decay rate of the second moment β_2, a small constant ε to avoid division by zero, and weight decay λ (which we will explain in section 4.1.7 as a regularization technique that forces the weights to be smaller). There are

Algorithm 2 Adam

Input: model with weights $f(\bullet; \boldsymbol{w})$, outputs of a model $f(\boldsymbol{x}^{(i)}; \boldsymbol{w})$ in batches, target values $\boldsymbol{y}^{(i)}$ in batches

Input: learning rate α, decay rate of the first moment β_1, decay rate of the second moment β_2, small constant ε, weight decay λ

1: **for** n = 1 **to** `number of iterations` **do**
2: $\quad \boldsymbol{g} \leftarrow \nabla_{\boldsymbol{w}} \frac{1}{B} \sum_{i=1}^{B} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{w}), \boldsymbol{y}^{(i)})$
3: $\quad \boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g}$
4: $\quad \boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{g}^2$
5: $\quad \boldsymbol{s} \leftarrow \frac{\boldsymbol{s}}{1-\beta_1^n}$
6: $\quad \boldsymbol{r} \leftarrow \frac{\boldsymbol{r}}{1-\beta_2^n}$
7: $\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left(\frac{\boldsymbol{s}}{\sqrt{\boldsymbol{r}}+\varepsilon} + \lambda \boldsymbol{w} \right)$
8: **end for**

Output: model $f(\bullet; \boldsymbol{w})$ with optimized weights $\boldsymbol{w}$

several notable differences between Adam and SGD.

First, Adam uses the *momentum* (remembering the size of the previous step) to accelerate the convergence. Intermediate variable $\boldsymbol{s}$ is introduced as the sum of the previous $\boldsymbol{s}$ and the current gradient $\boldsymbol{g}$ in line 3 of algorithm 2. The momentum is the previous step's $\boldsymbol{s}$. β_1 is the kept fraction of the last step, usually set close to 1.

Second, Adam uses the *adaptive learning rate* to avoid the problem of the learning rate being too small or too large. Intermediate variable $\boldsymbol{r}$ is introduced as the sum of the previous $\boldsymbol{r}$ and the square of the current gradient $\boldsymbol{g}$ in line 4 of algorithm 2. The learning rate is scaled by the inverse of the square root of $\boldsymbol{r}$ (the operation of square root and inverse is done element-wise on the vector $\boldsymbol{r}$), adapting the learning rate to the size of the gradient $\boldsymbol{g}^2$ in line 7 of algorithm 2 (the square of the gradient is used to assert positive values). On top of that, $\boldsymbol{r}$ also has momentum as seen in line 4 of algorithm 2, where β_2 (usually set close to 1) is the fraction of the previous $\boldsymbol{r}$ that is kept.

The operations in lines 5 and 6 of algorithm 2, called *bias correction*, are used to avoid the initial bias of $\boldsymbol{s}$ and $\boldsymbol{r}$. At the beginning of the training, the values of $\boldsymbol{s}$ and $\boldsymbol{r}$ are zero, so the first step would be close to zero since

$1 - \beta_1$ and $1 - \beta_2$ are close to zero. To correct this, $\boldsymbol{s}$ and $\boldsymbol{r}$ are divided by $1 - \beta_i^n$ which makes the first steps larger.

LAMB (*Layer-wise Adaptive Moments optimizer for Batch training*) [60] is a state-of-the-art optimizer algorithm built upon Adam, which helps train large models. It has the same hyperparameters as Adam. LAMB introduces the

Algorithm 3 LAMB

Input: model with weights $f(\bullet; \boldsymbol{w})$, outputs of a model $f(\boldsymbol{x}^{(i)}; \boldsymbol{w})$ in batches, target values $\boldsymbol{y}^{(i)}$ in batches

Input: learning rate α, decay rate of the first moment β_1, decay rate of the second moment β_2, small constant ε, weight decay λ

1: **for** n = 1 **to** `number of iterations` **do**
2: $\quad \boldsymbol{g} \leftarrow \nabla_{\boldsymbol{w}} \frac{1}{B} \sum_{i=1}^{B} L(f(\boldsymbol{x}^{(i)}; \boldsymbol{w}), \boldsymbol{y}^{(i)})$
3: $\quad \boldsymbol{s} \leftarrow \beta_1 \boldsymbol{s} + (1 - \beta_1)\boldsymbol{g}$
4: $\quad \boldsymbol{r} \leftarrow \beta_2 \boldsymbol{r} + (1 - \beta_2)\boldsymbol{g}^2$
5: $\quad \boldsymbol{s} \leftarrow \frac{\boldsymbol{s}}{1-\beta_1^n}$
6: $\quad \boldsymbol{r} \leftarrow \frac{\boldsymbol{r}}{1-\beta_2^n}$
7: $\quad \boldsymbol{m} \leftarrow \frac{\boldsymbol{s}}{\sqrt{\boldsymbol{r}}+\varepsilon} + \lambda \boldsymbol{w}$
8: $\quad \boldsymbol{w} \leftarrow \boldsymbol{w} - \alpha \left\|\boldsymbol{w}\right\| \frac{m}{\|m\|}$
9: **end for**

Output: model $f(\bullet; \boldsymbol{w})$ with optimized weights $\boldsymbol{w}$

normalization of the step, which can be seen in line 8 of algorithm 3. This normalization is done layer-wise to ensure that the weights will not diverge. The second improvement is the scaling based on the size of the weights, which can be seen in line 8 of algorithm 3 as $\|\boldsymbol{w}\|$. Both normalizations are l_2-norms [60].

Learning Rate Scheduling

Learning Rate Scheduling is a process of changing the learning rate during the training process. Usually, it is done as a decay of the learning rate to allow the model to converge into the absolute minimum of the loss function. Let us denote the learning rate at the n-th iteration as α_n and N as the total number of steps. There are three common methods of learning rate decay:

- **Linear** $\alpha_n = \alpha_0 \left(1 - \frac{n}{N}\right)$,
- **Exponential** $\alpha_n = \alpha_0 \cdot d^n$, where $d \in (0, 1)$ is a number,
- **Cosine** $\alpha_n = \alpha_0 \cdot \frac{1}{2} \left(1 + \cos\left(\frac{\pi n}{N}\right)\right)$.

Exploding Gradient

Huge models tend to diverge as they have a lot of parameters adjusted at the same time, in a process called *exploding gradient*. There are two common methods to prevent this [53]:

- **Gradient Clipping** allows a maximum value c of the norm of the gradient $\|\boldsymbol{g}\|$, if it is larger than c, it is scaled down to $c\frac{\boldsymbol{g}}{\|\boldsymbol{g}\|}$.
- **Warmup** is a method that increases the learning rate from zero to the desired value (usually linear) at the beginning of the training in some number of *warmup steps*.

4.1.6 Activation Functions

Deep Learning neural networks are composed of two types of operations: *linear* and *non-linear*. Linear (more precisely, affine) operations are basic matrix multiplications. Non-linear operations are the *activation functions*, which are applied element-wise to the output of the linear operations. Alternating between linear and non-linear operations is the key to the power of Deep Learning neural networks, which can approximate *any* function with enough layers and learned parameters. A *Universal Approximation Theorem* [61] states that a simple FC Network (see section 4.2) with one hidden layer containing a finite number of neurons can approximate any continuous function on a compact subset of $\mathbb{R}^n$ to any desired accuracy.

In this section, we will briefly discuss the most common activation functions:

- **tanh** - old-fashioned Machine Learning activation [4]

$$\tanh(x) = \frac{e^x - e^{-x}}{e^x + e^{-x}} \tag{4.11}$$

- **ReLU** (Rectified Linear Unit) - most common, 'goto' Deep Learning activation

$$\text{ReLU}(x) = \max(0, x), \tag{4.12}$$

- **GELU** (Gaussian Error Linear Unit) [63] - smooth version of ReLU,

$$\text{GELU}(x) = x\Phi(x) = x\frac{1}{2}\left(1 + \text{erf}\left(\frac{x}{\sqrt{2}}\right)\right), \tag{4.13}$$

  where $\Phi(x)$ is the cumulative distribution function of the standard normal distribution and $\text{erf}(x)$ is the error function,

[4] However it is still being used in Deep Learning , for example in the RNN[62].

- **Swish or SiLU** (Sigmoid Linear Unit) [64] - another smooth version of ReLU,

$$\text{Swish}(x) = x\sigma(x) = \frac{x}{1 + e^{-x}}, \tag{4.14}$$

- **Sigmoid** - used after the last layer to convert a number to the probability

$$\sigma(x) = \frac{1}{1 + e^{-x}}, \tag{4.15}$$

- **Softmax** - used after the last layer to convert a vector to a probability distribution

$$\text{softmax}(\boldsymbol{x}) = \frac{e^{\boldsymbol{x}}}{\sum_{j=1}^{n} e^{x_j}}. \tag{4.16}$$

Sigmoid and Softmax were discussed in section 4.1.3. Graph with the activation functions can be seen in figure 4.1 (softmax is excluded since in one dimension it reduces to sigmoid),

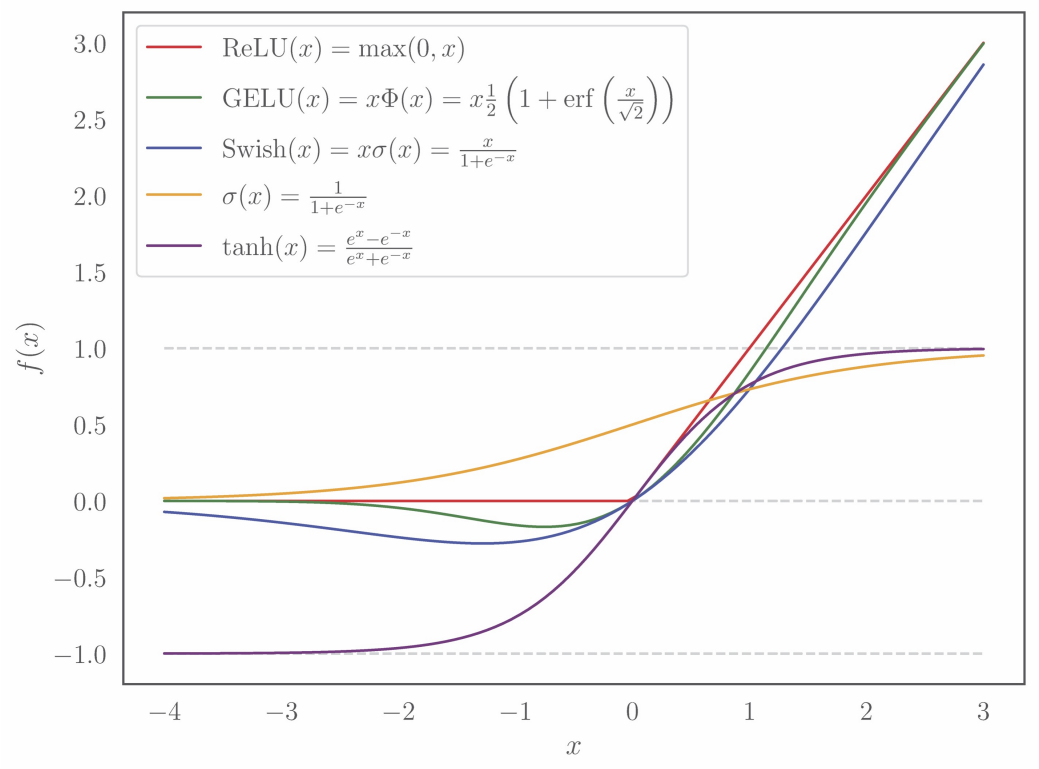


Figure 4.1 Activation functions

4.1.7 Regularization

Overfitting happens when the model memorizes the training data instead of learning the underlying pattern, which is why we need the validation dataset to estimate the *generalization error*. **Underfitting** is the opposite when the model is not trained enough to capture the underlying pattern. These terms are generally used across all Machine Learning algorithms.

Remarkably, deep networks performance *increases* with the number of parameters without overfitting. There is a study [65] showing that if you pass the classical regime of 'Bias-Variance Tradeoff' (a more complex model has a lower bias but higher variance) with the number of parameters, the modern regime of 'Larger Model is Better' goes forever. This is another proof of the statements in section 4.1 saying that Deep Learning models can learn abstract concepts.

However, to be able not to overfit the model, we need to use **regularization**. Regularization is a technique that prevents the model from overfitting by obstructing or penalizing the learning process. We will list the most common regularization techniques:

Weight decay or **L2 regularization** is a penalty added to the loss function that is a l_2-norm of the weights.

$$L \leftarrow L + \lambda \left\| \boldsymbol{w} \right\|^2, \tag{4.17}$$

where λ is the regularization strength. This regularization is usually implemented into the optimizer because the gradient can be explicitly calculated. See section 4.1.5.

Dropout is a technique that randomly sets some of the weights to zero *during training*. The probability of a weight being set to zero is the only hyperparameter. Dropout forces the network to learn different data representations and not rely on individual neurons.

Label smoothing is a technique that replaces the one-hot encoded labels with a distribution of probabilities. For example, if the target is $\boldsymbol{y} = (0, 1, 0)$ (second class), and we set the hyperparameter of label smoothing to 0.1, the new label is $\tilde{\boldsymbol{y}} = (0.1, 0.8, 0.1)$.

Augmentation is a technique that applies random transformations to the input data. For example, if we have an image as an input, we can randomly rotate it, crop it, and change its brightness.

Ensambling is a technique that combines multiple models into one by averaging their predictions.

4.1.8 Metrics

A *metric* measures how well the model performs. It *evaluates* the model. We will only discuss the metrics for binary classification since we will use those. The output of a binary classification model is a single number between zero and one $o \in [0, 1]$, where zero corresponds to the first class, and one corresponds to the second class. We make a *threshold* $\tau \in (0, 1)$, where we assign the output to the first class if $o < \tau$ and to the second class if $o \geq \tau$. Usually, the threshold is set to 0.5, but it can be adjusted to optimize the desired output of the model. Going further, we will assume that the threshold is set to 0.5 unless stated otherwise.

Let us denote the number of inputs that are correctly classified as the first class T_0, the number of inputs that are correctly classified as the second class T_1, the number of inputs that are incorrectly classified as the first class F_0, the number of inputs that are incorrectly classified as the second class F_1. Usually, these quantities are denoted as *true negatives* (TN), *true positives* (TP), *false negatives* (FN), and *false positives* (FP), respectively. However, they are confusing and refer to one class as positive and the other as negative, which is not the case in our binary classification, where both classes are equally important. We try to simplify the notation by using the notation *true* T and *false* F to denote the correct and incorrect classification, respectively, and the numbers 0 and 1 to indicate the output class.

A diagram of the following quantities is shown in figure 4.2. We can now define the following metrics with a *fixed threshold*:

Accuracy is the fraction of correctly classified events

$$\text{Accuracy} = \frac{T_0 + T_1}{T_0 + T_1 + F_0 + F_1}. \tag{4.18}$$

Or generally a probability of correct classification.

Efficiency ε_i of class i is the fraction of correctly identifying the class i

$$\varepsilon_i = \frac{T_i}{T_i + F_i}. \tag{4.19}$$

This metric is referred to by various terms if $i = 1$: *true positive rate* (TPR) sensitivity, recall, hit rate, probability of detection...

Rejection ε_i^{-1} of class i is the inverse of the fraction of correctly identifying the class i

$$\varepsilon_i^{-1} = \frac{1}{\varepsilon_i} = \frac{T_i + F_i}{T_i}. \tag{4.20}$$

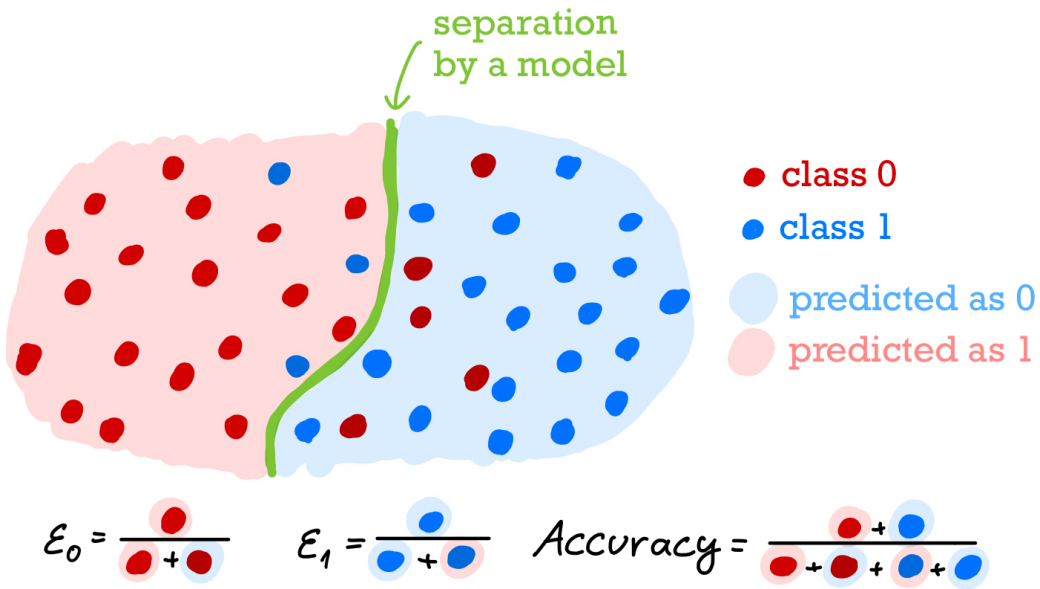


Figure 4.2 Visualization of the metrics. Red dots correspond to class 0, and blue dots to class 1. The model is represented by a green line that separates the two classes. Red highlighted dots were tagged by the model as class 0, and blue highlighted dots as class 1. In formulas at the bottom, the highlighted dots represent the number of total dots with attributes given by the color and highlight.

False Rate φ_i of class i is the fraction of incorrectly identifying the class i

$$\varphi_i = \frac{F_i}{T_i + F_i} \tag{4.21}$$

This metric is referred to by various terms if $i = 1$: *false positive rate* (FPR), fall-out, probability of false alarm...

Confusion Matrix (CM) is a plot where on the x-axis are the predicted classes, and on the y-axis are the true classes. The CM is a good way to visualize the performance of the model. The diagonal represents the efficiencies of a given class and the off-diagonal represents the false rates of the given class.

Following metrics are defined with a *variable threshold*:

ROC (Receiver Operating Characteristic) curve is the plot of the efficiency ε_1 against the false rate φ_1 for different thresholds. It is good to note that there is no point in evaluating it for the second class since the ROC curve would reflect wrt—the diagonal.

AUC is the Area Under the ROC Curve, a simple integration of the ROC curve.

Rejection at Efficiency $\varepsilon_i^{-1}@_x\varepsilon_j$ is the rejection of the one class i at a given efficiency x of the second class j with an adjusted threshold.

4.2 Fully Connected Network

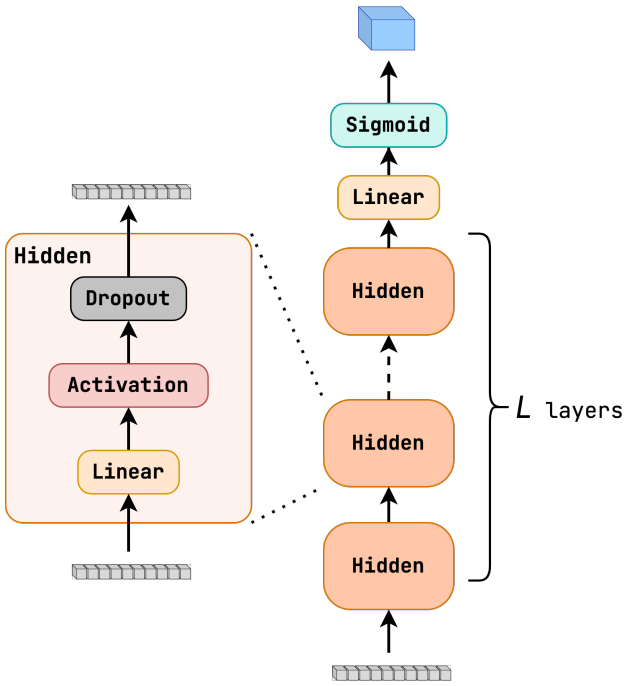


Figure 4.3 Diagram of the Fully Connected Network.

Fully Connected Network (FC) [19] is the most straightforward neural network, where all the neurons in one layer are connected to all the neurons in the previous and next layers. It is the same as MLP [19], but instead of $\tanh$ activation, we use newer activation functions described in section 4.1.6.

The input is a vector $\boldsymbol{x}^{(0)} \in \mathbb{R}^{n_0}$ (input layer) and the output is a vector $\boldsymbol{o} \equiv \boldsymbol{x}^{(L+1)} \in \mathbb{R}^{n_{L+1}}$ (output layer), where n_{L+1} is the number of classes. Between input and output layers there are L hidden layers $\boldsymbol{x}^{(l)} \in \mathbb{R}^{n_l}$, where n_l is the number of neurons in the l-th layer, i.e. *layer size*. The output of the l-th layer is calculated as

$$\boldsymbol{x}^{(l+1)} = f(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}), \tag{4.22}$$

where f is the activation function, $\boldsymbol{W}^{(l)} \in \mathbb{R}^{n_l \times n_{l-1}}$ is the weight matrix (trainable parameters), and $\boldsymbol{b}^{(l)} \in \mathbb{R}^{n_l}$ is the bias vector (trainable parameters). On the last layer, the softmax or sigmoid activation function is used. After each layer, we apply the dropout. The activation $f : \mathbb{R} \to \mathbb{R}$ is applied element-wise to the

vector $\boldsymbol{x}^{(l)}$ (the only exception is the softmax activation, which is applied on the whole vector to normalize it). Going further we always assume that the activation function is applied element-wise. A diagram of the FC Network is shown in figure 4.3. `Linear` corresponds to the matrix multiplication and bias addition, and `Hidden` to the sequential application of the `Linear` layer `Activation` and `Dropout`. At the output, there is no dropout.

A more artistic diagram of the FC Network is shown in figure 4.4.

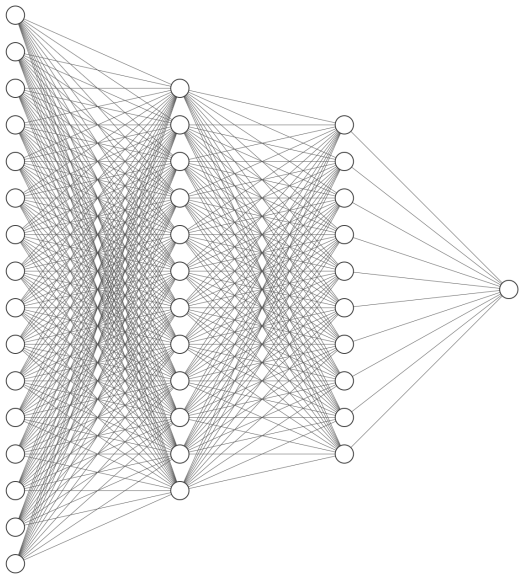


Figure 4.4 Artistic diagram of the Fully Connected Network[5]. Each node is a *neuron* corresponding to a number in a vector $\boldsymbol{x}^{(l)}$, and each edge is a weight corresponding to a number in matrix $\boldsymbol{W}^{(l)}$.

In our case, the input vector $\boldsymbol{x}^{(0)}$ are *high-level jet variables* and the output vector is just one number o, which is the probability of the jet being a quark (1) or gluon (0). This type of network allows us to utilize the reconstructed jet, whose properties are given by the PFOs . However, it **cannot utilize** the information about the **PFOs** themselves.

4.3 Highway Network

Highway Network [18] is an extension of the FC Network network, that introduces a *gate* to the network. The gate is a *bypass* that allows the output of one layer to pass through the next layer without any modification. Another term for the

[5] `https://github.com/ashishpatel26/Tools-to-Design-or-Visualize-Architecture-of-Neural-Network`.

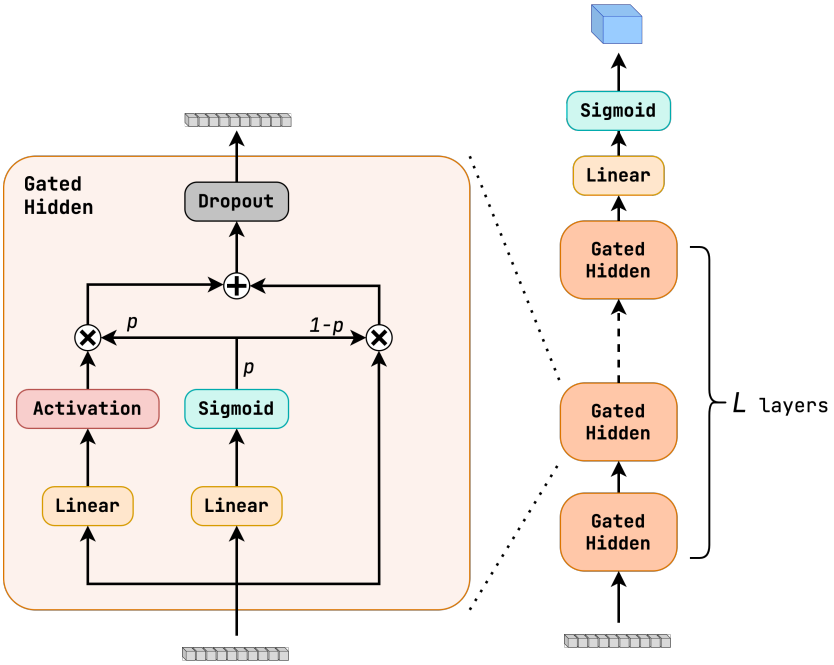


Figure 4.5 Diagram of the Highway Network.

bypass is a *residual connection*. The gate is a *learnable* function controlling the amount of data bypassing the layer. It multiplies the output of the hidden layer by a probability p and adds the input of the layer multiplied by $1 - p$

$$\boldsymbol{x}^{(l+1)} = pf(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}) + (1-p)\boldsymbol{x}^{(l)}. \tag{4.23}$$

The trick is estimating the probability p of using the newly calculated output rather than the input. It is calculated from the input vector $\boldsymbol{x}^{(l)}$ and another weight matrix $\boldsymbol{W}_{\text{g}}^{(l)}$ (and bias $\boldsymbol{b}_{\text{g}}^{(l)}$) followed by a sigmoid activation function, whose output is a probability.

$$\boldsymbol{p}^{(l)} = \sigma(\boldsymbol{W}_{\text{g}}^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}_{\text{g}}^{(l)}), \tag{4.24}$$

where the probabilities are calculated for each element of the vector $\boldsymbol{x}^{(l)}$ separately and are different for each layer. The output of the Highway Network layer is then calculated as

$$\boldsymbol{x}^{(l+1)} = \boldsymbol{p}^{(l)} \odot f(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)} + \boldsymbol{b}^{(l)}) + (1 - \boldsymbol{p}^{(l)}) \odot \boldsymbol{x}^{(l)}, \tag{4.25}$$

where $\odot$ is the element-wise multiplication. If we put it all together, one layer of Highway Network can be written as

$$\boldsymbol{x}^{(l+1)} = \sigma(\boldsymbol{W}_{\text{g}}^{(l)}\boldsymbol{x}^{(l)}+\boldsymbol{b}_{\text{g}}^{(l)})\odot f(\boldsymbol{W}^{(l)}\boldsymbol{x}^{(l)}+\boldsymbol{b}^{(l)})+(1-\sigma(\boldsymbol{W}_{\text{g}}^{(l)}\boldsymbol{x}^{(l)}+\boldsymbol{b}_{\text{g}}^{(l)}))\odot\boldsymbol{x}^{(l)}. \tag{4.26}$$

Everything else (output, dropout, activation function) is the same as in the FC Network . The diagram of the Highway Network is shown in figure 4.5.

Highway Network uses the same input as the FC Network network, i.e. the *high-level jet variables*. The only advantage of the Highway Network network is that it is **more stable** when training the network with many layers.

4.4 Particle Flow and Energy Flow Network

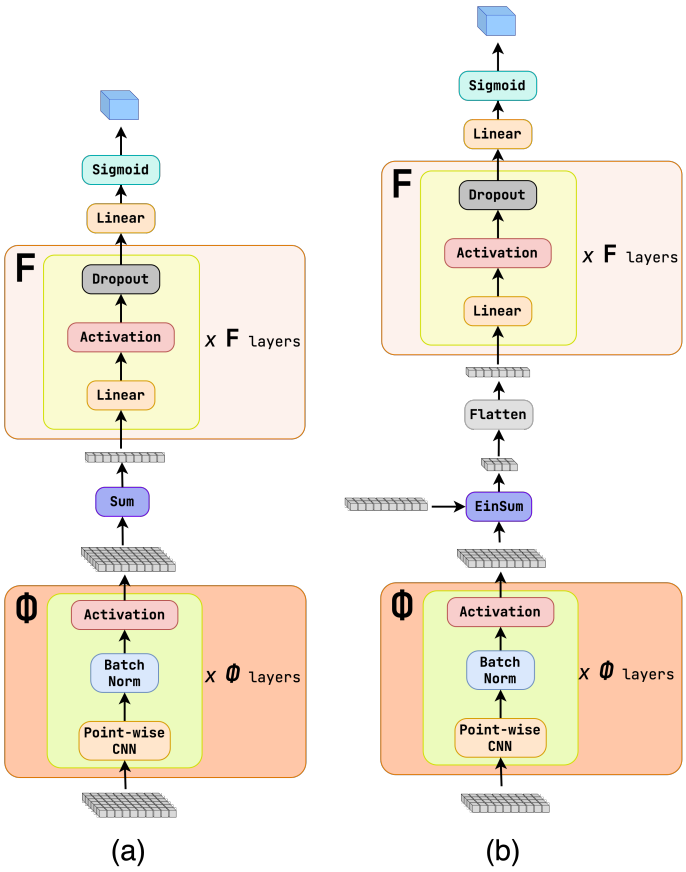


Figure 4.6 Diagram of PFN (a) and EFN (b). The `Einsum` layer corresponds to `tf.einsum` function and Point-wise CNN to `tf.keras.layers.Conv1D(filters=layer_size, kernel_size=1)`.

Particle Flow Network (PFN) and *Energy Flow Network* (EFN) [20] are two networks that are designed to utilize the information about the PFOs . They are developed specifically for HEP , using the theory of *Deep Sets* [66].

The general idea has three steps:

1. *Per PFO Mapping*

2. *Summing over PFOs*

3. *Constructing Observable*

In the case of PFN , this can be expressed in the form

$$\mathcal{O}_{\text{PFN}} = F\left(\sum_{i=1}^{N} \Phi(\boldsymbol{z}_i, \boldsymbol{\varphi}_i)\right), \tag{4.27}$$

where the function F constructs the *observable* $\mathcal{O}$, Φ is the *per* Particle Flow Object *(PFO) mapping*, $\boldsymbol{z}_i$ is a vector of PFO energy properties (p_{T}, E) and $\boldsymbol{\varphi}_i$ is a vector of PFO angular properties (η, ϕ).

EFN is similar to PFN but is specifically designed to be IRC safe (see definition 1)

$$\mathcal{O}_{\text{EFN}} = F\left(\sum_{i=1}^{N} \boldsymbol{z}_i \Phi(\boldsymbol{\varphi}_i)\right). \tag{4.28}$$

The mapping Φ is the same as in PFN , but the PFO energy properties are *not used*. The linearity in $\boldsymbol{z}_i$ satisfies the IRC conditions.

To construct the mapping Φ and the function F, the authors [20] use the *Universal Approximation Theorem* [61] discussed in section 4.1.6 to make the argument that FC Network is sufficient to approximate any function. For the F function, we use the architecture of the FC Network network with F`_layers` a number of hidden layers, but for the Φ function, we use the *point-wise Convolutional Neural Network* (point-wise CNN) [67], which is a slight improvement over the FC Network network still agreeing with (4.27) or (4.28). The point-wise CNN is followed by *batch normalization* and activation to form a block that is repeated Φ`_layer` times.

In the case of EFN , the summation over the PFOs is a matrix multiplication over the PFO index. The output is a matrix (because both $\boldsymbol{z}_i$ and $\boldsymbol{\varphi}_i$ are matrices) which is then flattened to a vector.[6] This is not the case in the original paper, but this modification's performance is much better, still satisfying the IRC conditions and the form of (4.28). The accompanying diagram of both PFN and EFN is shown in figure 4.6.

The main advantage of PFN and EFN is that they are designed to use the information about the PFOs , which is not the case for the FC Network or Highway Network . On top of that, EFN is IRC-safe, which is crucial in the case of HEP . Another advantage of IRC safe models is the stability of performance between

[6] This is done using the `tf.einsum` function, hence the name in figure 4.6, followed by a `tf.reshape`.

data from different physics simulation frameworks [68]. However, the biggest downside of PFN and EFN is that they **do not allow** the **PFOs to communicate** and share information.

4.5 Transformer

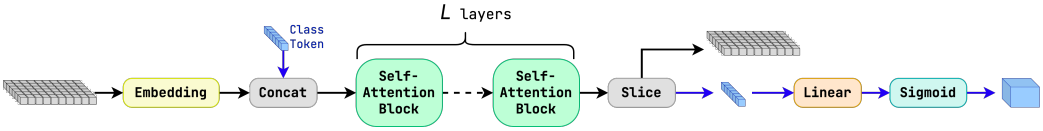


Figure 4.7 Diagram of the Transformer. The Class Token is represented as a blue row of boxes. Operations concerning the Class Token are traced with a blue arrow.

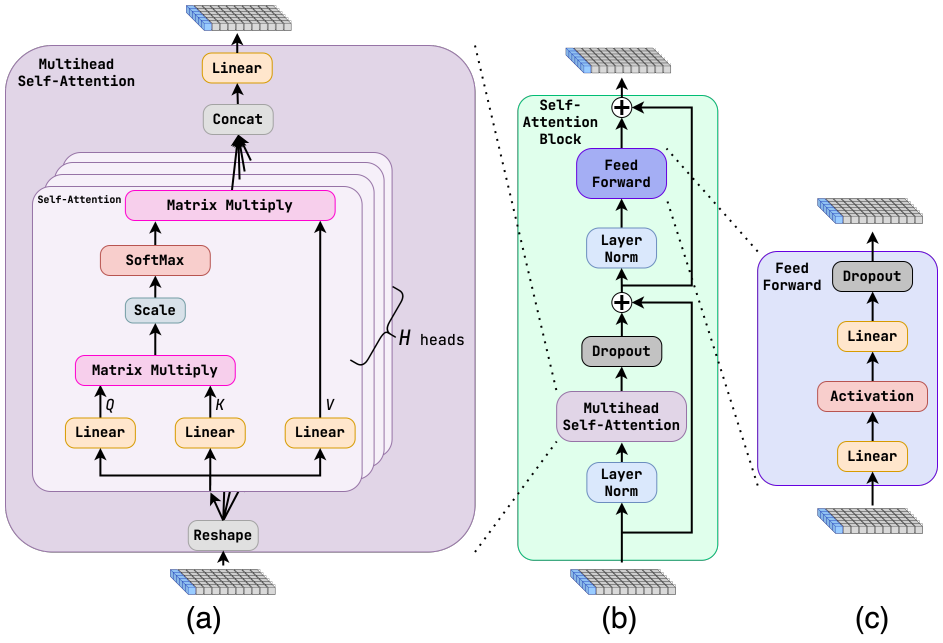


Figure 4.8 Diagram of the Transformer layers. (b) is the *Self-Attention Block*, with the main components being the *Multihead Self-Attention* (a) and the *Feed Forward Network* (c). The Class Token is represented as a blue row of boxes.

Transformer [21] is a network designed to learn from set data with fixed *embedding* dimension. It is based on the *Attention* mechanism, which is a core component of Transformer network, allowing the input elements to **communicate** with each other. The overall structure of the original Transformer consists of *Encoder* and *Decoder* blocks, but in our use case, we only use the Encoder part,

which is shown in figure 4.7. The Encoder part is supposed to form a *representation* of the input data based on the task. On the other hand, the Decoder part reconstructs some output data from the internal representation of the Encoder part. To give a more concrete example, imagine a translation between languages such as Czech and Slovak. The Encoder represents what the input Czech sentence is trying to say, i.e., the abstract information, and the Decoder reconstructs the Slovak sentence from the internal representation of the Encoder. As stated, we are only interested in forming the internal representation of the input data, so we only use the Encoder part of the Transformer , accompanied by information extraction from the internal representation.

The whole Transformer starts with an *embedding* of the input set, expanding the dimension of the input data to the `embed_dim`, allowing the model to learn more complex representations. The embedding layer is just a FC Network layer with `embed_dim` neurons and multiple hidden layers, as shown in figure 4.9 (a). After which come the *Self-Attention* (SA) Blocks (section 4.5.3), whose core components are the *Multihead Self-Attention* (MHSA) (section 4.5.1) and *Feed Forward Network* (FFN) (section 4.5.2). The *class extraction* is done using the *Class Token* (section 4.5.4).

The Transformer architecture provides a way for **PFOs to communicate** and exchange valuable information, creating a complete representation of the jet (input data). Despite its complexity, the Transformer provides an immense advantage over the PFN and EFN , which is the ability to **learn long-range dependencies** in the input data.

4.5.1 Multihead Self-Attention

The input of the *Multihead Self-Attention* layer is a set of N vectors with dimension `embed_dim`, denoted as C, with overall batched shape $B \times N \times C$, where B is the batch size. Every input vector creates 3 'signs':

1. **Queries** Q representing questions of vectors asking other vectors in the set (for example: 'What is the highest p_{T}? in the set')

2. **Keys** K representing if the given vector has the answer to the questions of other vectors (for example: 'I have the highest p_{T}.')

3. **Values** V representing the answer to the questions of other vectors (for example: 'My p_{T} is 100 GeV.')

These three components are constructed from the input data. In the case of general Attention for Q, K, and V can be different (some other input asks the question, some additional input has the keys, and some other the values). Self-Attention is

a particular case of Attention where Q, K, and V are created from the same input, i.e., the input data is asking itself the questions. Going further in this section, we will only focus on SA .

The MHSA layer is shown in figure 4.8 (a) for visual aid. Q, K, and V are constructed from the same input data by the three `Linear` layers. After, the Q and V are matrix-multiplied across the embedding index (C), resulting in a matrix of shape $B \times N \times N$. This matrix is then scaled, and the softmax activation is applied to form a probability distribution for each vector over the set, stating how much *Attention* should be paid to each vector in the set to get the answer to the asked question. This can be expressed as

$$\texttt{Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}}\right) \tag{4.29}$$

resulting in a matrix of shape $B \times N \times N$. Multiplying the `Attention` by the V matrix extracts the answer, resulting in a matrix of shape $B \times N \times C$, the same shape as the input data.

Instead of doing the Attention mechanism across the whole embedding dimension, the Transformer uses multiple *heads*. Each head is a separate Attention mechanism with the head dimension $d = C/H$, where H is the number of heads. Each head's input and output shape is $B \times N \times C/H$, constructed by reshaping the input data to $B \times H \times N \times C/H$ (the calculations across the heads are done in parallel, hence the extra dimension). The heads are concatenated, resulting in an output of shape $B \times N \times C$. As a result, the argument of Attention must be scaled by $\sqrt{d}$, as stated in equation (4.29). At the end, a linear projection across the embedding dimension C is done, to allow the heads to exchange some information.

4.5.2 Feedforward Network

The *Feed Forward Network* (FFN) is a simple network with two `Linear` layers with `Activation` between them. Its primary purpose is to allow the individual vectors to learn from the data obtained in the MHSA layer. The size of the first layer is `embed_dim` $\cdot$ `expansion`, where `expansion` is a hyperparameter. The second layer is the `embed_dim` size to retain the original dimension of the input data. The dropout is applied to the output of the second layer.

4.5.3 Self-Attention Block

The *Self-Attention Block* (SA Block) has a MHSA part and a FFN part as seen in figure 4.8 (b). Before the MHSA and FFN part, the input data is normalized using

the *Layer Normalization* (LN) . LN computes the mean and standard deviation of the input data and normalizes it across the embedding dimension such that the mean is 0 and the standard deviation is 1. The mean and standard deviation are learned parameters, so the LN is a trainable layer.

The dropout layer is put after the MHSA preventing overfitting.

Across the MHSA and FFN part is the *residual connection*, which is a simple addition of the input data and the output of the MHSA and FFN part. This forces the model to learn new information on top of what is already in the input data.

The SA Block are stacked on each other, constructing L layers, which are the heart of the Transformer .

4.5.4 Class Extraction

The Transformer is a general-purpose model capable of learning any representation. To extract the class information of the whole input, we use the *Class Token* [12]. The *Class Token* is a vector of trainable parameters (of the same `embed_dim` shape as input vectors) concatenated at the beginning of the input data. It passes through the Transformer layers with the input vectors extracting and storing the class information. At the end of SA Blocks, the output is sliced (taking only the first vector, the *Class Token*) and passed through a `Linear` layer to get the final output followed by sigmoid or softmax Activation.

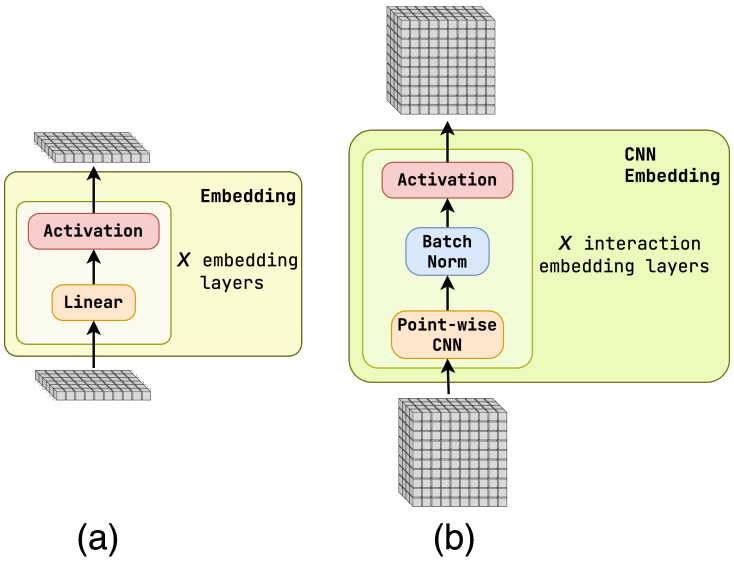


Figure 4.9 Embedding layers. (a) is a per-input vector embedding, and (b) is the embedding of interaction variables.

4.6 Particle Transformer

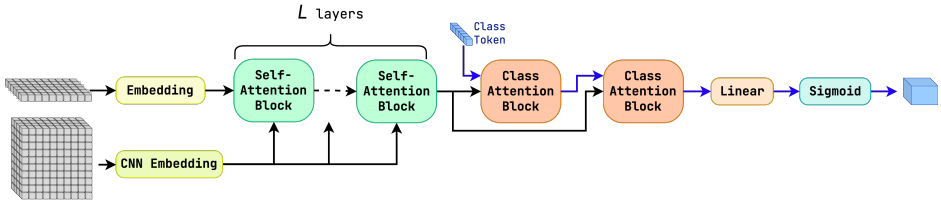


Figure 4.10 Particle Transformer architecture.

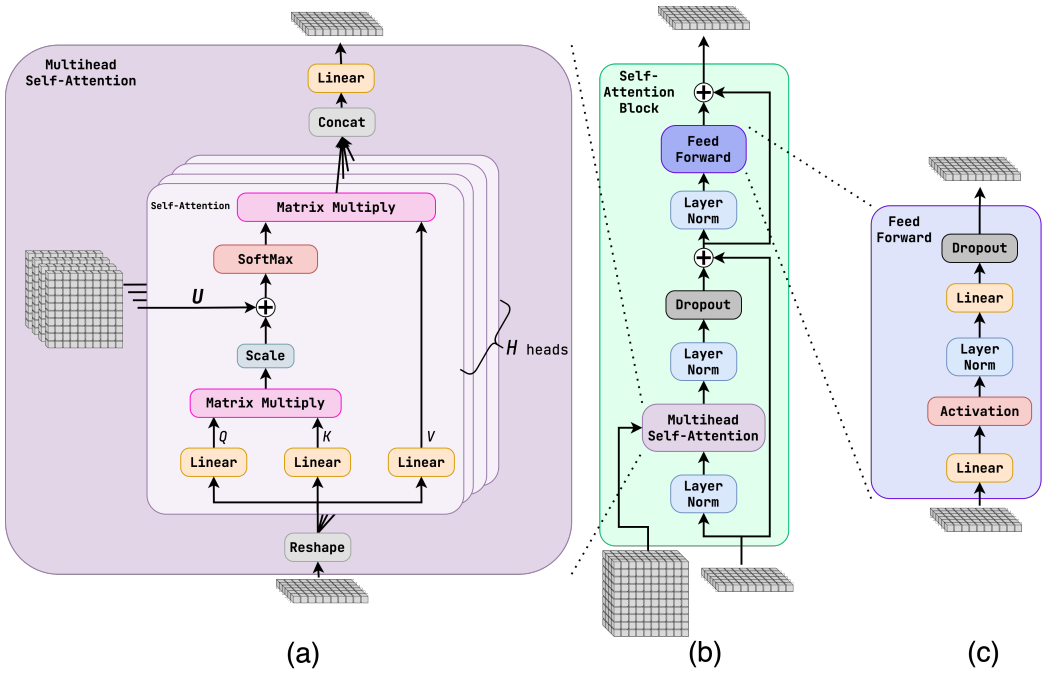


Figure 4.11 Self-Attention layers of Particle Transformer.

Particle Transformer [8] is a Transformer variant with particle physics in mind. It is based on the CaiT [22] architecture (see figure 4.10) with one crucial extension: *the interaction variables*. Interaction variables are given for each pair of input vectors, producing an overall second input of shape $B \times N \times N \times C_{int}$, where C_{int} is the number of interaction variables. These variables are manually engineered[7] and embedded using the *CNN embedding*, consisting of multiple blocks of point-wise CNN , batch normalization and Activation, as shown in

[7]The explicit form depends on the input type. In the case of PFOs, the variables are shown in section 5.3.2

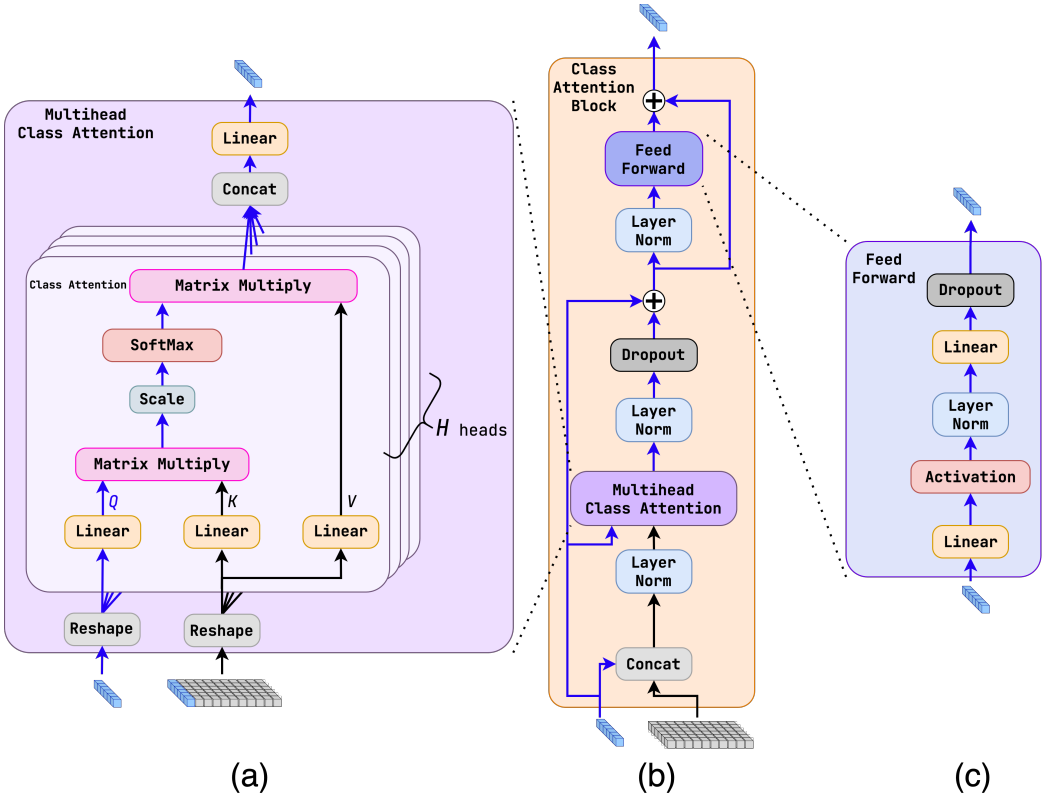


Figure 4.12 Class Attention layers of Particle Transformer.

figure 4.9 (b). Since the variables are symmetric, we only need to embed the upper triangle of the matrix, and the lower triangle is filled with the transposed upper triangle. The diagonal is filled with zeros.

Afterward, they are used as the second input *to every* SA Block, which is shown in figures 4.10 and 4.11, to modify the Attention as

$$\texttt{Attention} = \text{softmax}\left(\frac{QK^T}{\sqrt{d}} + U\right), \tag{4.30}$$

where U is embedded interaction variables. The point-wise CNN embedding of interaction variables is done into H (number of heads) dimensions, resulting in a matrix of shape $B \times N \times N \times H$ providing each head with its own interaction variables matrix U of shape $B \times N \times N$.

The only difference between SA Blocks in Transformer and ParT is the addition of two LN , right after the MHSA and right before the second `Linear` layer of the FFN . However, ParT utilizes the *Class-Attention* (CA) layers (section 4.6.1) for feature extraction, which is shown in figure 4.10 and more detailed in figure 4.12. It introduces the Class Token later in the architecture, allowing the MHSA to

focus purely on the input data. The output class is calculated from the Class Token as in Transformer .

Utilizing the interaction variables helps the model focus on the input data's essential parts. Each head of the MHSA focuses on a different part of the input data. The improvement of the ParT with the interaction variables compared to one without them is significant [8] (see section 5.5). On top of that, the CA blocks allow a more complex representation of PFOs and the jet information by separating the attention mechanisms.

4.6.1 Class Attention Block

Class Attention Block (CA Block) is a *Multihead Attention* (MHA) layer with the *Class Token as the query*. The input of the CA Block is a set of vectors of shape $B \times N \times C$ and the Class Token of shape $B \times C$. The Class Token is concatenated to this set of vectors, resulting in a tensor of shape $B \times (N + 1) \times C$, additionally it is used as a query for the MHA , as seen in figure 4.12. This allows the model to focus on the class information. The Class Token asks for it and stores it.

Carrying out the matrix multiplications of the MHA yields a tensor of the same shape as the Class Token, i.e., a vector of shape $B \times C$. This vector is then pushed through the FFN , outputting a vector of 'attended' Class Token. The residual connections use only the Class Token. The traces of the Class Token are displayed as blue arrows in figure 4.12.

Two CA Blocks are put after the SA Blocks, both having the same input of the set of vectors, and the output of the first CA Block is used as the Class Token input for the second CA Block. The second CA Block output is a vector of shape $B \times C$, with the extracted information about the class of the entire input, which is then passed through a `Linear` layer to get the final output followed by sigmoid or softmax Activation.

4.7 Dynamically Enhanced Particle Transformer

The *Dynamically Enhanced Particle Transformer* (DeParT) is an enhancement of the ParT architecture, utilizing the *Talking Multi-Head Attention* [13] mechanism (section 4.7.1). Other improvements include the use of *Stochastic Depth* (section 4.7.2), *Layer Scale* (section 4.7.3), and *gated* FFN (section 4.7.4), all visualized together in figure 4.13. The overall structure of the model is the same as in figure 4.10, but the SA Blocks are replaced with *Talking Self-Attention Blocks* (see section 4.7.1). The only modifications of CA Blocks are the replacements of classical FFN with gated FFN .

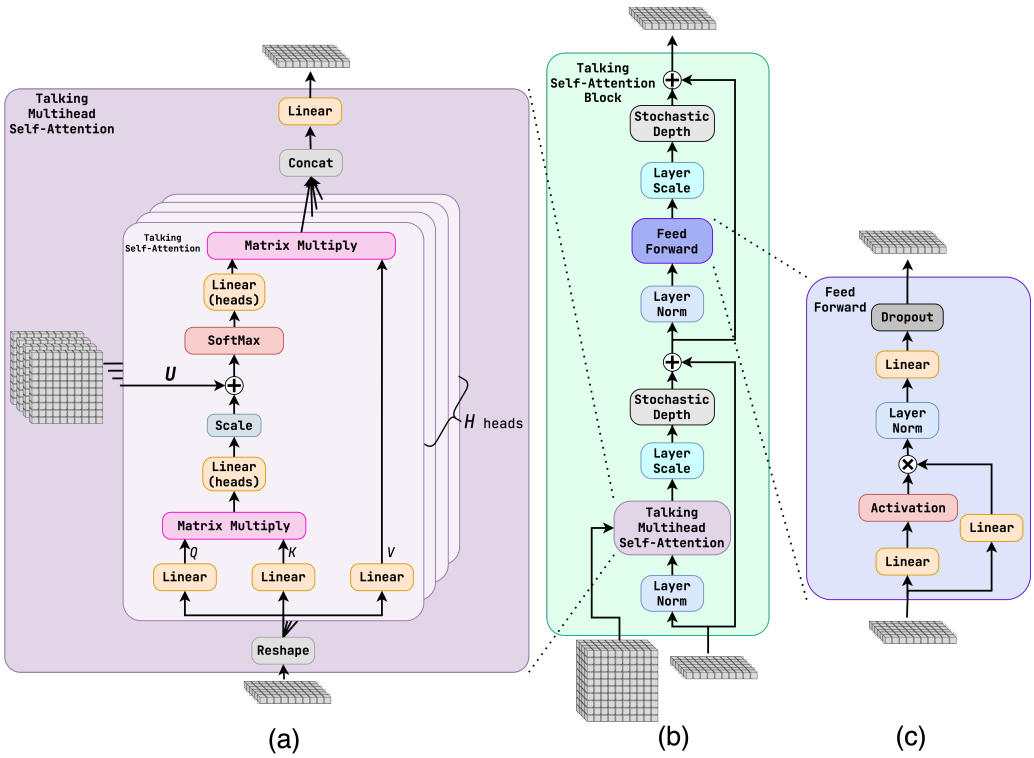


Figure 4.13 Talking Self-Attention layers of Dynamically Enhanced Particle Transformer.

The *Talking Multihead Self-Attention* utilizes the *interaction variables* even more by allowing the heads to exchange information about a given feature each head possesses. On top of that, the *Layer Scale* and *Stochastic Depth* allow us to train bigger models.

4.7.1 Talking Self-Attention Block

The *Talking Self-Attention Block* [13] is an extension of the MHA , allowing individual heads to *talk* to each other. This is done by doing a linear projection across the heads, in other words, applying a `Linear` layer across the heads dimension H. We do this twice in the MHA , right after the matrix multiplication of the query and key and right before the matrix multiplication of the value and the attention weights, as shown in figure 4.13.

4.7.2 Stochastic Depth

Stochastic Depth [69] is a type of dropout, whose *probability is dependent* on the layer number and drops the *entire layer* rather than individual elements of the output. The deeper the layer, the bigger the probability of dropping it, linearly dependent as

$$p_l = \frac{p_L}{2} \cdot \left(1 + \frac{l}{L}\right), \tag{4.31}$$

p_L is the probability of dropping the last layer, L is the total number of layers, and l is the current layer number. The first layer is indexed as $l = 0$.

In each Talking Self-Attention Block is the Stochastic Depth applied twice, after the MHSA and after the FFN .

4.7.3 Layer Scale

Layer Scale [22] is a type of normalization that helps the convergence of Transformers with many layers. Residual connections cause a bottleneck in the flow of gradients. This can be prevented by scaling the outputs of SA and FFN by an diagonal matrix $\mathrm{diag}\left(\lambda_{l,1}, \ldots, \lambda_{l,d}\right)$

$$\begin{aligned} x'_l &= x_l + \mathrm{diag}\left(\lambda_{l,1}, \ldots, \lambda_{l,d}\right) \times \mathrm{SA}\left(\mathrm{LN}\left(x_l\right)\right), \\ x_{l+1} &= x'_l + \mathrm{diag}\left(\lambda'_{l,1}, \ldots, \lambda'_{l,d}\right) \times \mathrm{FFN}\left(\mathrm{LN}\left(x'_l\right)\right), \end{aligned} \tag{4.32}$$

where x_l is the input of the layer, x'_l is the output of the SA layer, and x_{l+1} is the output of the FFN layer. The parameters $\lambda_{l,i}$ and $\lambda'_{l,i}$ are learned during the training, but at the beginning, are initialized with small values (e.g., 10^{-5}).

Layer Scale is applied after the SA and FFN layers in every SA Block.

4.7.4 Gated Feed-Forward Network

Gated Feed-Forward Network is a type of FFN , which uses a *Gated Linear Units* (GLU) [70]. Rather than being an improvement of the FFN , it is an improvement of the Activation after a linear layer. We can express the GLU in the form

$$\boldsymbol{o} = f(\boldsymbol{W}\boldsymbol{x}) \cdot \boldsymbol{W}_{\mathrm{g}}\boldsymbol{x}, \tag{4.33}$$

where $\boldsymbol{o}$ is the output, $\boldsymbol{x}$ is the input, $\boldsymbol{W}$ are the weights of the linear layer, $\boldsymbol{W}_g$ are the *weights of the gate* of the same shape as $\boldsymbol{W}$, and f is the activation function. We apply the ReLU activation in all gated FFN since it is the best performing [70]. There is also no bias in the GLU , which makes the model perform better.

By introducing the second weight matrix $\boldsymbol{W}_g$, the overall size of the FFN would be bigger. To have the same size as the classical FFN , the layer expansion is multiplied by the factor of $2/3$.

Chapter 5

Training Jet Taggers

5.1 Main Goal

In our study, we teach a model to predict the probability of a jet being a quark or a gluon. In other words, we want to *tag* the jet with a label saying whether it came from a quark or a gluon. The dataset containing the jet information and the targets is the MC simulations described in section 3.2. The target is a variable constructed from `jets_PartonTruthLabelID`, which can have nine values: u-quark (1), d-quark (2), s-quark (3), c-quark (4), b-quark (5), t-quark (6), gluon (21), pileup (-1) or unknown (-999). The numbers in brackets are the *PDG ID* (PID) of the parton [25].

Quarks (PID = 1, 2, 3, 4, 5, 6) are put into a single class corresponding to the number 1, while gluons (PID = 21) are put into a single class corresponding to the number 0. We refer to these targets as *Truth Labels*. Jets with `jets_PartonTruthLabelID` equal to -1 or -999 are omitted from the dataset.

Before we can train a model, we need to make some assumptions about the data. Primarily, the *jet tag cannot depend on the underlying event*. In other words, we must be able to predict the jet tag using only the jet information, i.e., *jet variables*. We will not use any physical weights coming from the MC simulation.

Previous ATLAS work primarily used only one variable to tag the jet [9]. Some improvement has been made by utilizing more variables and training a BDT [11], basic Machine Learning algorithm [10]. No other models have been successfully deployed. We will train *all models* introduced in chapter 4 on the same dataset and compare the results. Our baseline model is the BDT , which we will retrain with the same dataset as the other models and the same hyperparameters as in [11]. One variable tagger is not included in the comparison, as it is inferior to the BDT [11].

5.2 Dataset

The dataset is constructed from the MC simulations described in section 3.2. We use the `Pythia` dijet MC simulation with anti-k_t jet clustering with $R = 0.4$ (see section 3.5). The event reconstruction is done with `Athena release 21`[1].

The preprocessing of the data is done in two stages: offline (section 5.2.1) and online (section 5.2.2). The offline preprocessing is done once, while the online preprocessing is done during training.

5.2.1 Offline Preprocessing

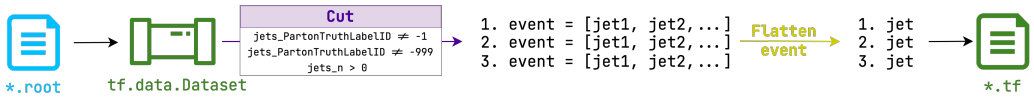


Figure 5.1 The offline preprocessing of the data.

The data comes in the form of a `ROOT` [71] file with a `TTree` containing the variables of interest. We use the `uproot` [72] package to convert each file to `Tensorflow` [73] native object `tf.data.Dataset`[2], which is afterward saved to a file. We utilize the proprietary format of `Tensorflow` datasets to reduce the time needed to load the data. The `ROOT` format is not optimized for fast sequential access.

After the first conversion, the data is preprocessed again. In this second stage, we remove pileup and unknown jets (those with PID $= -1,\ -999$), filter events without any jet, and *flatten events* into individual jets. The `tf.data.Dataset` is then efficiently saved to shards of equal size, enabling parallel loading of the data. The whole process is illustrated in figure 5.1.

5.2.2 Online Preprocessing

The online preprocessing utilizing the `tf.data.Dataset` API allows a pipeline construction of the dataset. Our preprocessing pipeline consists of the following steps:

1. Load each JZ slice separately.
2. Filter data with the following cuts:
    - `jets_passFJVT` $== 1$,

[1] `https://atlassoftwaredocs.web.cern.ch/athena/athena-releases/`
[2] `https://www.tensorflow.org/api_docs/python/tf/data/Dataset`

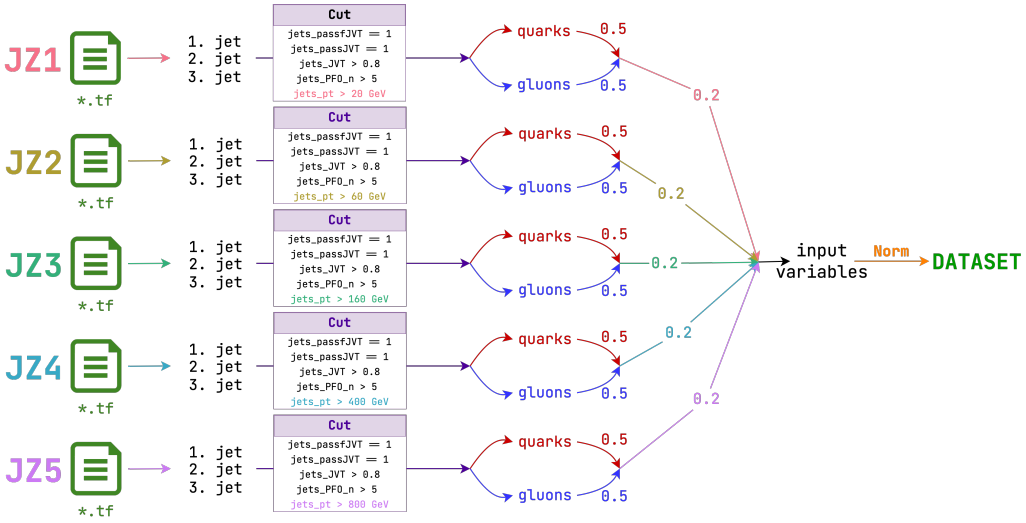


Figure 5.2 The online preprocessing of the data.

- `jets_passJVT` $== 1$,
- `jets_Jvt` > 0.8,
- `jets_PFO_n` > 5,
- `jets_pt` $>$ *(based on JZ slice, see section 5.2.3)*,

3. For each JZ slice, resample the quark and gluon datasets, each with equal probability 0.5,

4. Combine JZ slices, each with the same probability (0.2 for each JZ slice when JZ slices 1-5 are used),

5. Construct desired input variables,

6. Normalize the input variables,

In step 2, we apply cut on the jet variables estimating the amount of pileup. ATLAS MC samples do not come with truth information about pileup jets to reduce the amount of used disk space, so we filter them out as much as possible using the abovementioned cuts.

We apply a cut on the jet p_{T} based on the JZ slice to make the spectrum for quarks and gluons more similar. This is extensively discussed in section 5.2.3.

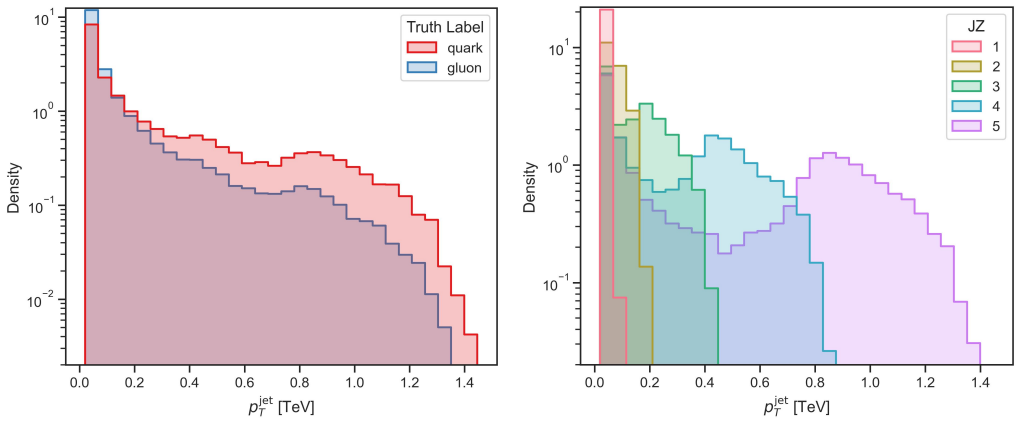
Resampling the quark and gluon datasets is done to make the dataset balanced. We can obtain the same result by reweighting each jet, but we have no prior knowl-

edge of the weights. The method `tf.data.dataset.rejection_resample`[3] allows resampling on the go by computing the ratio dynamically as the data is processed.

Combining JZ slices with equal probability ensures the model works on the full spectrum of jets and *does not use* p_{T} to make the decision. This is a non-trivial step, as in actual physical data, the jet p_{T} spectrum is highly dominated by low p_{T} jets, while the high p_{T} jets are rare. However, some applications are specifically interested in the high p_{T} jets, so we want to ensure the model can also handle them.

The construction of variables and their normalization is explained section 5.3. The whole offline preprocessing is illustrated in figure 5.2.

5.2.3 JZ Cuts



(a) colored by their Truth Label

(b) colored by their JZ slice

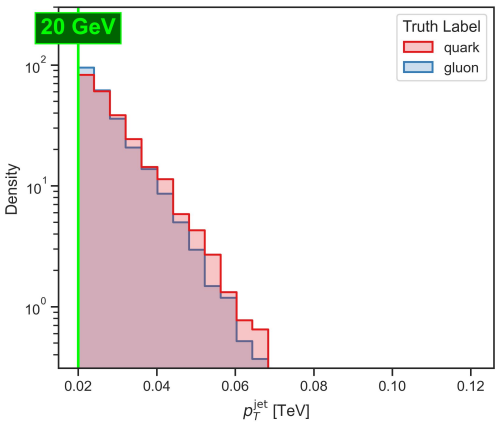
Figure 5.3 Normalized p_{T} distributions of jets before the JZ cuts are applied.

As stated in section 3.2, data is split into JZ slices, each having a different p_{T} distribution. The distribution of the jet p_{T} is shown in figure 5.3. The number of quarks and gluons is the same.

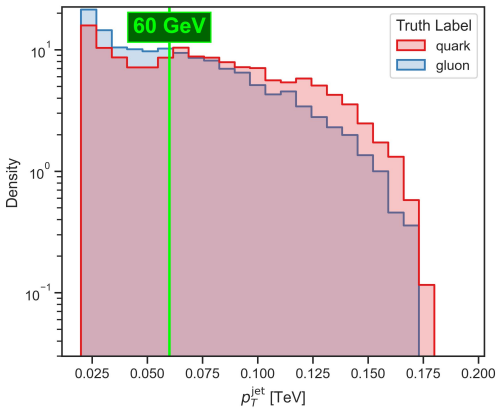
From figure 5.3a, we can see that the p_{T} distribution of the jets is different for each Truth Label. Gluons dominate the low p_{T} region[4], while quarks dominate the high p_{T} region. All JZ slices contribute to these low p_{T} regions, mainly with soft gluon jets. This can be seen from figure 5.4, where all JZ slices are shown separately. We do not want the tagger to use the variable $p_{\mathrm{T}}^{\mathrm{jet}}$ to distinguish

[3] `https://www.tensorflow.org/api_docs/python/tf/data/Dataset#rejection_resample`

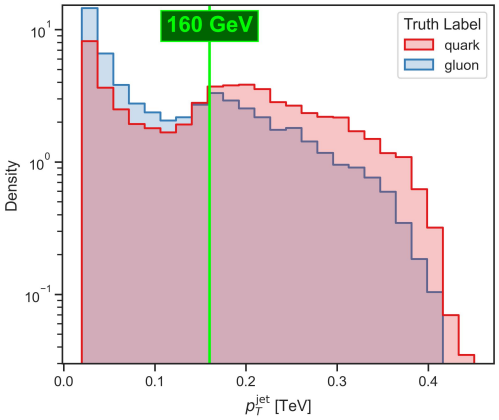
[4] This might not be obvious, but pay attention to the log scale of the y-axis.



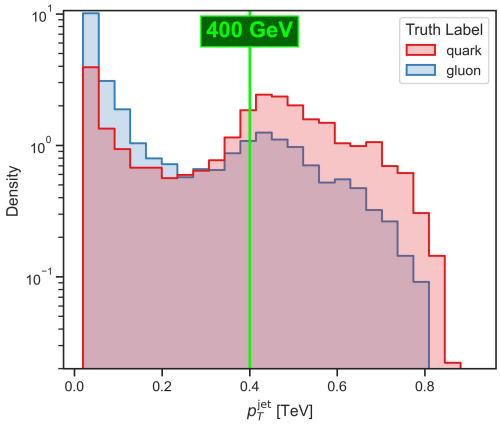
(a) JZ slice 1, 20 GeV cut



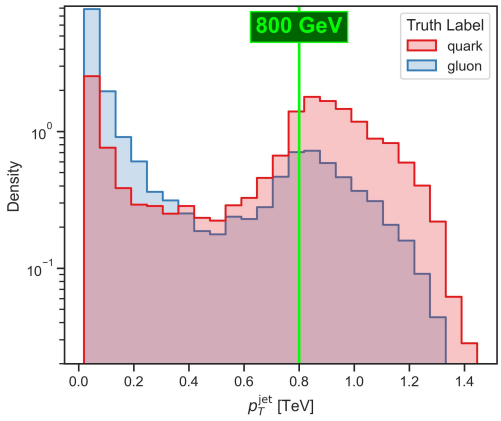
(b) JZ slice 2, 60 GeV cut



(c) JZ slice 3, 160 GeV cut

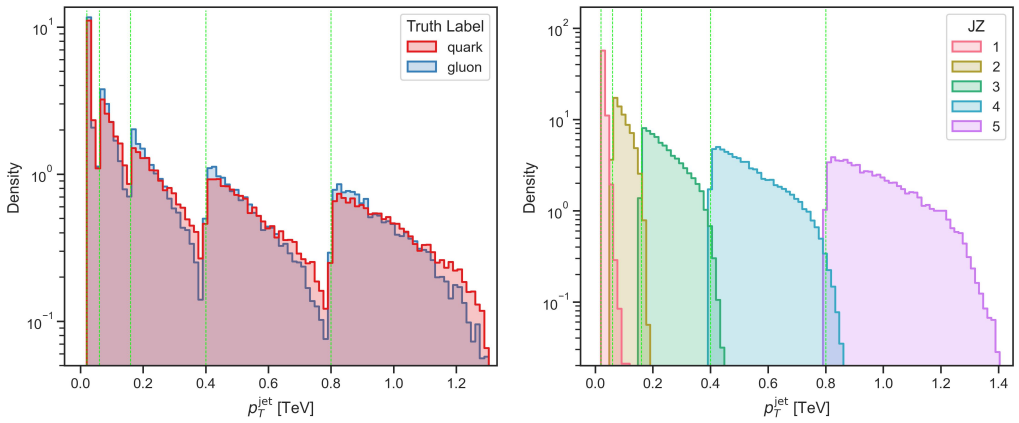


(d) JZ slice 4, 400 GeV cut



(e) JZ slice 5, 800 GeV cut

Figure 5.4 Normalized p_{T} distributions of jets before the JZ cuts are applied.



(a) colored by their Truth Label **(b)** colored by their JZ slice

Figure 5.5 Normalized p_T distributions of jets after the JZ cuts are applied. Lime lines correspond to the p_T cuts.

between quarks and gluons and avoid jets created from soft gluons. We separately apply a cut on the jet p_T for each JZ slice. These cuts are visualized in figure 5.4 with their respective values. Every jet with lower p_T than the cut is removed from the dataset.

The p_T distributions of the jets after the cuts are applied are shown in figure 5.5. Contributions from individual JZ slices are also visible in figure 5.5a, forming a 'comb-like' distribution shape. The respective cuts are shown with lime lines to make it easier to see the cuts. The soft gluons are suppressed, and the p_T distribution of the jets is more similar for each Truth Label.

5.3 Input Variables

We will construct four types of variables:

- **PFO variables** - variables that are construsted from the 4-momenta of the PFOs , they are calculated for *each PFO* in a given jet.
- **PFO interaction variables** - variables that are calculated from a pair of 4-momenta of two PFOs , they are calculated *for each pair of PFOs* in a given jet.
- **BDT variables** - variables that are calculated from PFOs for a whole jet.
- **High-level jet variables** - variables that are pre-calculated for the jet in the `Athena` framework.

The distributions of these variables are shown in appendix B.

5.3.1 PFO Variables

Table 5.1 PFO variables used as an input to constituent-based models. E is the energy of the PFO, p_{T} is the transverse momentum of the PFO, and η and ϕ are the pseudorapidity and azimuthal angle of the PFO, respectively. [8]

PFO Variables
$\Delta\eta = \eta - \eta^{\mathrm{jet}}$
$\Delta\phi = \phi - \phi^{\mathrm{jet}}$
$\Delta R = \sqrt{\Delta\eta^2 + \Delta\phi^2}$
$\log p_{\mathrm{T}}$
$\log E$
$\log \frac{p_{\mathrm{T}}}{p_{\mathrm{T}}^{\mathrm{jet}}}$
$\log \frac{E}{E^{\mathrm{jet}}}$
m

The complete list of PFO variables is shown in table 5.1 [8]. They are calculated for each PFO in a given jet. An input is then a vector of PFO variables for each PFO in a given jet, forming an input shape of $N \times 8$, where N is the number of PFOs in a jet.

We want the tagger to be rotationally invariant, so we only use angles information wrt. the jet axis. These are the first three variables in table 5.1. The dynamical variables include the p_{T} and E of the PFO . Long-tailed distributions may cause problems for the neural network, so the logarithms of p_{T} and E are better suited. To emphasize the importance of a given PFO in a jet, the logarithm of the ratios of p_{T} (E) of a PFO to a jet $p_{\mathrm{T}}^{\mathrm{jet}}$ (E^{jet}) are used. Most of the PFOs are neutral particles or cannot be associated with massive particles, so we assume they are relativistic and neglect their masses. However, some leave tracks and are reconstructed as charged pions with non-zero masses. We utilize this information by adding the mass as a variable, which is not used in [8].

These variables are inputted to Transformer , ParT , and DeParT . PFN and EFN also use these variables, apart from the mass m.

5.3.2 PFO Interaction Variables

The complete list of PFO interaction variables is shown in table 5.2 [8]. They are calculated for each pair of PFOs in a given jet. An input is a matrix of PFO interaction variables for each pair of PFOs in a given jet, forming a shape of $N \times N \times 4$, where N is the number of PFOs in a jet. The matrix is symmetric, and the diagonal is manually filled with zeros.

Table 5.2 Interaction PFO variables used as an input to interacting constituent-based models. E^a and E^b are the energies of the two PFOs, $p^{\mu,a}$ and $p^{\mu,b}$ are the 4-momenta of the two PFOs, η^a and η^b are the pseudorapidities of the two PFOs, ϕ^a and ϕ^b are the azimuthal angles of the two PFOs, and p_{T}^a and p_{T}^b are the transverse momenta of the two PFOs. [8]

PFO Interaction Variables
$\log \Delta = \log \sqrt{(\eta^a - \eta^b)^2 + (\phi^a - \phi^b)^2}$
$\log k_{\mathrm{T}} = \log \left(\min \left(p_{\mathrm{T}}^a, p_{\mathrm{T}}^b\right)\Delta\right)$
$z = \min \left(p_{\mathrm{T}}^a, p_{\mathrm{T}}^b\right)/(p_{\mathrm{T}}^a + p_{\mathrm{T}}^b)$
$\log m^2 = \log \left(p^{\mu,a} + p^{\mu,b}\right)^2$

The kinematic variable is the angular distance Δ. The invariant mass m^2 is calculated from the 4-momenta of the two PFOs . We utilize the same variable k_{T} as used in the jet reconstruction section 3.5 (however, it is labeled differently, according to [8]). The last variable, z, is also dynamic, the fraction of p_{T} carried by less energetic PFO . Again we take the *logarithm of all these variables* (except z because it is bounded, see the distribution in appendix B.2) to squish the distributions.

These variables are a secondary input to ParT and DeParT . We will train these models both with and without interaction variables. Those trained with interaction variables are denoted as *Interacting ParT* and *Interacting DeParT*.

5.3.3 BDT Variables

The complete list of BDT variables is shown in table 5.3 [11][5]. They are calculated for the whole jet from all PFOs , forming a 5-dimensional vector as an input.

N_{PFO} is the number of PFOs in a jet, i.e. the *multiplicity*. Before BDT , taggers used this single variable to tag jets. Variables W_{PFO} and $C_1^{\beta=0.2}$ are carefully engineered to achieve the best performance. The $p_{\mathrm{T}}^{\mathrm{jet}}$ and η^{jet} were only added in the last version of the tagger [11].

Only BDT uses these variables as input.

5.3.4 High-level Jet Variables

The complete list of high-level jet variables is shown in table 5.4, and their definitions and commonly used symbols are given in appendix A. These variables are precomputed by the `Athena`[6] framework and are available for each jet. Together

[5] In the original paper [11], they do not use PFOs , but only track (trk) information.

[6] `https://atlassoftwaredocs.web.cern.ch/athena/athena-releases/`

Table 5.3 BDT variables used as an input to the Boosted Decision Tree. p_{T} is the transverse momentum of the PFO, η is the pseudorapidity of the PFO, ϕ is the azimuthal angle of the PFO, η^{jet} is the pseudorapidity of the jet, and ϕ^{jet} is the azimuthal angle of the jet, $p_{\mathrm{T}}^{\mathrm{jet}}$ is the transverse momentum of the jet. Summations are over PFOs in the jet. [11]

BDT Variables
$p_{\mathrm{T}}^{\mathrm{jet}}$
η^{jet}
$N_{\mathrm{PFO}} = \sum_{\mathrm{PFO} \in \mathrm{jet}}$
$W_{\mathrm{PFO}} = \frac{\sum_{a \in \mathrm{jet}} p_{\mathrm{T}}^{a} \sqrt{(\eta^{a}-\eta^{\mathrm{jet}})^2+(\phi^{a}-\phi^{\mathrm{jet}})^2}}{\sum_{a \in \mathrm{jet}} p_{\mathrm{T}}^{a}}$
$C_1^{\beta=0.2} = \frac{\sum_{a,b \in \mathrm{jet}}^{a \neq b} p_{\mathrm{T}}^{a} p_{\mathrm{T}}^{b} \left(\sqrt{(\eta^{a}-\eta^{b})^2+(\phi^{a}-\phi^{b})^2}\right)^{\beta=0.2}}{\left(\sum_{a \in \mathrm{jet}} p_{\mathrm{T}}^{a}\right)^2}$

Table 5.4 High-level jet variables commonly used in ATLAS analyses. They come precomputed by `Athena` in the jet container. Index 0 refers to the first entry of that variable in the jet container.

High-level Jet Variables	
`ActiveArea4vec_(m,pt,eta,phi)`	`fJVT`
`JetConstitScaleMomentum_(m,pt,eta,phi)`	`passFJVT`
`averageInteractionsPerCrossing[0]`	`Jvt`
`FracSamplingMax`	`JvtRpt`
`FracSamplingMaxIndex`	`passJVT`
`GhostMuonSegmentCount`	`JVFCorr`
`ChargedPFOWidthPt1000[0]`	`Timing`
`TrackWidthPt1000[0]`	`EMFrac`
`NumChargedPFOPt1000[0]`	`Width`
`NumChargedPFOPt500[0]`	`chf`
`SumPtChargedPFOPt500[0]`	`PFO_n`
`(m,pt,eta,phi)`	

they form a 30-dimensional vector as an input.

FC Network and Highway Network use these variables as input.

5.3.5 Normalization

To make the learning process easier, we normalize the input variables. This helps the neural network to converge faster and avoids numerical problems. Only BDT variables are not normalized since BDT performance is not affected by the normalization.

We can sketch the normalization procedure as follows:

$$\xi \to \tilde{\xi} = \frac{\xi - \mu}{\sigma}, \tag{5.1}$$

where ξ is the input variable, μ is the mean of the variable, and σ is the standard deviation of the variable. Each variable is normalized separately, shifting the mean to zero and squishing the distribution to unit variance.

The mean and standard deviation are precomputed on a subset of the data before training. We utilize the `tf.keras.layers.Normalization`[7] with `axis=-1`, and the method `adapt` to compute the mean and standard deviation. Means and standard deviations are fixed, *untrainable parameters*.

5.4 Training Configuration

The training is performed with `TensorFlow 2.9.1` [73] and `Keras 2.9.0` [74]. The general training configuration is the same for all models, with some exceptions for BDT . The training dataset consists of **5M jets**. The validation and test datasets both consist of 0.5M jets. Each model is trained for **12 epochs** (except BDT , which has no epochs) with a **batch size of 512** (resulting in 118K steps). Validation is performed after each epoch, calculating the AUC and Accuracy. We employ the **LAMB optimizer** (with $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\varepsilon = 10^{-6}$) with an initial learning rate of 0.001, **cosine-decaying** to zero over the 12 epochs. No clipnorm nor weight decay is used, but a **linear warmup** is applied for the first 100 steps of the training. We utilize the **binary cross-entropy** loss function without label smoothing. We use 500 batches to calculate the mean and standard deviation for the normalization layer. Hyperparameters are summarized in table 5.5. There was no extensive hyperparameter tuning. They were chosen based on related works [13, 8, 22, 12].

All models have approximately **2.6M parameters**, except BDT . There were trained on two `NVIDIA TESLA V100` GPUs with 16GB or 32GB of memory (depending on the model, interaction variables are more memory-hungry).

[7] `https://www.tensorflow.org/api_docs/python/tf/keras/layers/Normalization`

Table 5.5 Summary of common training configuration for all models.

Data	
Training Dataset	5M jets
Validation Dataset	0.5M jets
Test Dataset	0.5M jets
Number of Epochs	12
Batch Size	512
Total Steps	118k
Normalization Layer	adaptation on 500 batches
Optimizer	
Optimizer	LAMB
Learning Rate	0.001
β_1	0.9
β_2	0.999
ε	10^{-6}
Learning Rate Scheduling	Cosine Decay
Warmup	Linear (100 steps)
Clip Norm	None
Loss	
Loss	Binary Cross Entropy
Label Smoothing	None
Weight Decay	None

5.4.1 BDT Configuration

The training is performed with the `TensorFlow Decision Forests 0.2.7`[8] library using the `tfdf.keras.GradientBoostedTreesModel` model. We train 600 trees with a maximum depth of 5. The minimum number of samples in a leaf is 5000. The shrinkage factor is 0.5, and L2 regularization is applied with a weight of 0.1. The loss is the same as for the neural networks, the binary cross-entropy. The `BEST_FIRST_GLOBAL` growing policy is used, and the split axis policy is `SPARSE_OBLIQUE`. All additional parameters are set to their default values[9]. Everything is summarized in table 5.6.

[8] `https://www.tensorflow.org/decision_forests`

[9] `https://www.tensorflow.org/decision_forests/api_docs/python/tfdf/keras/GradientBoostedTreesModel`

Table 5.6 Configuration of BDT model.

Parameter	BDT
Number of Trees	600
Growing Strategy	`BEST_FIRST_GLOBAL`
Maximum Depth	5
Split Axis	`SPARSE_OBLIQUE`
Shrinkage	0.5
Minimum Examples per Leaf	5k
L2 Regularization	0.1

5.4.2 Transformer, ParT, and DeParT Configuration

Table 5.7 Configuration of Transformer, ParT, and DeParT models.

Parameter	Transformer	ParT	DeParT
Embedding Dimension	128	128	128
Self-Attention Block Layers	13	11	11
Class Attention Block Layers	-	2	2
Heads	8	8	8
Expansion	4	4	4
Dropout	-	0.1	-
Stochastic Depth Drop Rate	-	-	0.1
Layer Scale Initialization	-	-	10^{-5}
Number of Embedding Layers	3	3	3
Number of Interaction Embedding Layers	-	3	3
Size of Interaction Embedding Layers	-	64	64
Activation	GELU	GELU	GELU

Transformer, ParT , and DeParT embed the input variables into 128 dimensions, with 3 embedding layers. The total number of layers of each model is 13, where in the case of DeParT and ParT , the last 2 layers are CA Blocks. All three models use 8 heads and FFN expansion factor of 4, which results in 512 dimensions in the FFN layer and GELU Activation. In the case of the Transformer , no dropout is applied, ParT uses dropout with a rate of 0.1, and DeParT uses Stochastic Depth Drop with a rate of 0.1. DeParT also uses the Layer Scale with an initial value of 10^{-5}.

ParT and DeParT are trained with and without the interaction variables. The interaction variables are embedded with 3 layers, each of size 64, while the last

one is the same as the number of heads, 8. All hyperparameters are summarized in table 5.7.

5.4.3 Fully Connected and Highway Network Configuration

Table 5.8 Configuration of FC and Highway models.

Parameter	Fully Connected	Highway
Layer Size	512	344
Number of Layers	11	11
Dropout	0.2	0.2
Activation	Swish	Swish

Both FC Network and Highway Network have 11 hidden layers with a 0.2 dropout rate and Swish Activation. FC Network has 512 neurons in each layer, while Highway Network has 344 neurons to account for the skip connections.

5.4.4 PFN and EFN Configuration

Table 5.9 Configuration of PFN and EFN models.

Parameter	PFN	EFN
Φ layers	5 x 512	4 x 512 + 1 x 136
F layers	6 x 512	6 x 512
Φ	CNN	CNN
Activation	ReLU	ReLU
Dropout	-	-

Both PFN and EFN use point-wise CNN for the Φ mapping. PFN has 5 layers of sizes 512, while EFN has 4 layers of sizes 512 and one of 136 to account for the summation over the PFOs while still having roughly the same number of parameters as PFN . For the F function, both models use 6 layers of sizes 512. All Activation functions are ReLU, and no dropout is applied.

5.5 Results

The training results are shown in table 5.10. Our proposed DeParT model with interaction variables outperforms all other models. It also outperforms all other

Table 5.10 Results of the different models. The best results are highlighted in bold.

Model	Accuracy	AUC	ε_q	ε_g
Interacting DeParT	**0.7433**	**0.8227**	**0.7204**	0.7680
Interacting ParT	0.7427	0.8223	0.7142	0.7730
DeParT	0.7414	0.8203	0.7163	0.7669
ParT	0.7404	0.8194	0.7092	**0.7777**
Transformer	0.7384	0.8170	0.7138	0.7634
PFN	0.7366	0.8150	0.7101	0.7661
EFN	0.7201	0.7966	0.6890	0.7555
Highway	0.7316	0.8089	0.7017	0.7617
Fully Connected	0.7306	0.8100	0.6998	0.7620
BDT	0.7073	0.7823	0.6721	0.7426

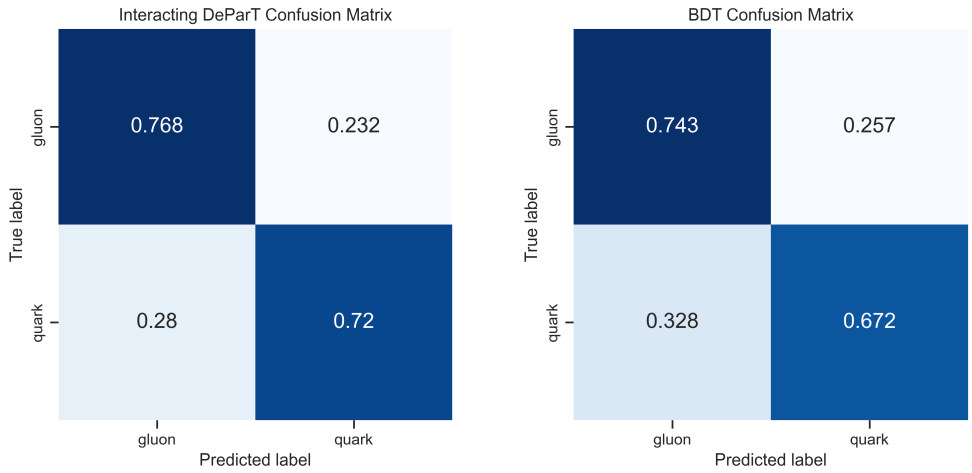
Table 5.11 Technical information about the models. The total number of parameters is given in millions. The time and memory are measured on a single NVIDIA Tesla V100 GPU 32GB. The time corresponds to the inference time of one batch of size 512. Memory is the maximum GPU memory usage during inference of batch with size 512.

Model	Params [10^6]	Batch Time [ms]	GPU Memory [MB]
Interacting DeParT	2.62	190.3	2578
Interacting ParT	2.62	169.7	1411
DeParT	2.61	118.0	574
ParT	2.61	95.8	536
Transformer	2.61	110.0	570
PFN	2.64	15.6	359
EFN	2.60	15.0	551
Highway	2.62	2.8	44
Fully Connected	2.64	1.6	34
BDT	-	20.4	-

models without interaction variables. What is fascinating is that the previously used model on jet tagging, BDT , gets obliterated by *all* other models. However, the differences between the models are not as significant as expected. This is due to the physical limit of the tagging task at low p_{T}. We will discuss this in more detail in section 5.5.1.

The technical information about the models is summarized in table 5.11. The disk space of saved models is roughly the same for all of them, around 35 MB, except for BDT , which is 1.4 MB. It is given by the number of total parameters,

which is roughly the same for all models. The tiny differences did not affect the performance.



(a) Interacting DeParT **(b)** BDT

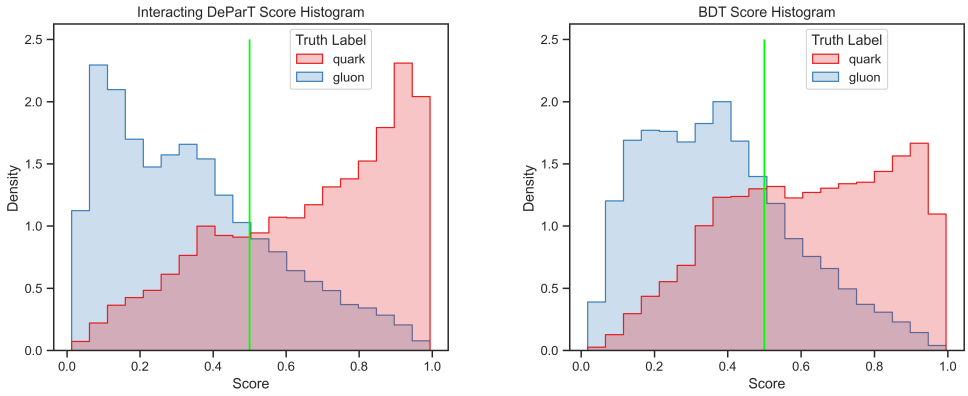
Figure 5.6 Confusion matrices for selected models.

Even though the interaction variables are computed from the PFO variables, i.e., the model can learn how to calculate them without being explicitly told, they still improve the performance. Because every head is given different input, it helps them focus on various aspects of the jet, and when combined, they can learn more complex patterns. Even more, allowing the heads to talk to each other (something that ParT does not, but DeParT does) improves the performance. On the other hand, interaction variables make the computation more expensive because they are used as a second input, the memory usage is much higher, and the inference time is longer, as seen in table 5.11.

Another exciting aspect is the inference time of the models, BDT especially. Since BDT dost not run on the GPU, it is significantly slower than Highway Network and FC Network , which are far more complex models. On top of that, even EFN and PFN that use PFO variables are faster than BDT .

Confusion Matrices are shown in figure 5.6 for selected models (the rest is in appendix C.1). They follow a similar pattern, where the quarks are more challenging to distinguish from the gluons than the gluons from the quarks. This can be seen from the lower left corner, where the quarks are misclassified as gluons.

Score outputs colored by their Truth Label of the selected models are shown in figure 5.7 (the rest is in appendix C.2). The middle lime line, at 0.5, is the threshold for the classification, everything left is classified as a gluon, and everything right



(a) Interacting DeParT **(b)** BDT

Figure 5.7 Output scores for selected models colored by the Truth Label. The lime line represents the classification threshold; everything left is tagged as a gluon, and everything right is tagged as a quark.

is classified as a quark. The DeParT model on figure 5.7a has better separation between the two classes than the BDT . However, all models have a small peak at 0.4, where they all suffer from misclassification. This peak comes from low p_{T} jets, where the tagging is limited, as discussed in section 5.5.1.

We have demonstrated the unsatisfactory performance of the previously used model, BDT , and the superiority of our proposed model, DeParT . In the following sections, we will look at the dependence of the performance on the p_{T}, η, and the pileup μ. We will omit the BDT model from the plots, as it is irrelevant and makes the plots unreadable.

5.5.1 Transverse Momentum Dependence

In figure 5.8, we can see that the accuracy of the models is dependent on the p_{T} of the jets. The higher the p_{T}, the better the performance. At low p_{T}, there seems to be an upper bound on the accuracy since none of the models can improve as much as they do at higher p_{T}. Different factors may cause this. The pure definition of `PartonTruthLabelID` can be inconsistent at low p_{T}, or the jet clustering algorithm has problems such as distinguishing hard jets and pileup. Also, at higher p_{T}, above 200 GeV, the performance is more or less stable with p_{T}.

The model's ordering is more visible if we zoom in on the high p_{T} range, as seen in figure 5.9. The EFN is the worst-performing model, which indicates that some kind of mapping of energy variables (like p_{T} or E) is necessary for the model to perform well and instead tackle the IRC safety differently. The Highway Network is almost identical to the FC Network , which is unsurprising,

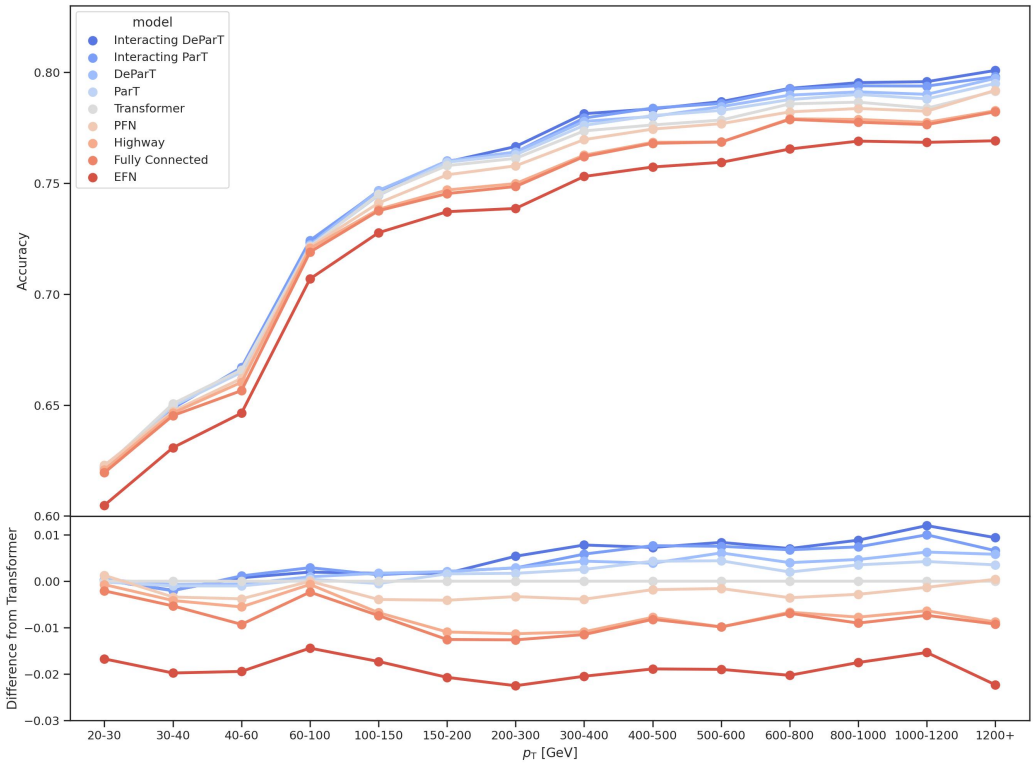


Figure 5.8 Accuracy of the models as a function of p_T and relative to the Transformer model.

as they are very similar models. The main advantage is that the Highway Network diverges less than the FC Network , but using the LAMB optimizer makes the FC Network converge to the same result[10].

The averaged difference from Transformer is displayed in figure 5.10 to emphasize the relative model performances.

Additional metrics are plotted in appendix C.3.

[10]When we trained them with Adam optimizer, the FC Network performed worse.

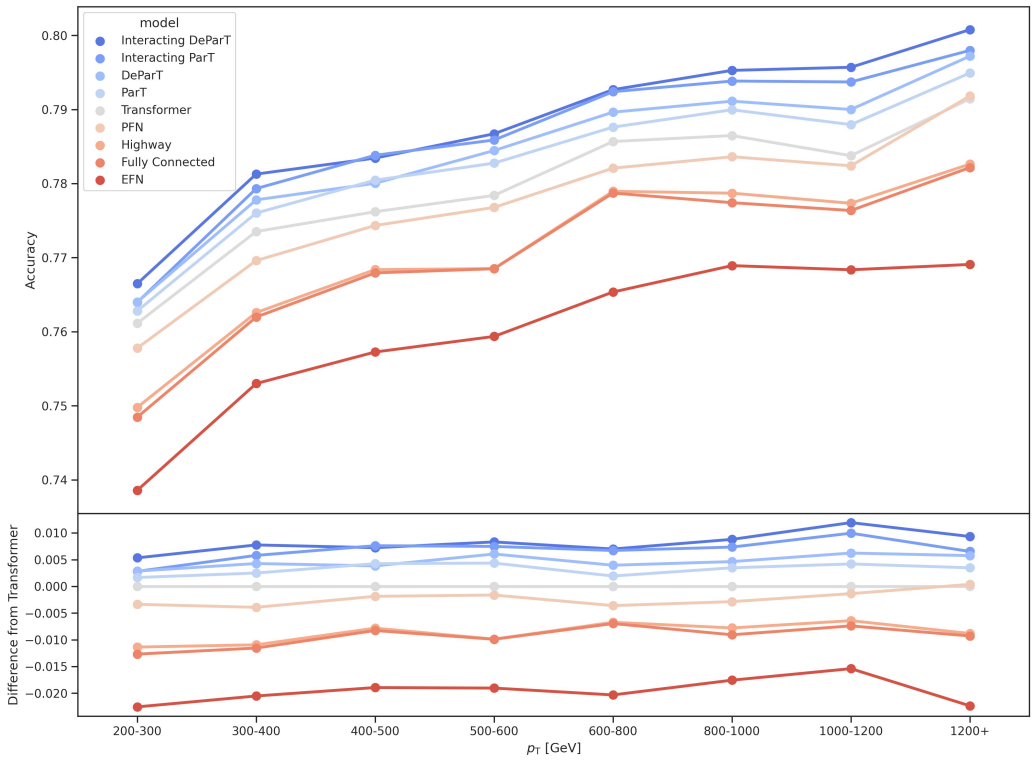


Figure 5.9 Zoom to the accuracy of the models as a function of p_T and relative to the Transformer model.

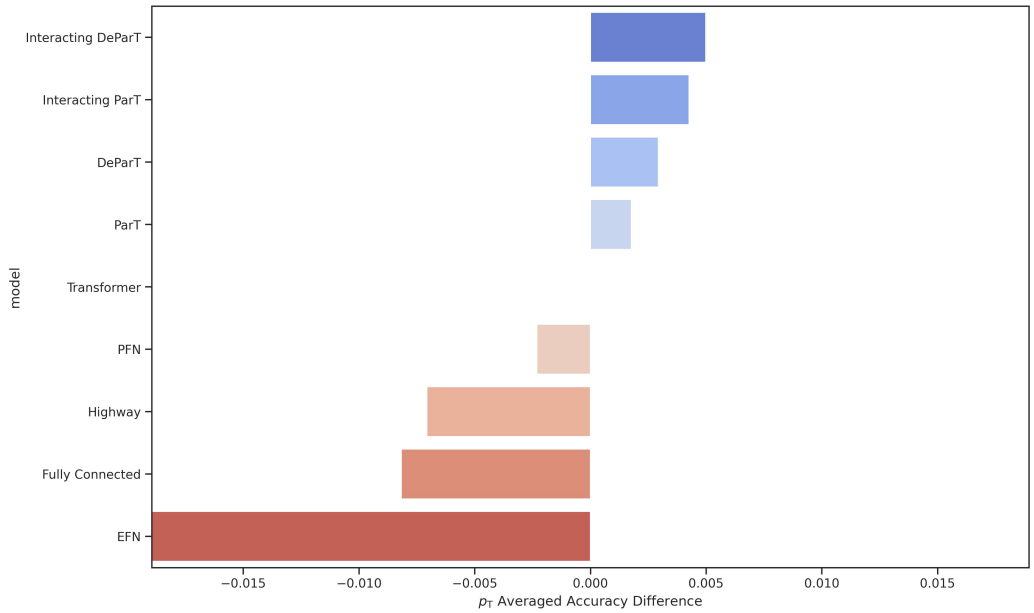


Figure 5.10 p_T averaged difference from the Transformer model.

5.5.2 Pseudo-rapidity Dependence

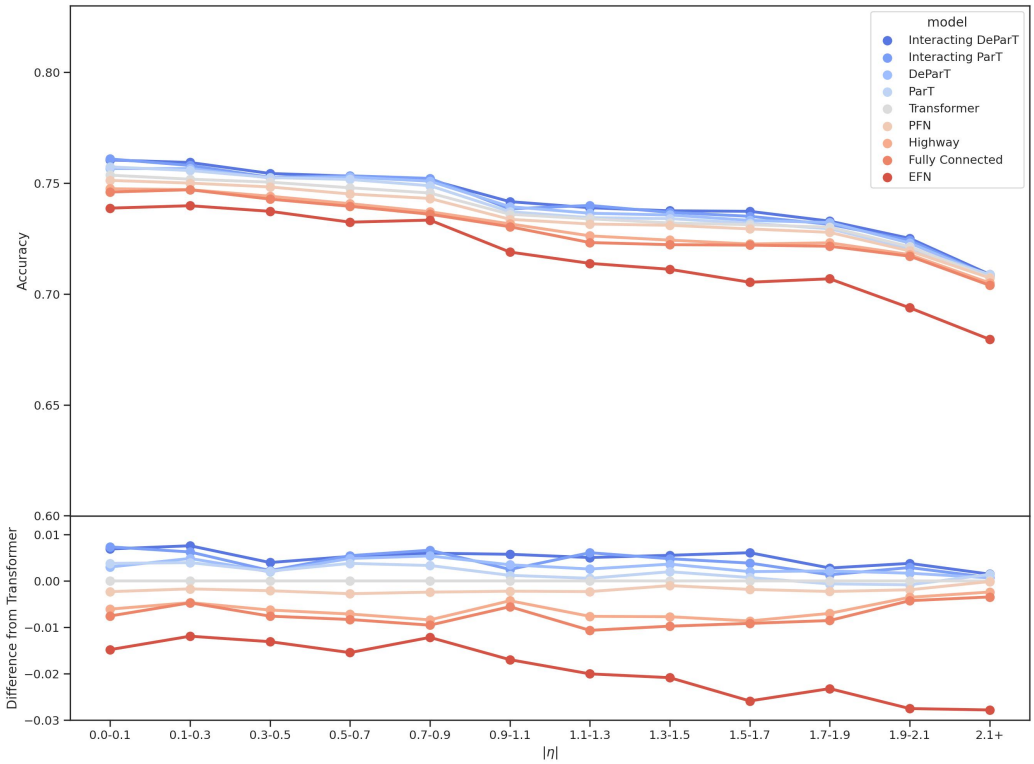


Figure 5.11 Accuracy of the models as a function of $|\eta|$ and relative to the Transformer model.

In figure 5.11, we can see that the accuracy of the models is way less dependent on the $|\eta|$ than on the p_{T}. The only slightly different part is at $|\eta| \approx 2.1$, where the tracker cannot measure the full jets. We also see that EFN is more dependent on the $|\eta|$ than the other models. This may be caused by the fact that EFN uses per PFO mapping only on the angular variables causing the model to be more sensitive to the $|\eta|$.

The relative performance of the models, shown in figure 5.12, is similar to the p_{T} dependence.

Additional metrics are plotted in appendix C.4.

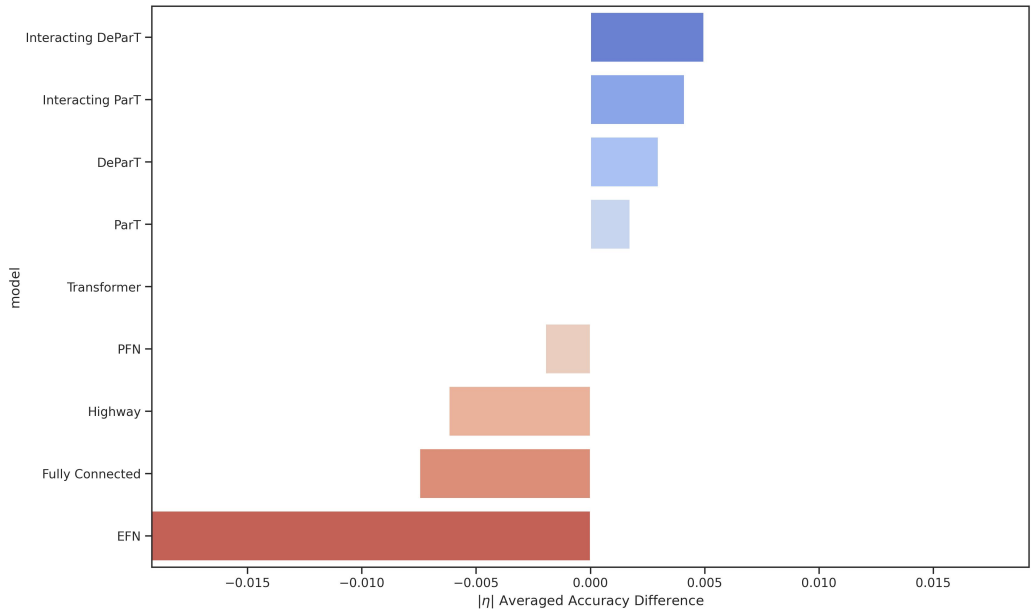


Figure 5.12 $|\eta|$ averaged difference from the Transformer model.

5.5.3 Pileup Dependence

Dependence on the pileup μ is shown in figure 5.13. It impacts the performance of the models the least out of the three variables. The trend is slightly downward, but it is not significant.

The relative performance of the models, shown in figure 5.14, is similar to the p_{T} and $|\eta|$ dependence.

Additional metrics are plotted in appendix C.5.

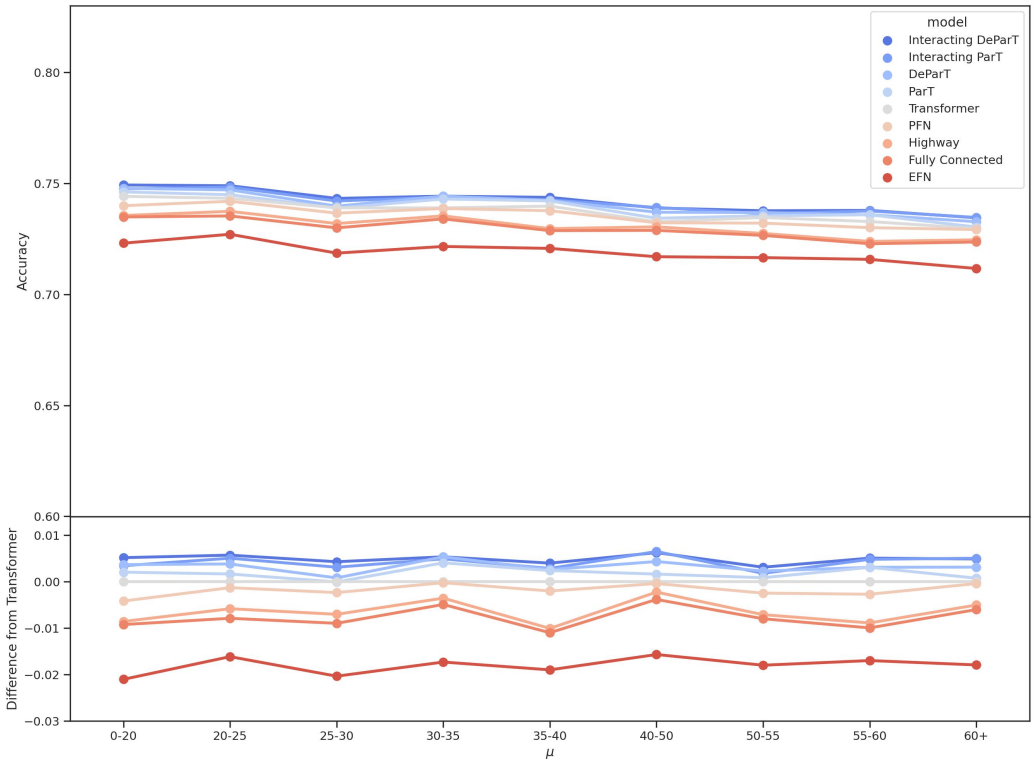


Figure 5.13 Accuracy of the models as a function of μ and relative to the Transformer model.

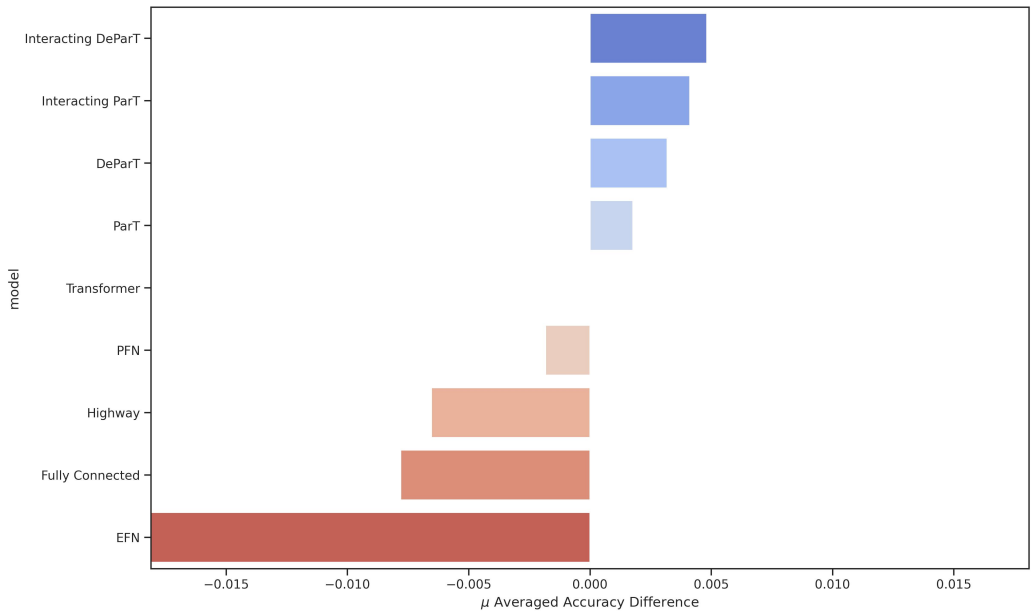


Figure 5.14 μ averaged difference from the Transformer model.

Conclusion

We have successfully developed a modern Transformer-like model that uses jet constituents to determine the original parton initiating the jet. To achieve this, we have reviewed the crucial physical aspects of QCD describing the behavior of partons and jets and the detector response to jets. The training data was a Monte Carlo simulation passed through the detector simulation with the object reconstruction the same as in the real data, provided by the ATLAS Run 2 simulations.

We reviewed modern architectures and techniques of Deep Learning and applied them to the problem of jet tagging. We touched on multiple problems that may arise in particle physics applications of Deep Learning, such as the truthfulness of labels, reconstruction of data, and the underlying physics.

Different models using various levels of jet information were trained and compared. Our proposed DeParT model outperformed all of them, including the previously used quark/gluon tagger and the state-of-the-art jet tagger ParT . We have demonstrated the Transformer architecture's advantage in extracting information from the jet constituents. They provide a more complete and flexible description of the jet than the previous models. On top of that, the Transformer architectures have far more immense potential, where it could be used to tag individual flavors of quarks separately on top of the quark/gluon tagging. The only downside of the model is a large number of parameters, which makes it computationally expensive.

The proposed model showed only little angular and pileup dependence on the performance, which is a desirable property for a jet tagger. There is room for improvement of the model in the energy dependence, which is a more challenging problem since the physics changes drastically.

Future work

A possible improvement is to make the model Infrared and Collinear Safe, which might be achieved by adding data augmentation steps. This would make the model more robust to the effects of the jet reconstruction and allow for a more

physical description.

The dependence of the model on the simulation framework should be investigated to make the model applicable. If the model performs similarly on different simulators (without being explicitly trained on them), we could be confident that the model learned physics and not the simulation's specifics. From this point, we would test the performance on real data and calibrate MC . Finally, the DeParT could be used in real data analysis.

Bibliography

[1] Murray Gell-Mann. "A Schematic Model of Baryons and Mesons". *Phys. Lett.* 8 (1964), pp. 214–215. DOI: 10.1016/S0031-9163(64)92001-3.

[2] *CERN (The European Organization for Nuclear Research)*. http://www.cern.ch/, last accessed April 29, 2023.

[3] "LHC Machine". *JINST* 3 (2008). Ed. by Lyndon Evans and Philip Bryant, S08001. DOI: 10.1088/1748-0221/3/08/S08001.

[4] G. Aad et al. "The ATLAS Experiment at the CERN Large Hadron Collider". *JINST* 3 (2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003.

[5] Matteo Cacciari, Gavin P. Salam, and Gregory Soyez. "The anti-k_t jet clustering algorithm". *JHEP* 04 (2008), p. 063. DOI: 10.1088/1126-6708/2008/04/063. arXiv: 0802.1189 [hep-ph].

[6] Morad Aaboud et al. "Jet reconstruction and performance using particle flow with the ATLAS Detector". *Eur. Phys. J. C* 77.7 (2017), p. 466. DOI: 10.1140/epjc/s10052-017-5031-2. arXiv: 1703.10485 [hep-ex].

[7] M. Aaboud et al. "Search for R-parity-violating supersymmetric particles in multi-jet final states produced in p–p collisions at s=13 TeV using the ATLAS detector at the LHC". *Physics Letters B* 785 (2018), pp. 136–158. ISSN: 0370-2693. DOI: https://doi.org/10.1016/j.physletb.2018.08.021. URL: https://www.sciencedirect.com/science/article/pii/S0370269318306270.

[8] Huilin Qu, Congqiao Li, and Sitian Qian. "Particle Transformer for Jet Tagging" (Feb. 2022). arXiv: 2202.03772 [hep-ph].

[9] ATLAS. *Quark versus Gluon Jet Tagging Using Charged Particle Multiplicity with the ATLAS Detector*. Tech. rep. Geneva: CERN, 2017. URL: https://cds.cern.ch/record/2263679.

[10] Yann Coadou. "Boosted decision trees" (Mar. 2022). DOI: 10.1142/9789811234033_0002. arXiv: 2206.09645 [physics.data-an].

[11] Wanyun Su et al. *Calibration of the quark/gluon jet tagging variables using matrix method with the ATLAS detector*. Tech. rep. Geneva: CERN, 2022. URL: https://cds.cern.ch/record/2802919.

[12] Jacob Devlin et al. "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding". *CoRR* abs/1810.04805 (2018). arXiv: 1810.04805. URL: http://arxiv.org/abs/1810.04805.

[13] Hugo Touvron, Matthieu Cord, and Herv'e J'egou. "DeiT III: Revenge of the ViT". In: *European Conference on Computer Vision*. 2022.

[14] Tom B. Brown et al. "Language Models are Few-Shot Learners". *CoRR* abs/2005.14165 (2020). arXiv: 2005.14165. URL: https://arxiv.org/abs/2005.14165.

[15] OpenAI. *GPT-4 Technical Report*. 2023. arXiv: 2303.08774 [cs.CL].

[16] Robin Rombach et al. "High-Resolution Image Synthesis With Latent Diffusion Models". In: *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*. 2022, pp. 10684–10695.

[17] Aditya Ramesh et al. "Zero-Shot Text-to-Image Generation". *CoRR* (2021). arXiv: 2102.12092.

[18] Rupesh Kumar Srivastava, Klaus Greff, and Jürgen Schmidhuber. "Highway Networks". *CoRR* abs/1505.00387 (2015). arXiv: 1505.00387. URL: http://arxiv.org/abs/1505.00387.

[19] C. Van Der Malsburg. "Frank Rosenblatt: Principles of Neurodynamics: Perceptrons and the Theory of Brain Mechanisms". In: *Brain Theory*. Ed. by Günther Palm and Ad Aertsen. Berlin, Heidelberg: Springer Berlin Heidelberg, 1986, pp. 245–248. ISBN: 978-3-642-70911-1.

[20] Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler. "Energy Flow Networks: Deep Sets for Particle Jets". *JHEP* 01 (2019), p. 121. DOI: 10.1007/JHEP01(2019)121. arXiv: 1810.05165 [hep-ph].

[21] Ashish Vaswani et al. "Attention Is All You Need". *CoRR* abs/1706.03762 (2017). arXiv: 1706.03762. URL: http://arxiv.org/abs/1706.03762.

[22] Hugo Touvron et al. "Going deeper with Image Transformers". *CoRR* abs/2103.17239 (2021). arXiv: 2103.17239. URL: https://arxiv.org/abs/2103.17239.

[23] Thorben Finke et al. "Learning the language of QCD jets with transformers" (Mar. 2023). arXiv: 2303.07364 [hep-ph].

[24] S. Chatrchyan et al. "The CMS Experiment at the CERN LHC". *JINST* 3 (2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004.

[25] P. A. Zyla et al. “Review of Particle Physics”. *PTEP* 2020.8 (2020), p. 083C01. DOI: 10.1093/ptep/ptaa104.

[26] S. M. Oser. “Neutrino oscillation results from the Sudbury Neutrino Observatory”. In: *31st International Conference on High Energy Physics*. Dec. 2001, pp. 8–17.

[27] P. A. M. Dirac. “The Quantum Theory of the Emission and Absorption of Radiation”. *Proceedings of the Royal Society of London Series A* 114.767 (Mar. 1927), pp. 243–265. DOI: 10.1098/rspa.1927.0039.

[28] R. P. Feynman. “Space-Time Approach to Quantum Electrodynamics”. *Physical Review* 76.6 (Sept. 1949), pp. 769–789. DOI: 10.1103/PhysRev.76.769.

[29] Marián Fecko. *Differential Geometry and Lie Groups for Physicists*. Cambridge University Press, 2006. DOI: 10.1017/CBO9780511755590.

[30] Matthew D. Schwartz. *Quantum Field Theory and the Standard Model*. Cambridge University Press, Mar. 2014. ISBN: 978-1-107-03473-0, 978-1-107-03473-0.

[31] F. Halzen and Alan D. Martin. *QUARKS AND LEPTONS: AN INTRODUCTORY COURSE IN MODERN PARTICLE PHYSICS*. 1984. ISBN: 978-0-471-88741-6.

[32] J. Chýla. *Quartks, partons and Quantum Chromodynamics*. 2009.

[33] Toichiro Kinoshita. “Mass Singularities of Feynman Amplitudes”. *Journal of Mathematical Physics* 3.4 (July 1962), pp. 650–677. DOI: 10.1063/1.1724268.

[34] T. D. Lee and M. Nauenberg. “Degenerate Systems and Mass Singularities”. *Physical Review* 133.6B (Mar. 1964), B1549–B1562. DOI: 10.1103/PhysRev.133.B1549.

[35] Gavin P. Salam. “Elements of QCD for hadron colliders”. In: *2009 European School of High-Energy Physics*. Nov. 2010. arXiv: 1011.5131 [hep-ph].

[36] Bo Andersson et al. “Parton Fragmentation and String Dynamics”. *Phys. Rept.* 97 (1983), pp. 31–145. DOI: 10.1016/0370-1573(83)90080-7.

[37] B. R. Webber. “A QCD Model for Jet Fragmentation Including Soft Gluon Interference”. *Nucl. Phys. B* 238 (1984), pp. 492–528. DOI: 10.1016/0550-3213(84)90333-X.

[38] Richard P. Feynman. “Very High-Energy Collisions of Hadrons”. *Phys. Rev. Lett.* 23 (24 1969), pp. 1415–1417. DOI: 10.1103/PhysRevLett.23.1415. URL: https://link.aps.org/doi/10.1103/PhysRevLett.23.1415.

[39] Martin zur Nedden. *The Run-2 ATLAS Trigger System: Design, Performance and Plan*. Tech. rep. Geneva: CERN, 2016. URL: https://cds.cern.ch/record/2238679.

[40] Torbjörn Sjöstrand et al. "An introduction to PYTHIA 8.2". *Computer Physics Communications* 191 (2015), pp. 159–177. DOI: 10.1016/j.cpc.2015.01.024. URL: https://doi.org/10.1016%2Fj.cpc.2015.01.024.

[41] M Asai. "Introduction to Geant4" (2000). DOI: 10.5170/CERN-2000-013.107. URL: https://cds.cern.ch/record/491492.

[42] G. Aad et al. "The ATLAS Simulation Infrastructure". *Eur. Phys. J. C* 70 (2010), pp. 823–874. DOI: 10.1140/epjc/s10052-010-1429-9. arXiv: 1005.4568 [physics.ins-det].

[43] "ATLAS Pythia 8 tunes to 7 TeV data" (Nov. 2014).

[44] Stefano Carrazza, Stefano Forte, and Juan Rojo. "Parton Distributions and Event Generators". In: *43rd International Symposium on Multiparticle Dynamics*. 2013, pp. 89–96. arXiv: 1311.5887 [hep-ph].

[45] John Back, Michal Kreps, and Thomas Latham. *EvtGen*. https://evtgen.hepforge.org. 2020.

[46] Georges Aad et al. "Topological cell clustering in the ATLAS calorimeters and its performance in LHC Run 1". *Eur. Phys. J. C* 77 (2017), p. 490. DOI: 10.1140/epjc/s10052-017-5004-5. arXiv: 1603.02934 [hep-ex].

[47] ATLAS. "Tagging and suppression of pileup jets" (May 2014).

[48] Morad Aaboud et al. "Identification and rejection of pile-up jets at high pseudorapidity with the ATLAS detector". *Eur. Phys. J. C* 77.9 (2017). [Erratum: Eur.Phys.J.C 77, 712 (2017)], p. 580. DOI: 10.1140/epjc/s10052-017-5081-5. arXiv: 1705.02211 [hep-ex].

[49] Ryan Atkin. "Review of jet reconstruction algorithms". *J. Phys. Conf. Ser.* 645.1 (2015). Ed. by Alan S. Cornell and Bruce Mellado, p. 012008. DOI: 10.1088/1742-6596/645/1/012008.

[50] Stefan Weinzierl. "The SISCone jet algorithm optimised for low particle multiplicities". *Comput. Phys. Commun.* 183 (2012), pp. 813–820. DOI: 10.1016/j.cpc.2011.12.007. arXiv: 1108.1934 [hep-ph].

[51] S. Catani et al. "Longitudinally invariant K_t clustering algorithms for hadron hadron collisions". *Nucl. Phys. B* 406 (1993), pp. 187–224. DOI: 10.1016/0550-3213(93)90166-M.

[52] Yuri L. Dokshitzer et al. "Better jet clustering algorithms". *JHEP* 08 (1997), p. 001. DOI: 10.1088/1126-6708/1997/08/001. arXiv: hep-ph/9707323.

[53] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. http://www.deeplearningbook.org. MIT Press, 2016.

[54] Kaiming He et al. "Masked Autoencoders Are Scalable Vision Learners". *CoRR* abs/2111.06377 (2021). arXiv: 2111.06377. URL: https://arxiv.org/abs/2111.06377.

[55] Claude Elwood Shannon. "A mathematical theory of communication". *ACM SIGMOBILE mobile computing and communications review* 5.1 (2001), pp. 3–55.

[56] David J. C. MacKay. *Information Theory, Inference, and Learning Algorithms*. Copyright Cambridge University Press, 2003.

[57] John VonNeumann. *Mathematische Grundlagen der Quantenmechanik*. ger. Berlin [u.a.]: Springer, 1932. URL: http://eudml.org/doc/203794.

[58] E.K.P. Chong and S.H. Zak. *An Introduction to Optimization*. Wiley-Interscience Series in Discrete Mathematics and Optimi. Wiley, 2004. ISBN: 9780471654001. URL: https://books.google.sk/books?id=bKdibaXVuUEC.

[59] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[60] Yang You et al. "Reducing BERT Pre-Training Time from 3 Days to 76 Minutes". *CoRR* abs/1904.00962 (2019). arXiv: 1904.00962. URL: http://arxiv.org/abs/1904.00962.

[61] Kurt Hornik, Maxwell Stinchcombe, and Halbert White. "Multilayer feedforward networks are universal approximators". *Neural networks* 2.5 (1989), pp. 359–366.

[62] Felix Gers, Jürgen Schmidhuber, and Fred Cummins. "Learning to Forget: Continual Prediction with LSTM". *Neural computation* 12 (Oct. 2000), pp. 2451–71. DOI: 10.1162/089976600300015015.

[63] Dan Hendrycks and Kevin Gimpel. "Bridging Nonlinearities and Stochastic Regularizers with Gaussian Error Linear Units". *CoRR* abs/1606.08415 (2016). arXiv: 1606.08415. URL: http://arxiv.org/abs/1606.08415.

[64] Prajit Ramachandran, Barret Zoph, and Quoc V. Le. "Searching for Activation Functions". *CoRR* abs/1710.05941 (2017). arXiv: 1710.05941. URL: http://arxiv.org/abs/1710.05941.

[65] Preetum Nakkiran et al. "Deep Double Descent: Where Bigger Models and More Data Hurt". *CoRR* abs/1912.02292 (2019). arXiv: 1912.02292. URL: http://arxiv.org/abs/1912.02292.

[66] Manzil Zaheer et al. “Deep Sets”. *CoRR* abs/1703.06114 (2017). arXiv: 1703.06114. URL: http://arxiv.org/abs/1703.06114.

[67] Binh-Son Hua, Minh-Khoi Tran, and Sai-Kit Yeung. “Point-wise Convolutional Neural Network”. *CoRR* abs/1712.05245 (2017). arXiv: 1712.05245. URL: http://arxiv.org/abs/1712.05245.

[68] ATLAS. *Constituent-Based Top-Quark Tagging with the ATLAS Detector*. Tech. rep. Geneva: CERN, 2022. URL: https://cds.cern.ch/record/2825328.

[69] Gao Huang et al. “Deep Networks with Stochastic Depth”. *CoRR* (2016). arXiv: 1603.09382. URL: http://arxiv.org/abs/1603.09382.

[70] Noam Shazeer. “GLU Variants Improve Transformer”. *CoRR* abs/2002.05202 (2020). arXiv: 2002.05202.

[71] Rene Brun et al. *root-project/root: v6.18/02*. Version v6-18-02. Aug. 2019. DOI: 10.5281/zenodo.3895860. URL: https://doi.org/10.5281/zenodo.3895860.

[72] Jim Pivarski et al. *scikit-hep/uproot: 3.12.0*. Version 3.12.0. July 2020. DOI: 10.5281/zenodo.3952728. URL: https://doi.org/10.5281/zenodo.3952728.

[73] Martín Abadi et al. *TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems*. Software available from tensorflow.org. 2015. URL: https://www.tensorflow.org/.

[74] François Chollet et al. *Keras*. https://keras.io. 2015.

Appendix A

Description of High-level Jet Variables

In this section, we provide a full list of high-level jet variables commonly used in ATLAS analyses. They come precomputed by `Athena` and come in the jet container. The list includes the name of the variable in the `ROOT` file, its description, and the symbol if used.

`ActiveArea4vec_(m,pt,eta,phi)` - 4-momentum of the energy carried by pileup in the cross section of a jet

`FracSamplingMax` - a fraction of energy deposited in the calorimeter cell with the highest energy deposition

`FracSamplingMaxIndex` - index of the calorimeter cell with the highest energy deposition

`GhostMuonSegmentCount` - number of muon segments (muon spectrometer) hit by jet constituents

`JetConstitScaleMomentum_(m,pt,eta,phi)` - uncalibrated 4-momentum of total constituent energy

`fJVT` (fJVT) - forward jet-vertex-tagger (see section 3.4)

`passFJVT` (passfJVT) - passed forward jet-vertex-tagger (see section 3.4)

`Jvt` (JVT) - jet-vertex-tagger (see section 3.4)

`JvtRpt` ($R_{p_{\mathrm{T}}}$) - a fraction of the jet momentum from primary vertex (see section 3.4)

`passJVT` (passJVT) - passed jet-vertex-tagger (see section 3.4)

`JVFCorr` (corrJVT) - corrected jet-vertex-fraction (see section 3.4)

`averageInteractionsPerCrossing` (μ) - average pileup (see section 3.4)

`Timing` - time of jet detection wrt. time of collision

`EMFrac` (f_{EM}) - a fraction of energy of a jet deposited in EM calorimeter

`Width` (W) - p_{T} weighted average distance of constituents from the jet axis (see table 5.3)

`chf` (f_{ch}) - charged fraction of a jet (p_{T} sum of track divided by $p_{\mathrm{T}}^{\mathrm{jet}}$)

`(m,pt,eta,phi)` ($m^{\mathrm{jet}}, p_{\mathrm{T}}^{\mathrm{jet}}, \eta^{\mathrm{jet}}, \phi^{\mathrm{jet}}$) - 4-momentum of calibrated jet

`PFO_n` (N_{PFO}) - number of PFOs

`ChargedPFOWidthPt1000[0]` ($W_{\mathrm{chPFO}}^{\mathrm{p_T}>1\mathrm{GeV}}$) - width of jet computed from charged PFOs with $p_{\mathrm{T}} > 1$ GeV from primary vertex

`TrackWidthPt1000[0]` ($W_{\mathrm{chtrk}}^{\mathrm{p_T}>1\mathrm{GeV}}$) - width of jet computed from tracks with $p_{\mathrm{T}} > 1$ GeV from primary vertex

`NumChargedPFOPt1000[0]` ($N_{\mathrm{chPFO}}^{\mathrm{p_T}>1\mathrm{GeV}}$) - number of charged PFOs with $p_{\mathrm{T}} > 1$ GeV from primary vertex

`NumChargedPFOPt500[0]` ($N_{\mathrm{chPFO}}^{\mathrm{p_T}>0.5\mathrm{GeV}}$) - number of charged PFOs with $p_{\mathrm{T}} > 0.5$ GeV from primary vertex

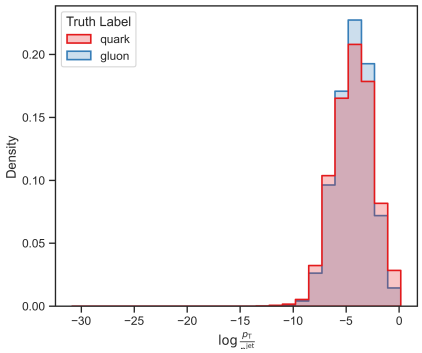
`SumPtChargedPFOPt500[0]` ($\sum_{\mathrm{chPFO}\in\mathrm{jet}} p_{\mathrm{T}}^{\mathrm{chPFO}}$) - sum of p_{T} of charged PFOs with $p_{\mathrm{T}} > 0.5$ GeV from primary vertex

Appendix B

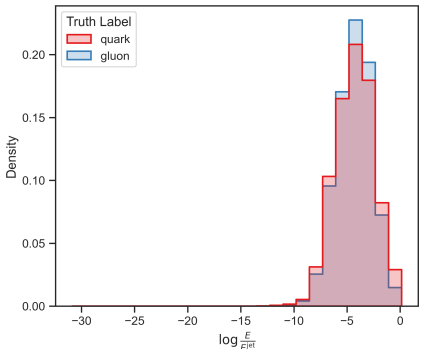
Input Variable Distributions

In this section, we provide distributions of all input variables. The dataset used is preprocessed as described in section 5.2.

B.1 PFO Variables

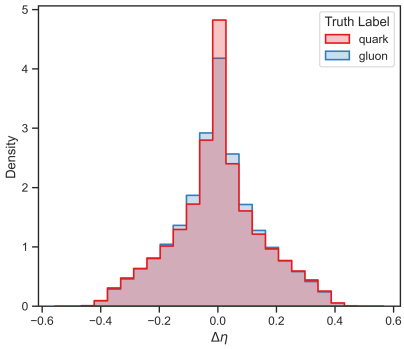


(a) PFO $\log p_{\mathrm{T}}/p_{\mathrm{T}}^{\mathrm{jet}}$

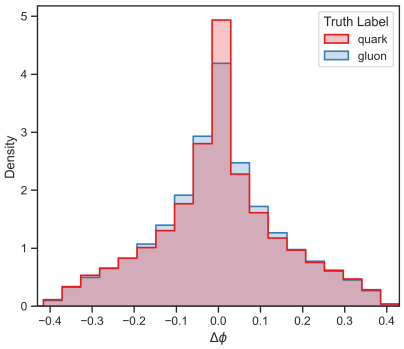


(b) PFO $\log E/E^{\mathrm{jet}}$

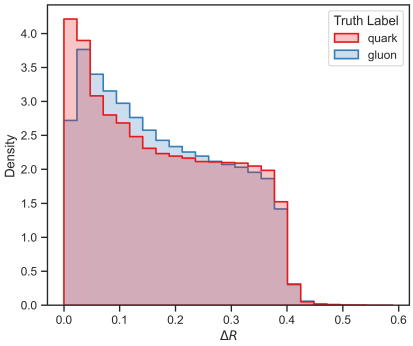
Figure B.1 Distributions of PFO variables part 1.



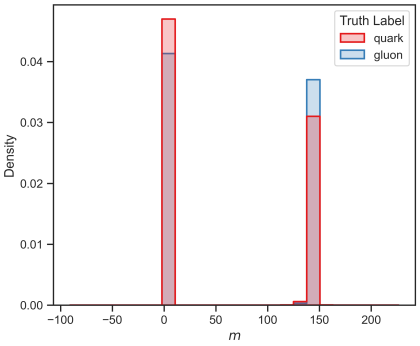
(a) PFO $\Delta\eta$



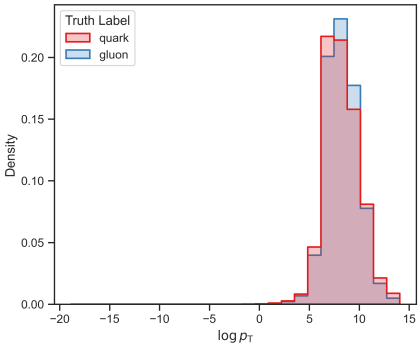
(b) PFO $\Delta\phi$



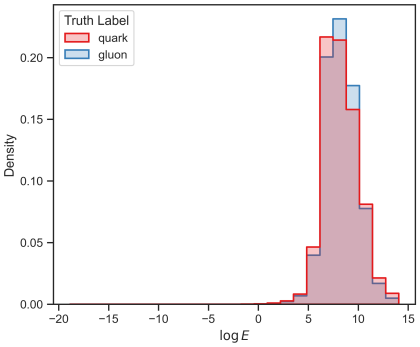
(c) PFO ΔR



(d) PFO m



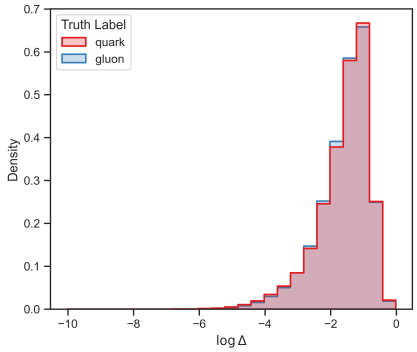
(e) PFO $\log p_{\mathrm{T}}$



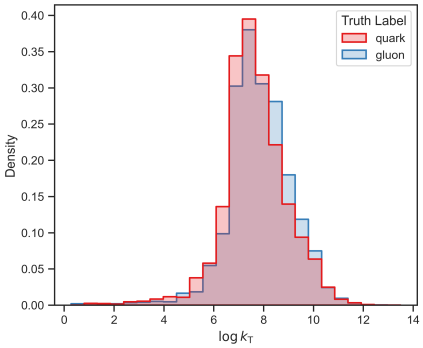
(f) PFO $\log E$

Figure B.2 Distributions of PFO variables part 2.

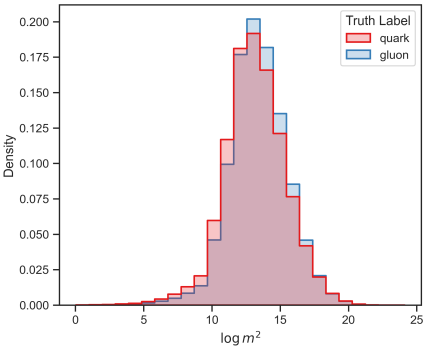
B.2 PFO Interaction Variables



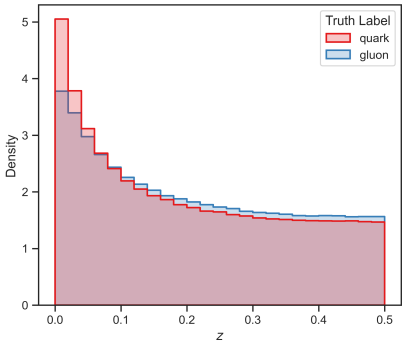
(a) PFO Interaction Δ



(b) PFO Interaction k_{T}



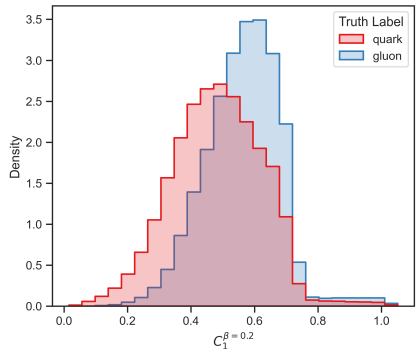
(c) PFO Interaction m^2



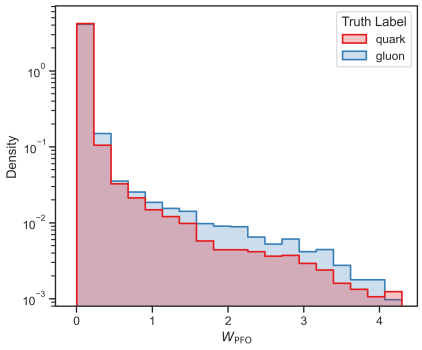
(d) PFO Interaction z

Figure B.3 Distributions of PFO interaction variables.

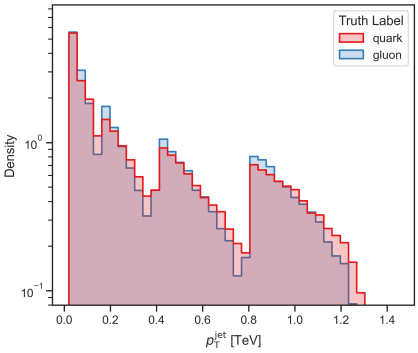
B.3 BDT Variables



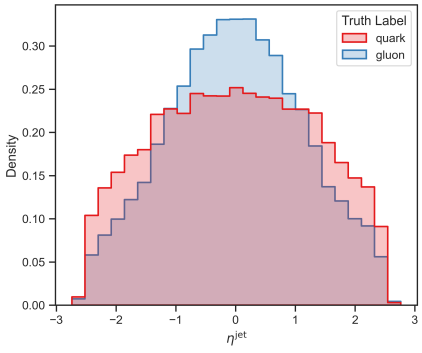
(a) BDT C_1



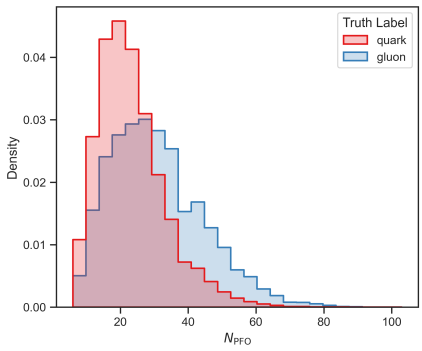
(b) BDT W_{PFO}



(c) BDT $p_{\mathrm{T}}^{\mathrm{jet}}$



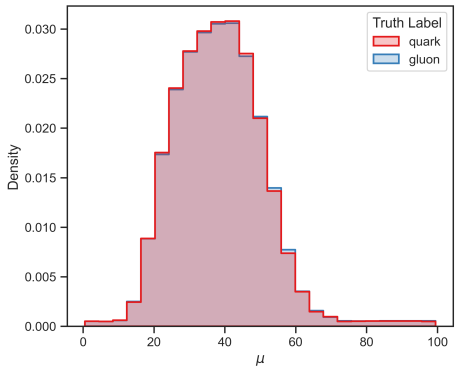
(d) BDT η^{jet}



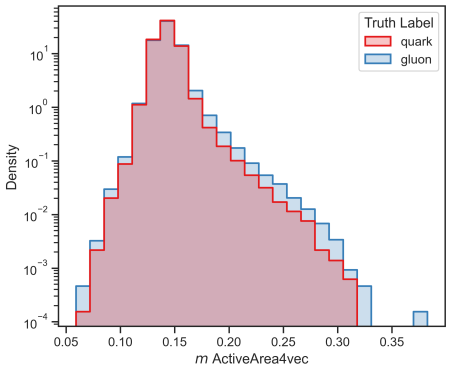
(e) BDT N_{PFO}

Figure B.4 Distributions of BDT variables.

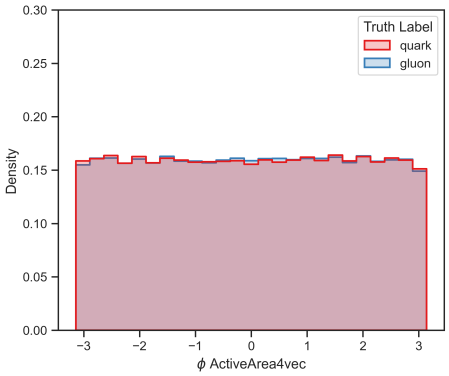
B.4 High-level Jet Variables



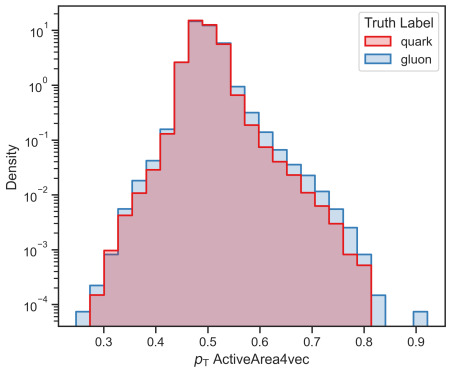
(a) `averageInteractionsPerCrossing[0]`



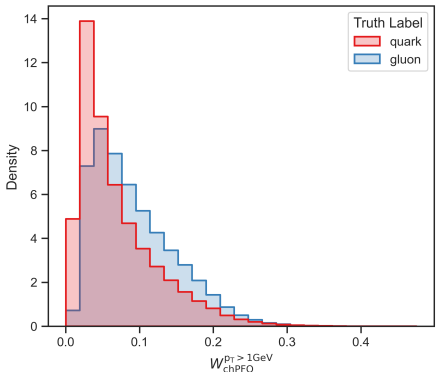
(b) `jets_ActiveArea4vec_m`



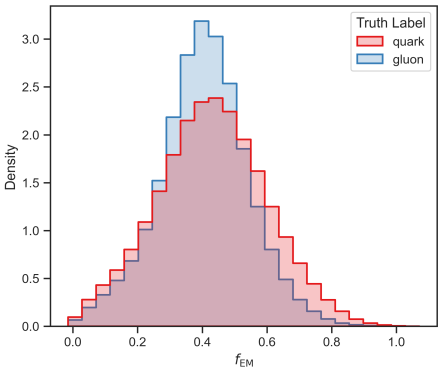
(c) `jets_ActiveArea4vec_phi`



(d) `jets_ActiveArea4vec_pt`

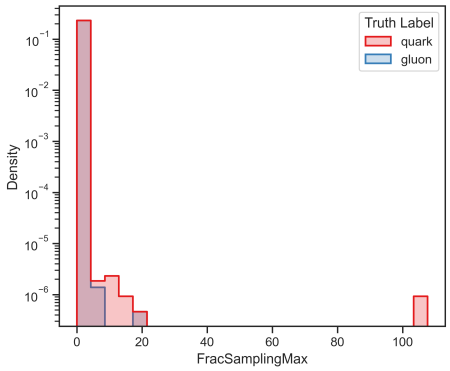


(e) `jets_ChargedPFOWidthPt1000[0]`

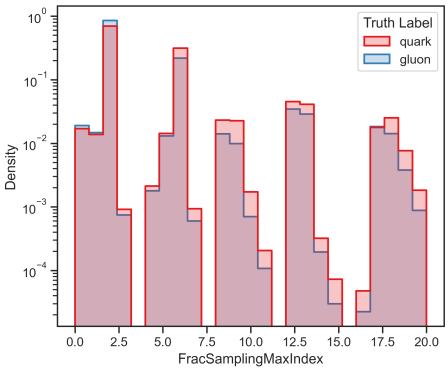


(f) `jets_EMFrac`

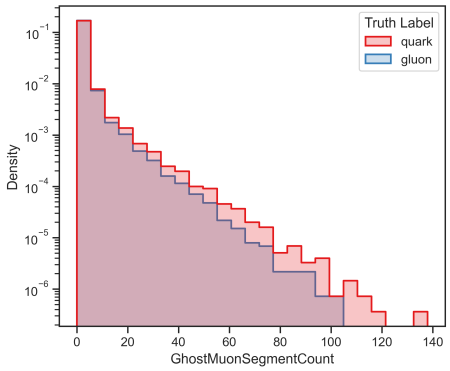
Figure B.5 High-level Jet Variables, part 1



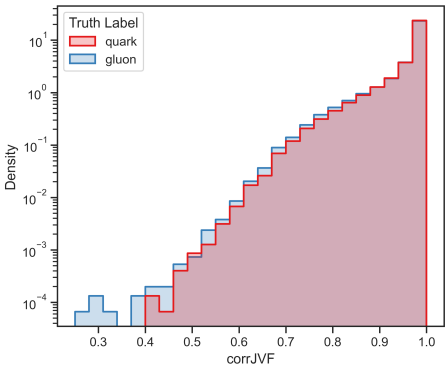
(a) `jets_FracSamplingMax`



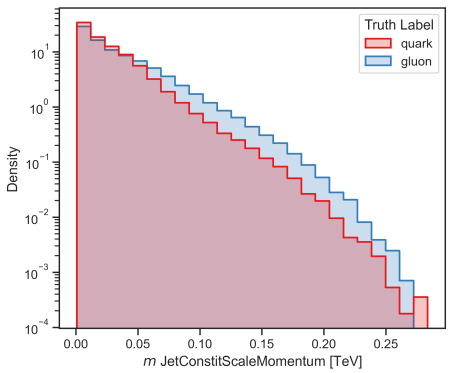
(b) `jets_FracSamplingMaxIndex`



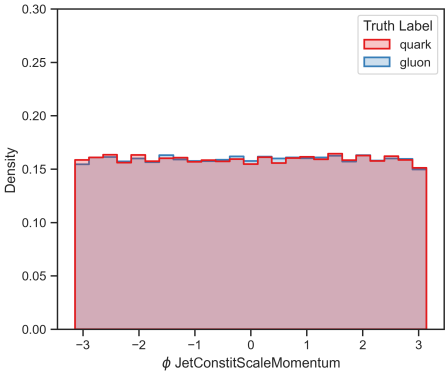
(c) `jets_GhostMuonSegmentCount`



(d) `jets_JVFCorr`

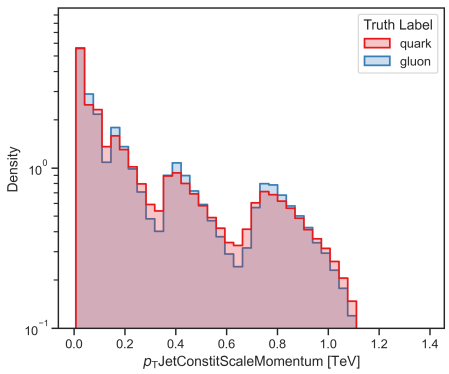


(e) `jets_JetConstitScaleMomentum_m`

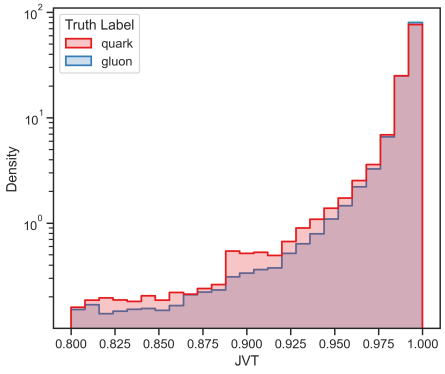


(f) `jets_JetConstitScaleMomentum_phi`

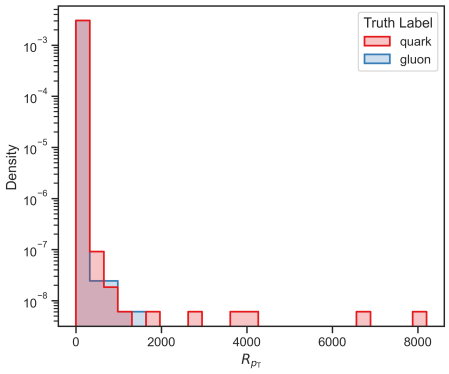
Figure B.6 High-level Jet Variables, part 2



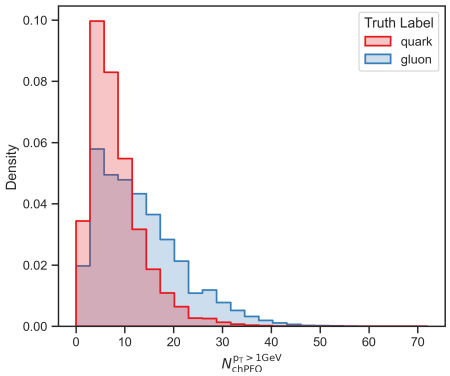
(a) `jets_JetConstitScaleMomentum_pt`



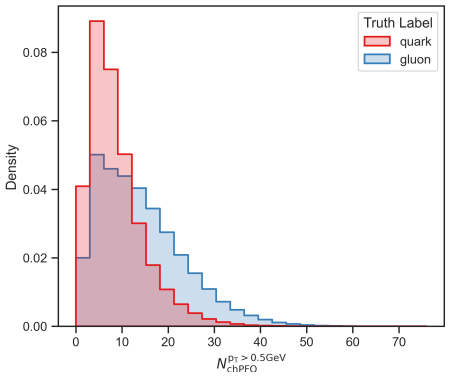
(b) `jets_Jvt`



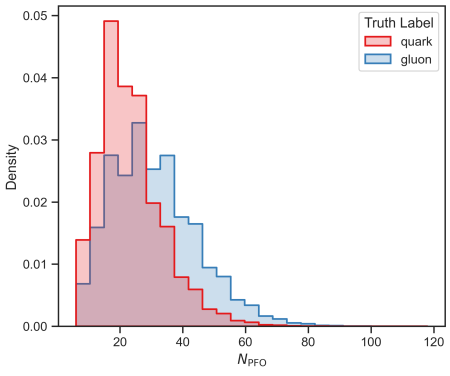
(c) `jets_JvtRpt`



(d) `jets_NumChargedPFOPt1000[0]`

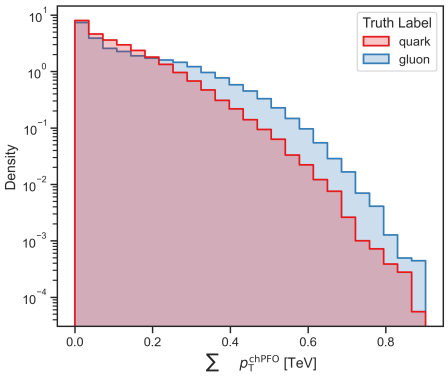


(e) `jets_NumChargedPFOPt500[0]`

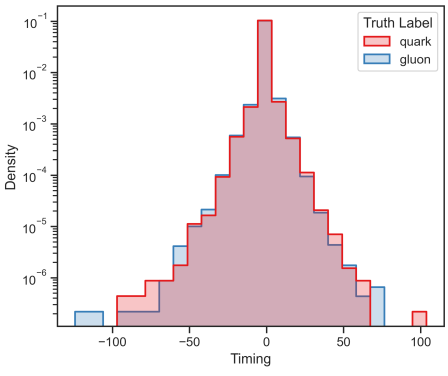


(f) `jets_PFO_n`

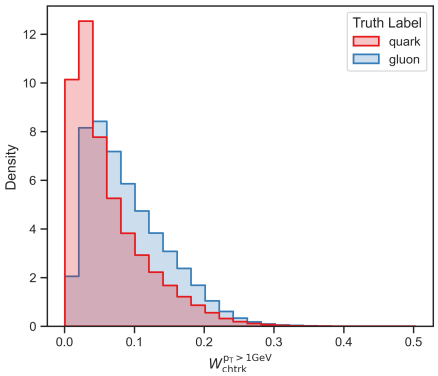
Figure B.7 High-level Jet Variables, part 3



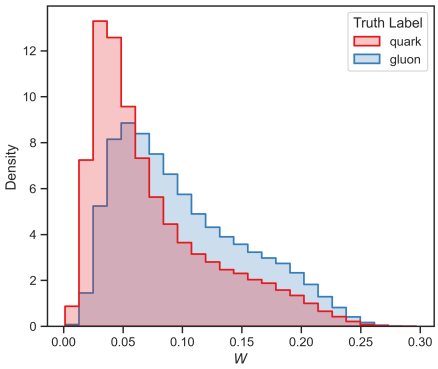
(a) `jets_SumPtChargedPFOPt500[0]`



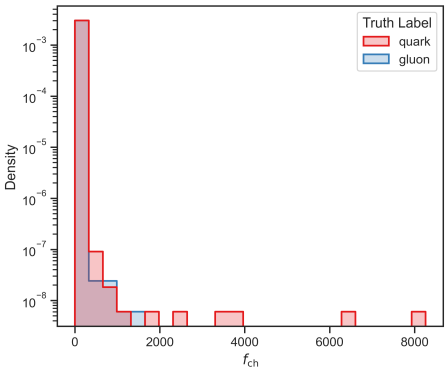
(b) `jets_Timing`



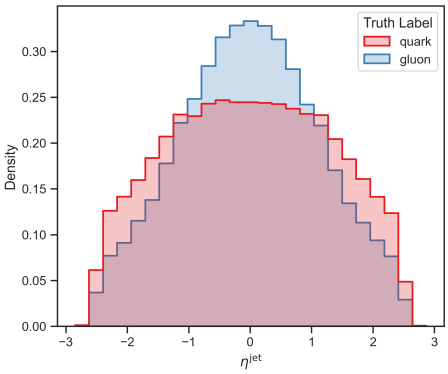
(c) `jets_TrackWidthPt1000[0]`



(d) `jets_Width`

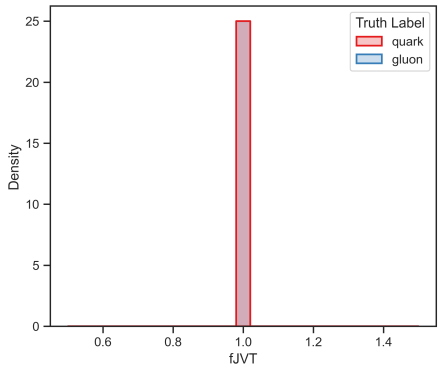


(e) `jets_chf`

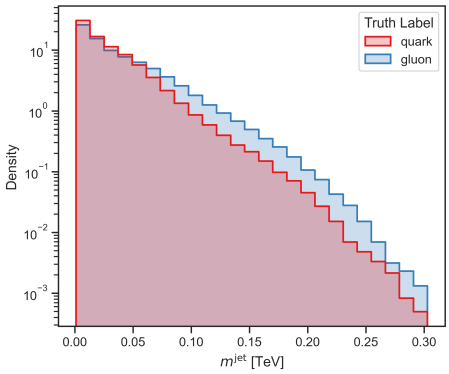


(f) `jets_eta`

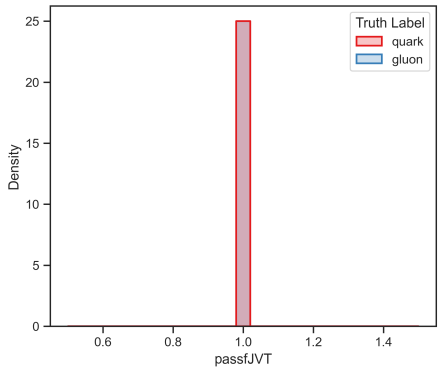
Figure B.8 High-level Jet Variables, part 4



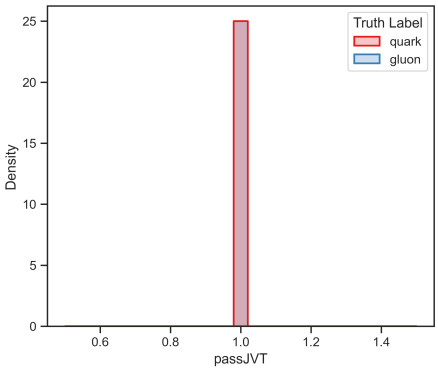
(a) `jets_fJVT`



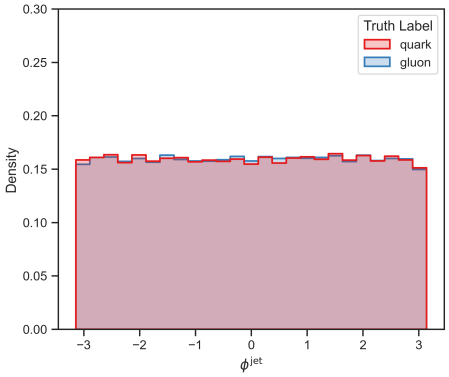
(b) `jets_m`



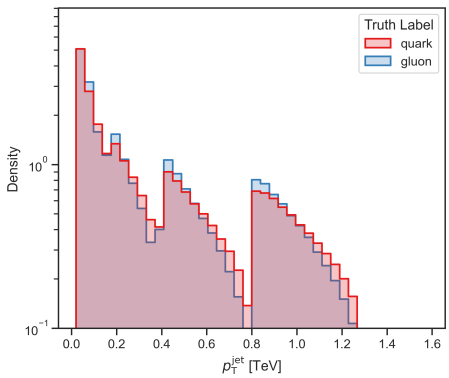
(c) `jets_passFJVT`



(d) `jets_passJVT`



(e) `jets_phi`



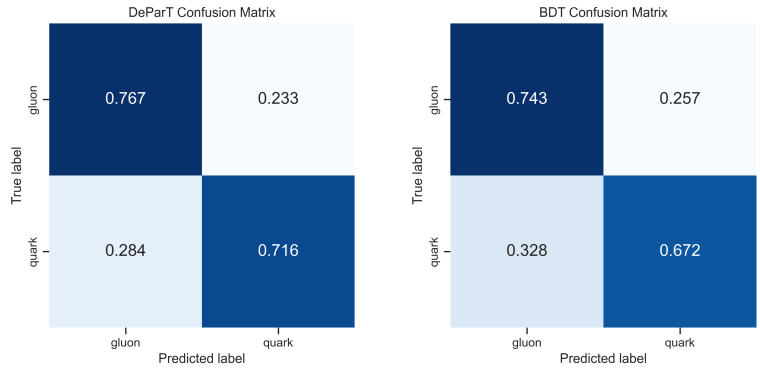
(f) `jets_pt`

Figure B.9 High-level Jet Variables, part 5

Appendix C

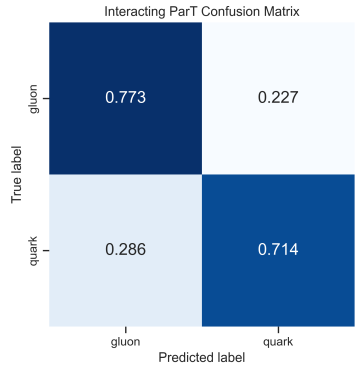
Additional Evaluation Plots

C.1 Confusion Matrix

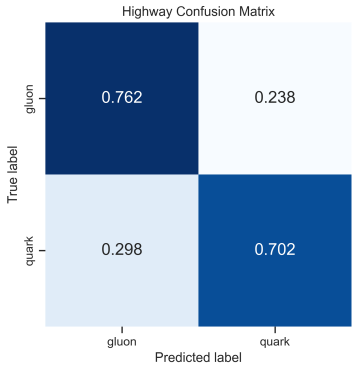


(a) DeParT

(b) BDT

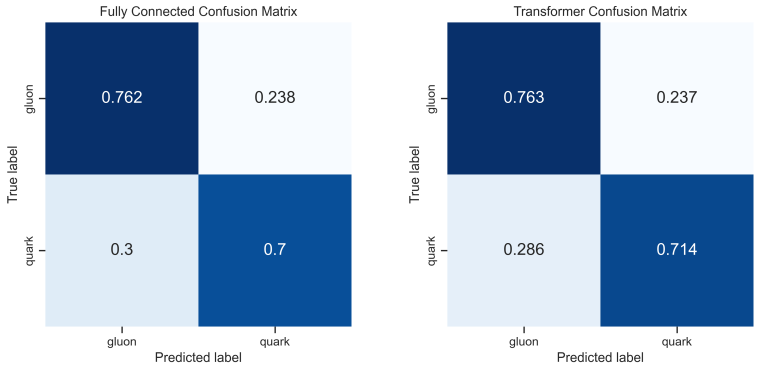


(c) Interacting ParT



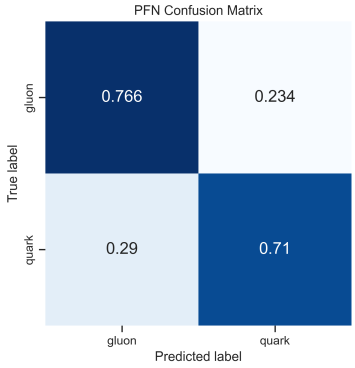
(d) Highway

Figure C.1 Confusion Matricies

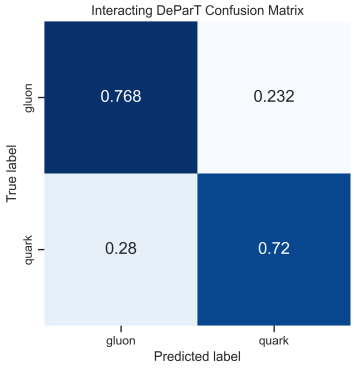


(a) Fully Connected

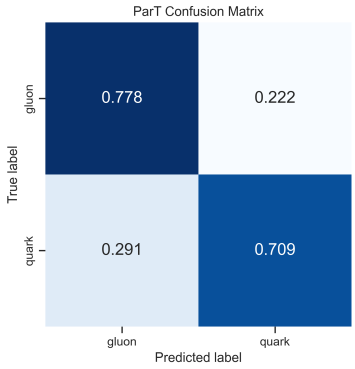
(b) Transformer



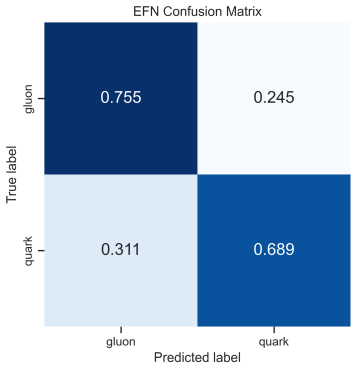
(c) PFN



(d) Interacting DeParT



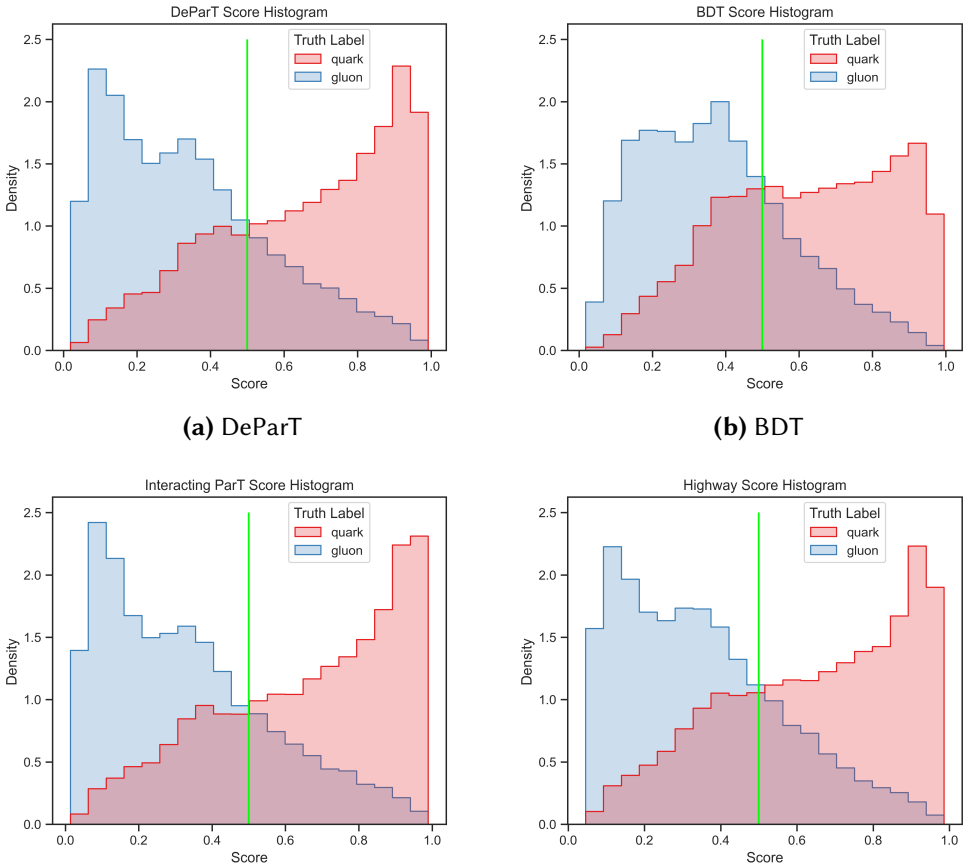
(e) ParT



(f) EFN

Figure C.2 Confusion Matricies

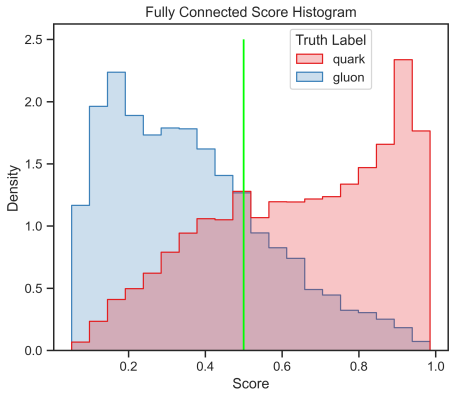
C.2 Score Histograms



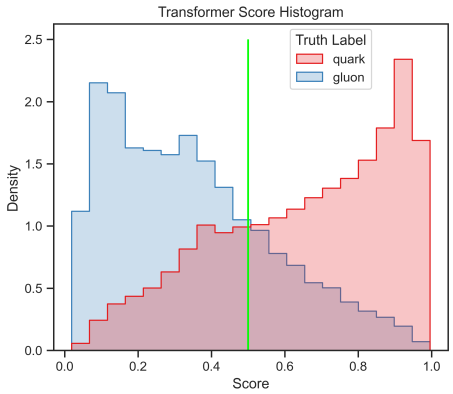
(a) DeParT **(b)** BDT

(c) Interacting ParT **(d)** Highway

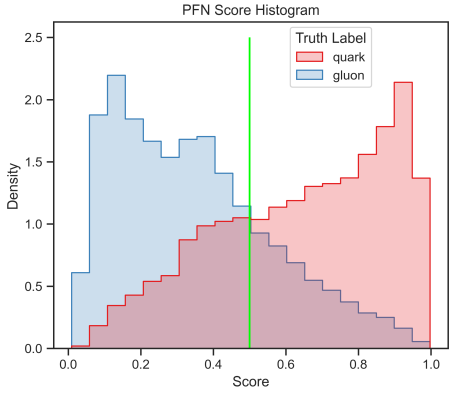
Figure C.3 Confusion Matricies



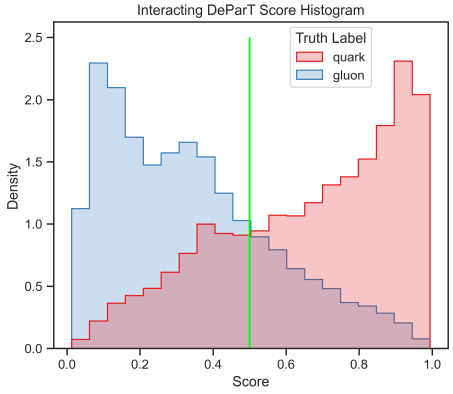
(a) Fully Connected



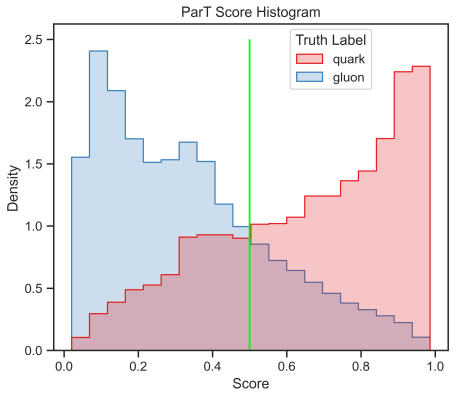
(b) Transformer



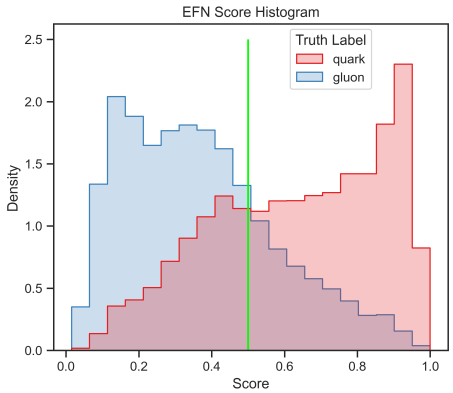
(c) PFN



(d) Interacting DeParT



(e) ParT



(f) EFN

Figure C.4 Confusion Matricies

C.3 Transverse Momentum Dependence

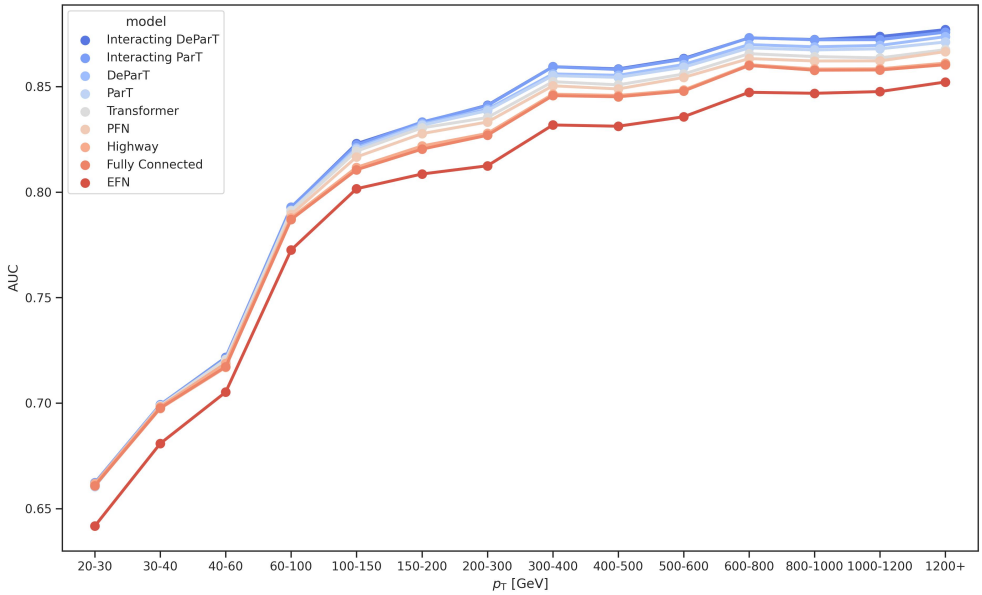


Figure C.5 AUC as a function of transverse momentum.

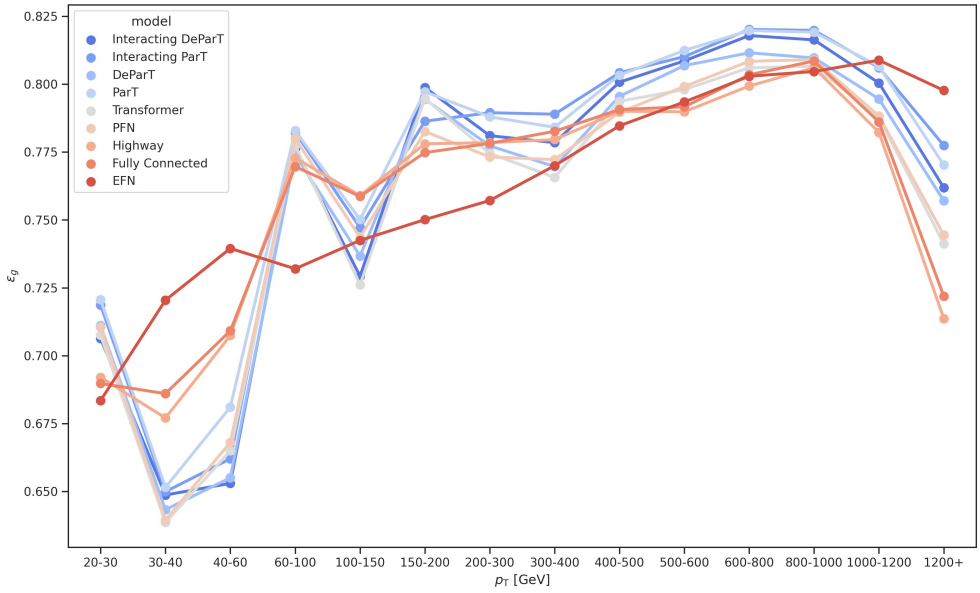


Figure C.6 Gluon efficiency as a function of transverse momentum.

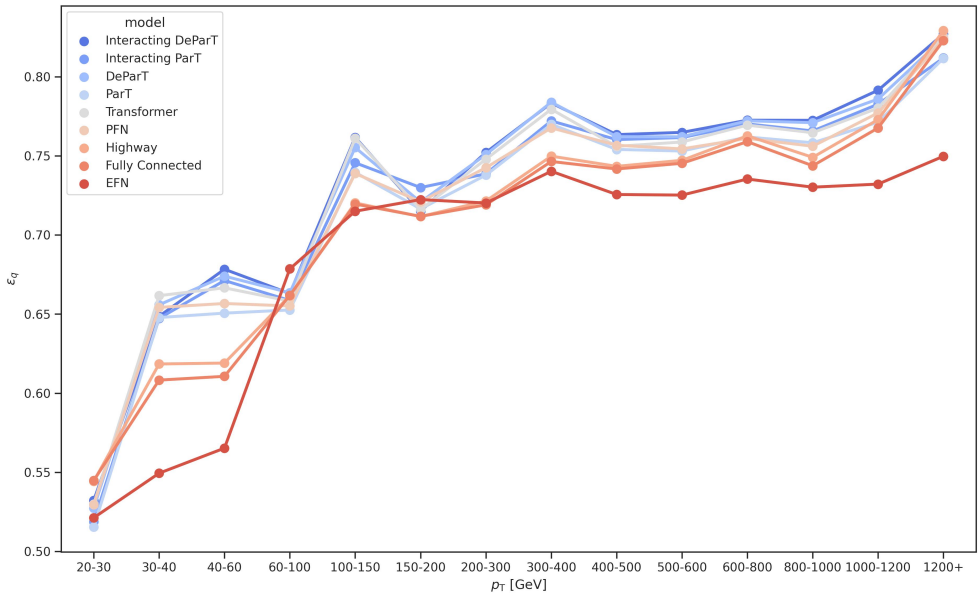


Figure C.7 Quark efficiency as a function of transverse momentum.

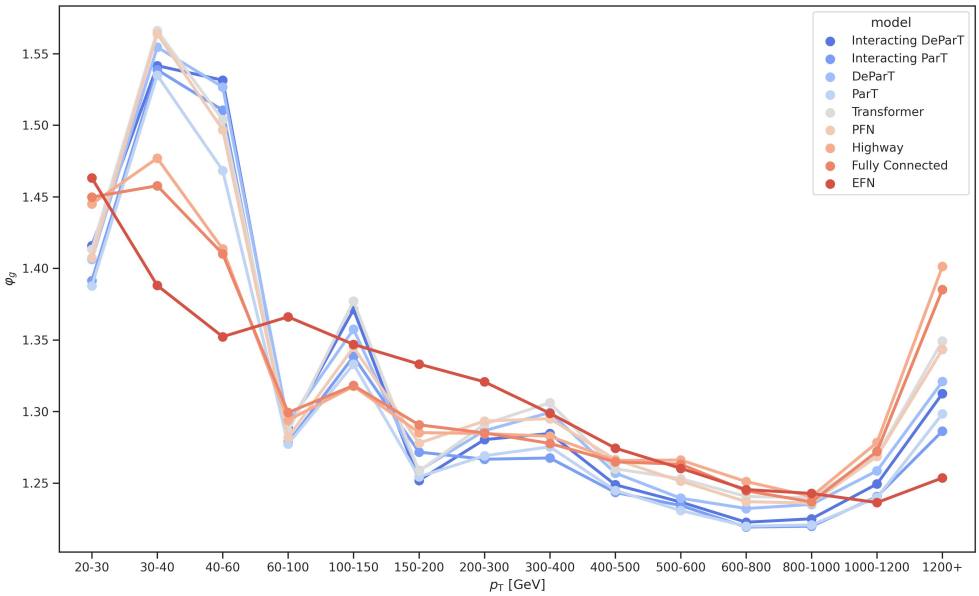


Figure C.8 Gluon rejection as a function of transverse momentum.

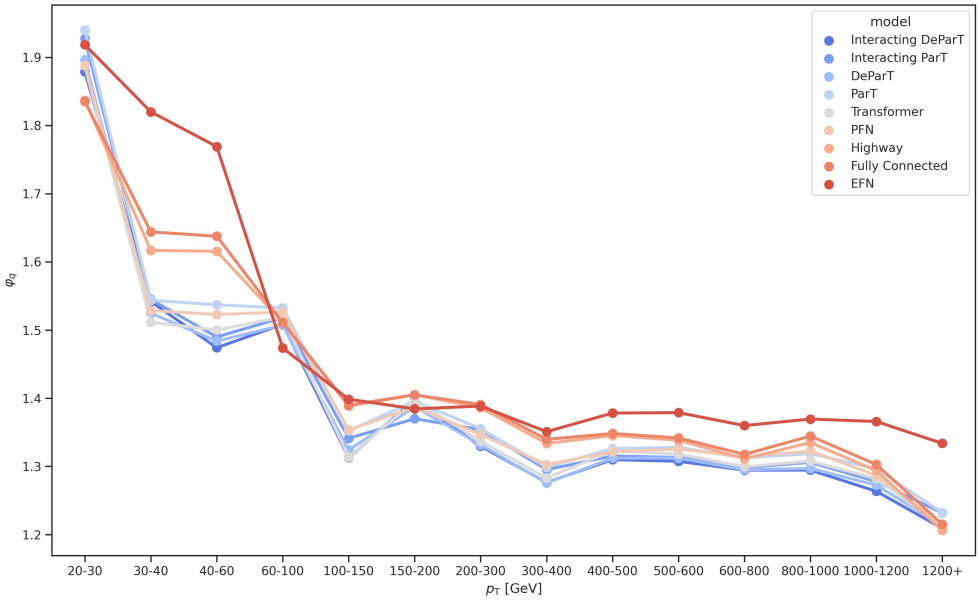


Figure C.9 Quark rejection as a function of transverse momentum.

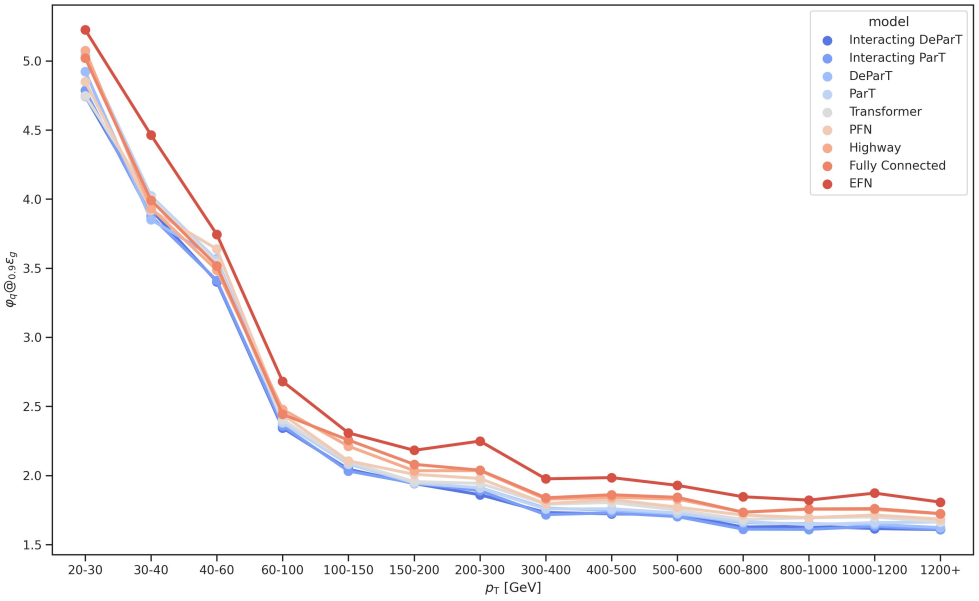


Figure C.10 Quark rejection at gluon efficiency of 0.9 as a function of transverse momentum.

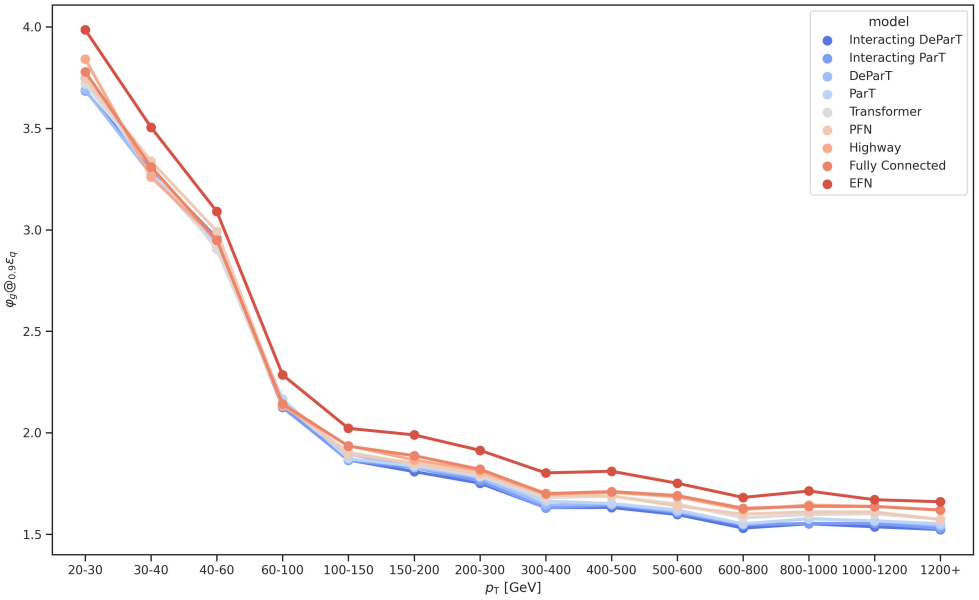


Figure C.11 Gluon rejection at quark efficiency of 0.9 as a function of transverse momentum.

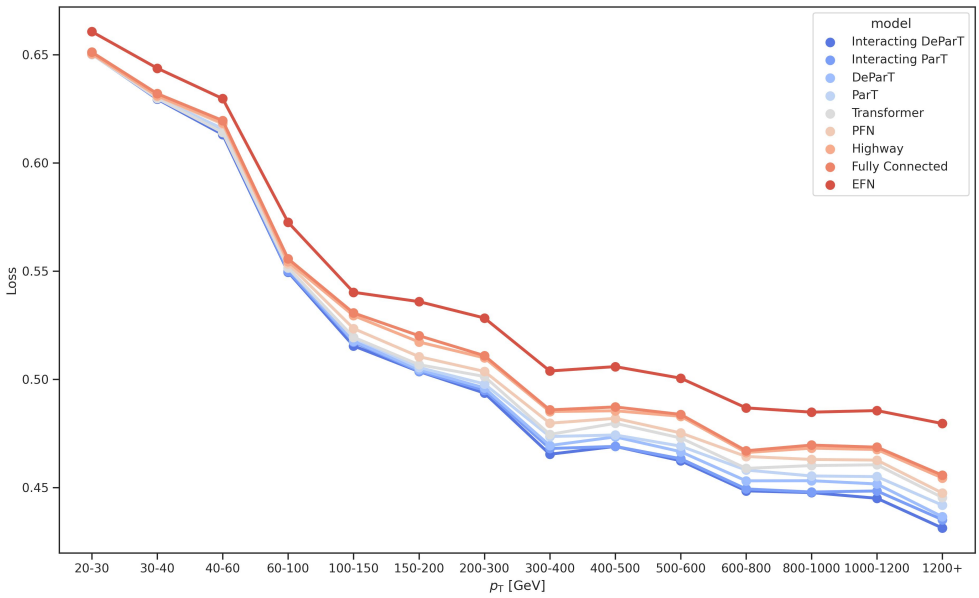


Figure C.12 Loss as a function of transverse momentum.

C.4 Pseudo-rapidity Dependence

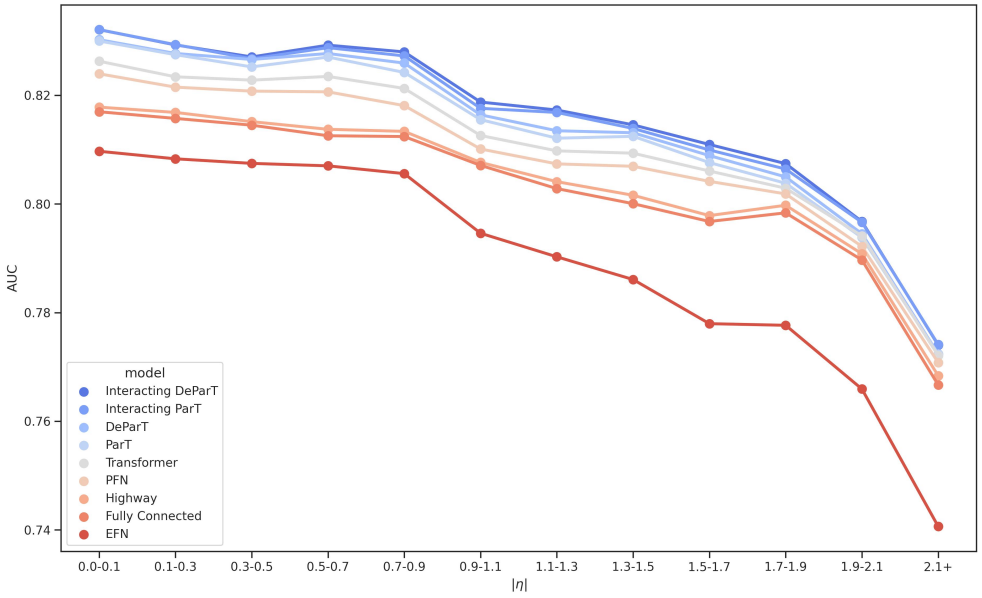


Figure C.13 AUC as a function of pseudo-rapidity.

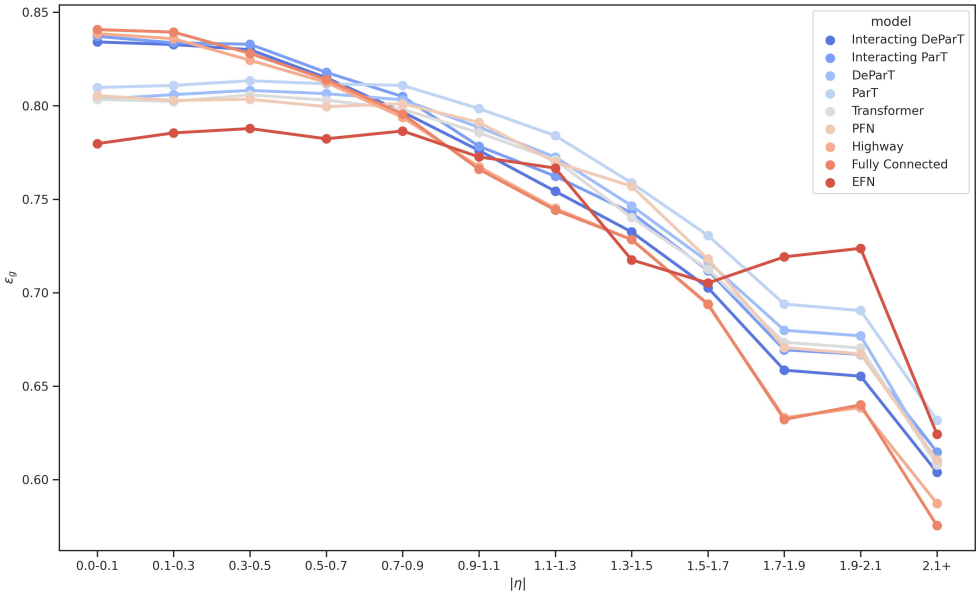


Figure C.14 Gluon efficiency as a function of pseudo-rapidity.

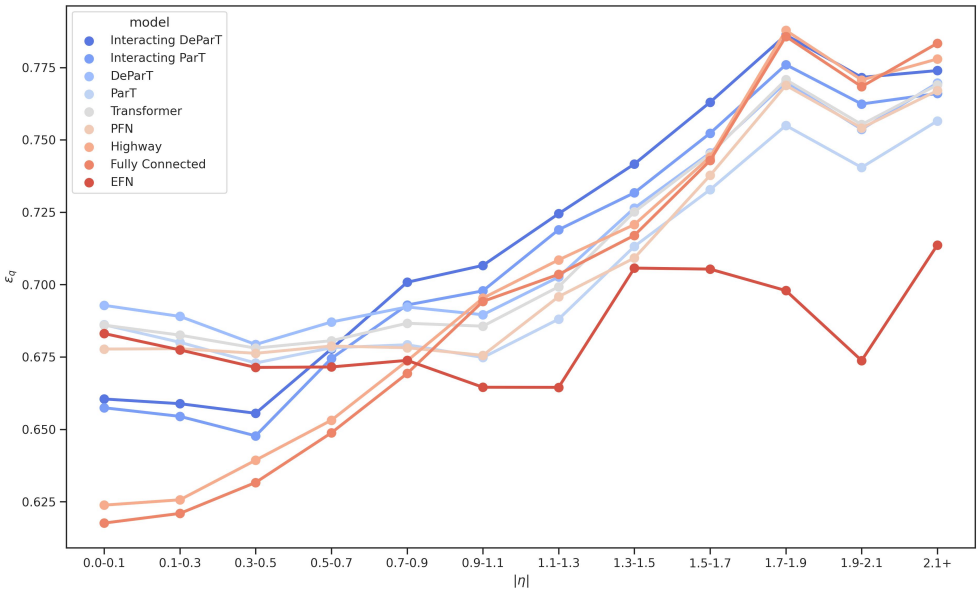


Figure C.15 Quark efficiency as a function of pseudo-rapidity.

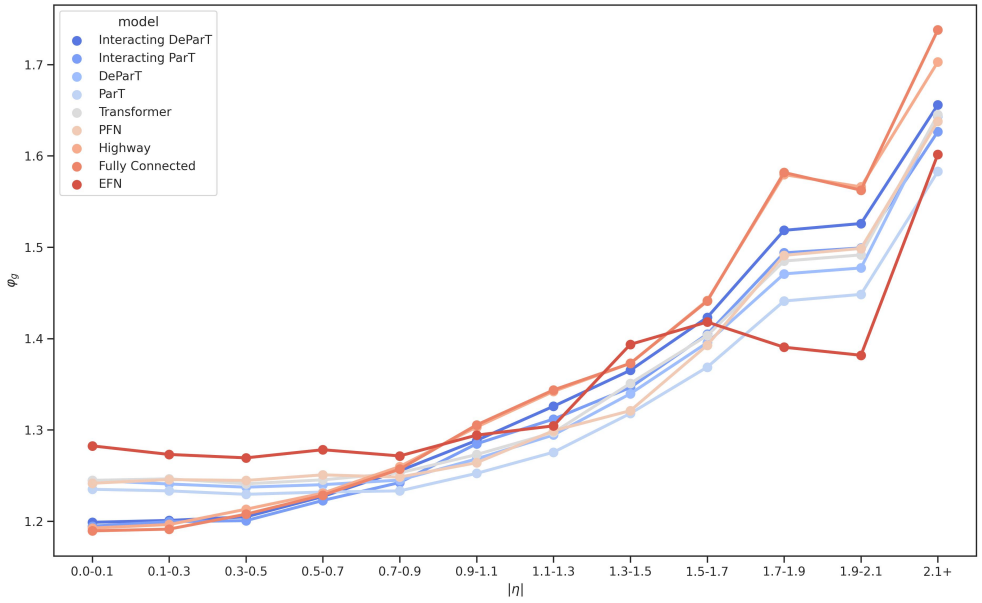


Figure C.16 Gluon rejection as a function of pseudo-rapidity.

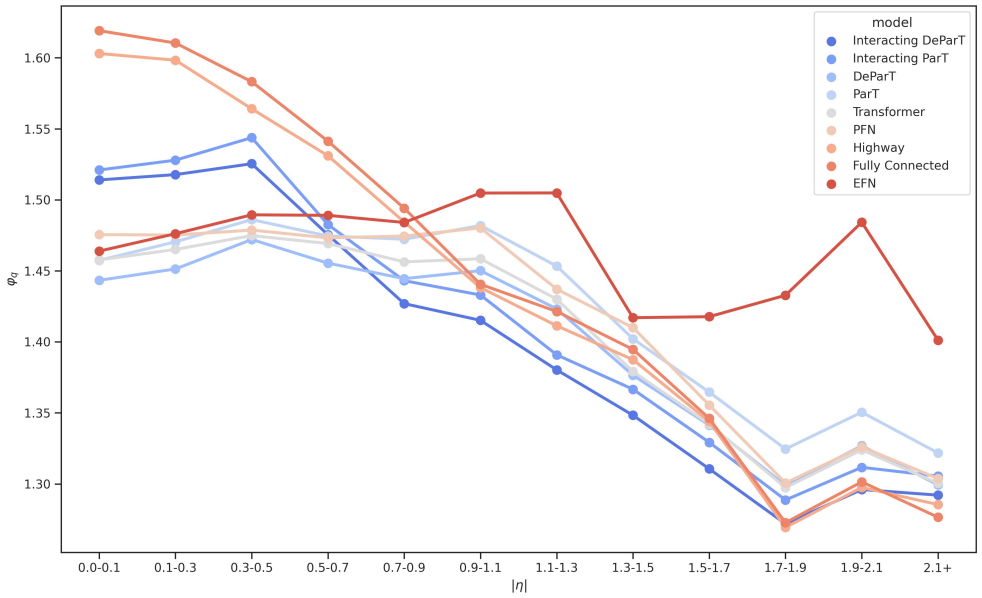


Figure C.17 Quark rejection as a function of pseudo-rapidity.

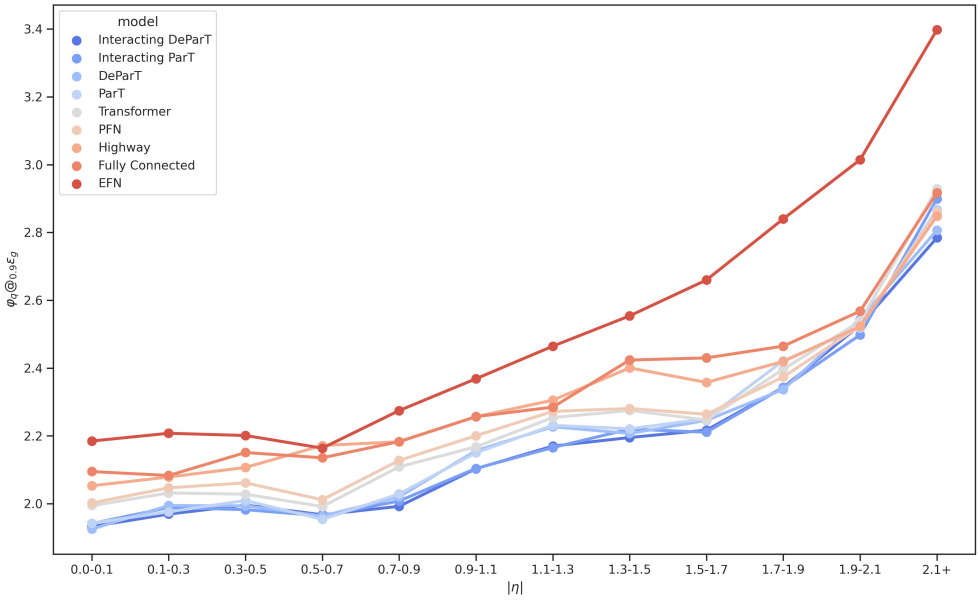


Figure C.18 Quark rejection at gluon efficiency of 0.9 as a function of pseudo-rapidity.

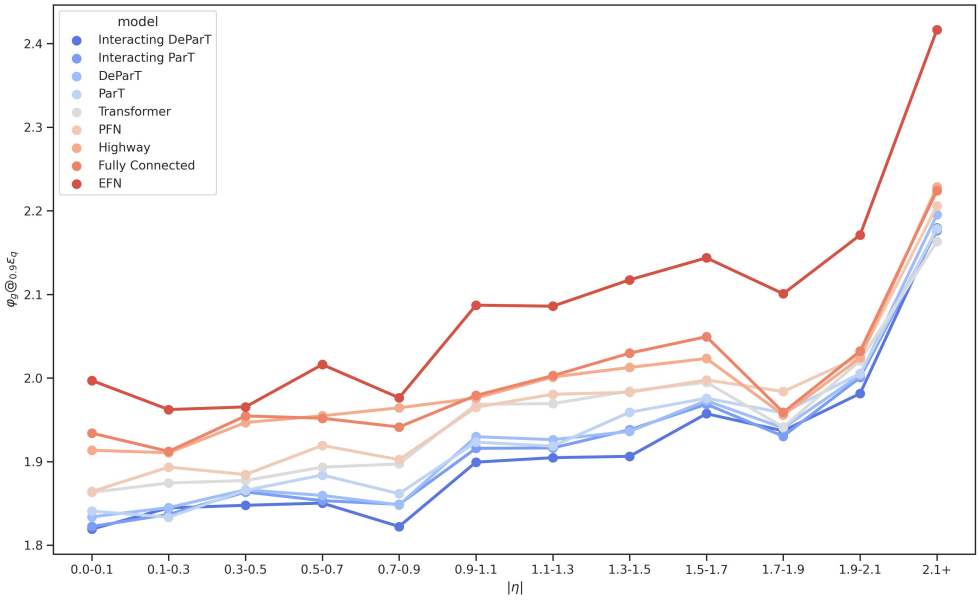


Figure C.19 Gluon rejection at quark efficiency of 0.9 as a function of pseudo-rapidity.

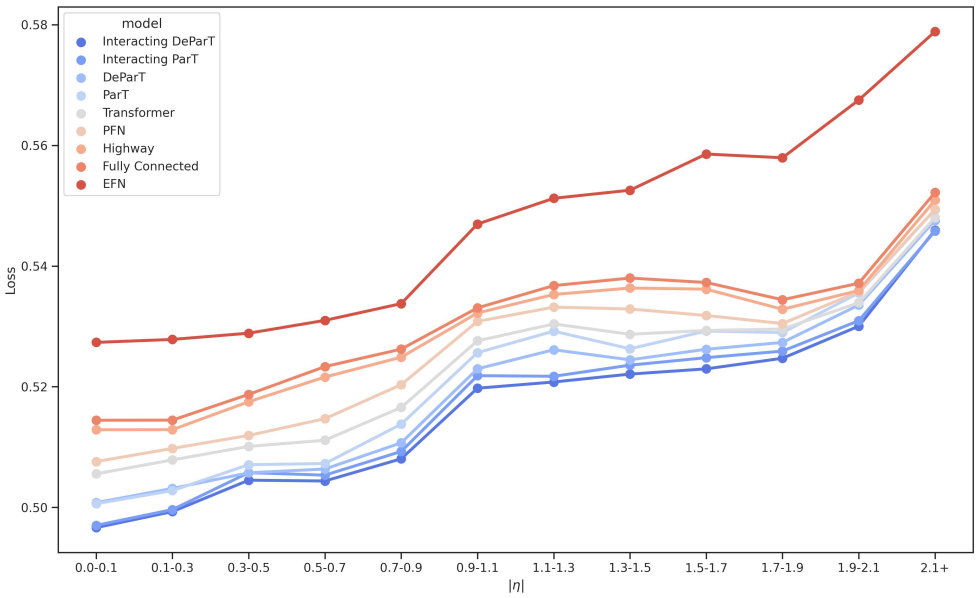


Figure C.20 Loss as a function of pseudo-rapidity.

C.5 Pileup Dependence

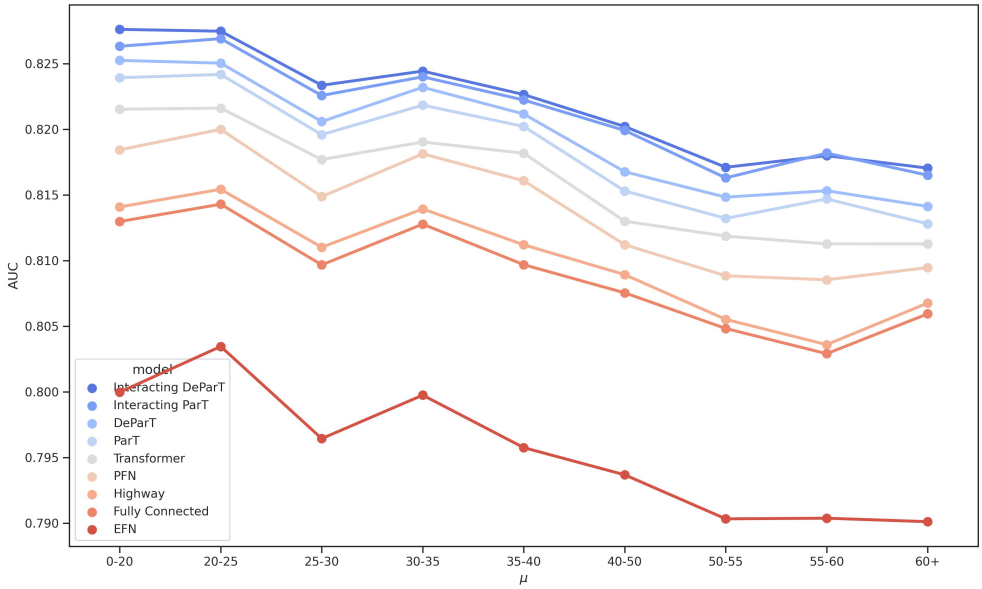


Figure C.21 AUC as a function of pileup.

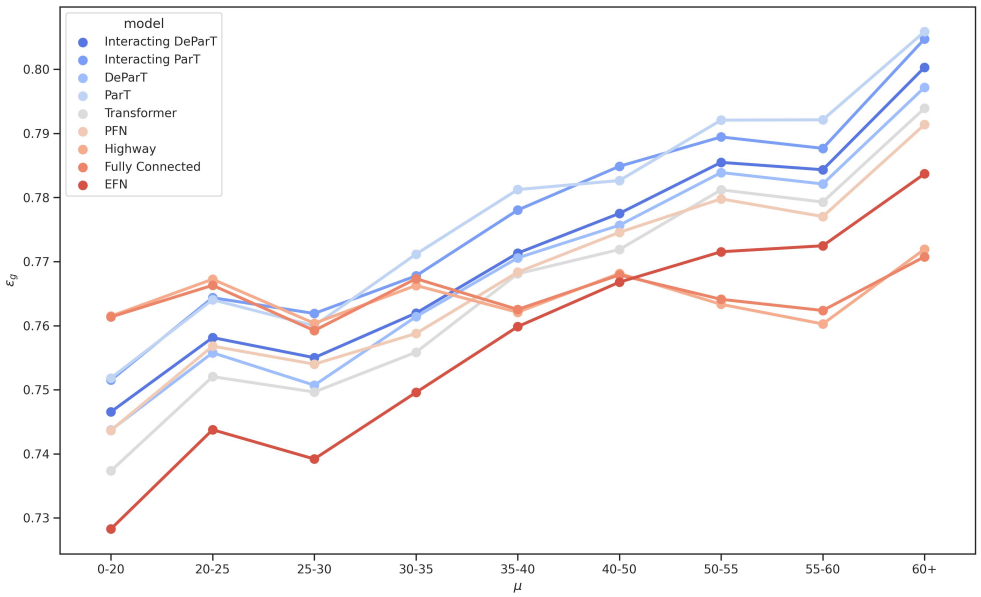


Figure C.22 Gluon efficiency as a function of pileup.

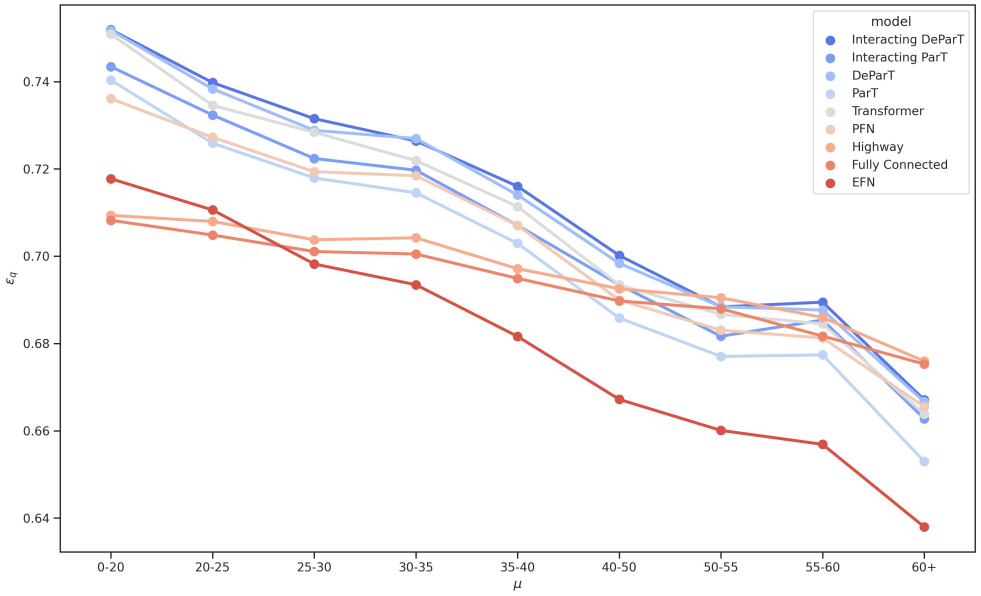


Figure C.23 Quark efficiency as a function of pileup.

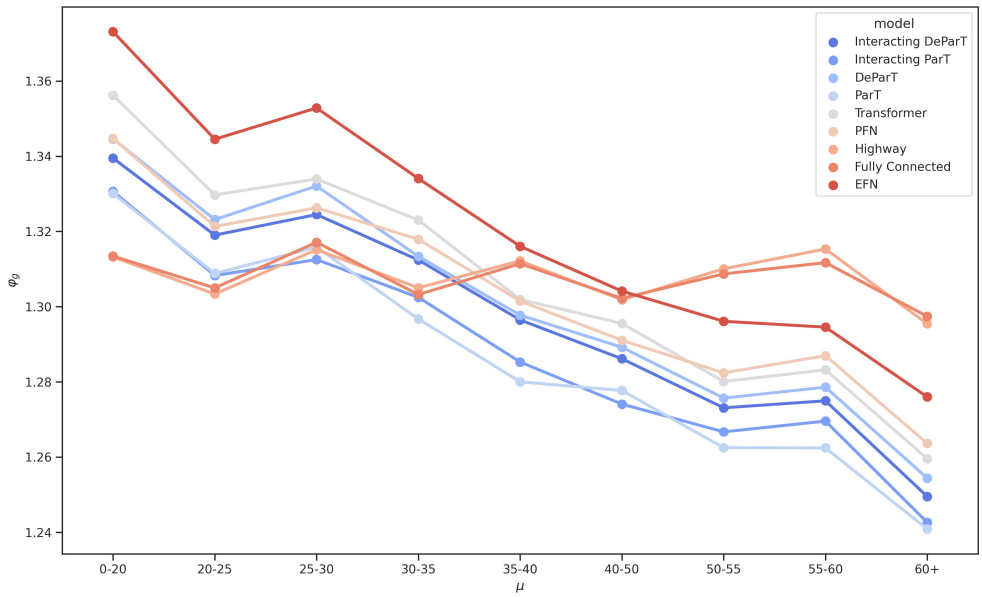


Figure C.24 Gluon rejection as a function of pileup.

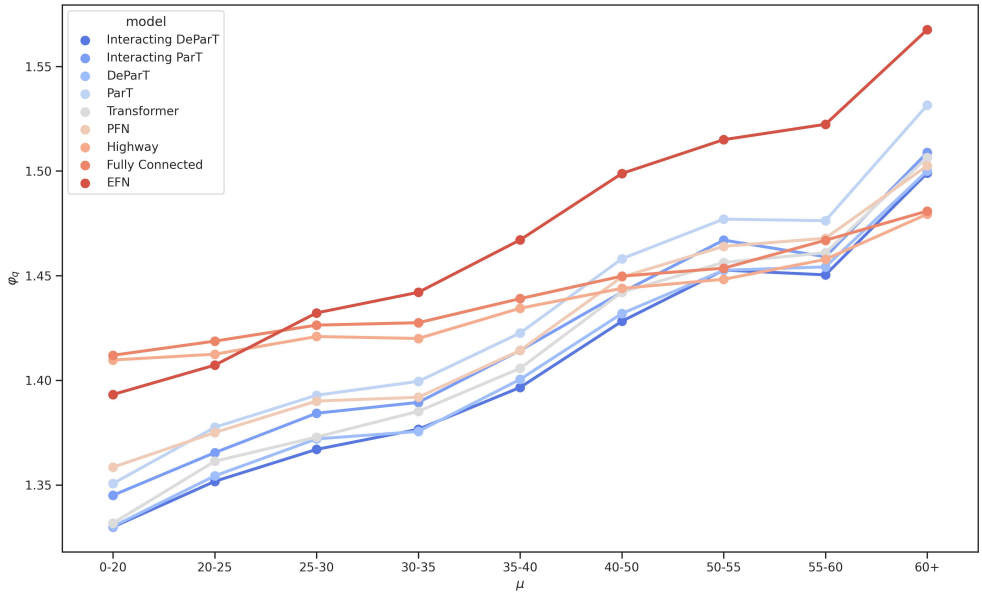


Figure C.25 Quark rejection as a function of pileup.

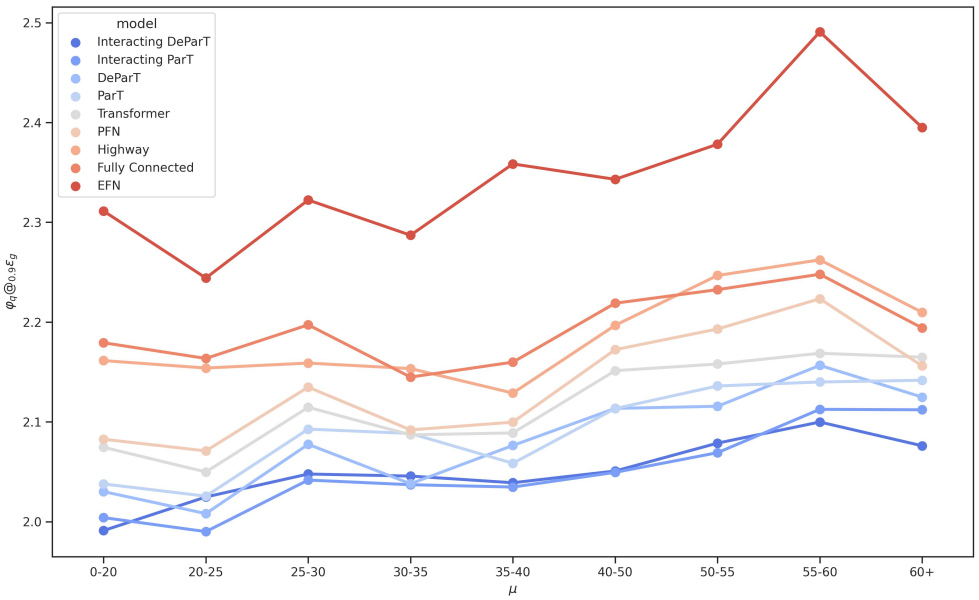


Figure C.26 Quark rejection at gluon efficiency of 0.9 as a function of pileup.

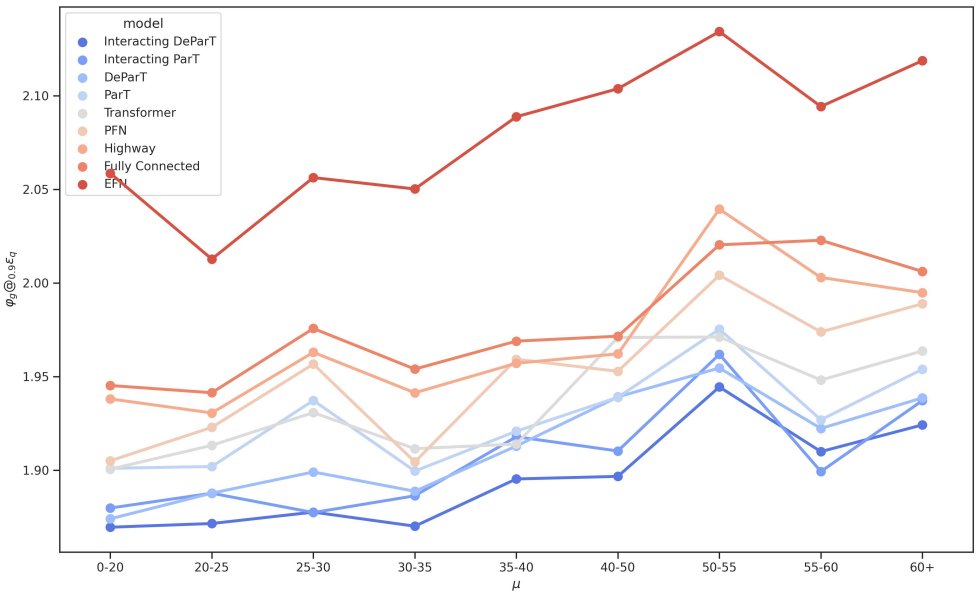


Figure C.27 Gluon rejection at quark efficiency of 0.9 as a function of pileup.

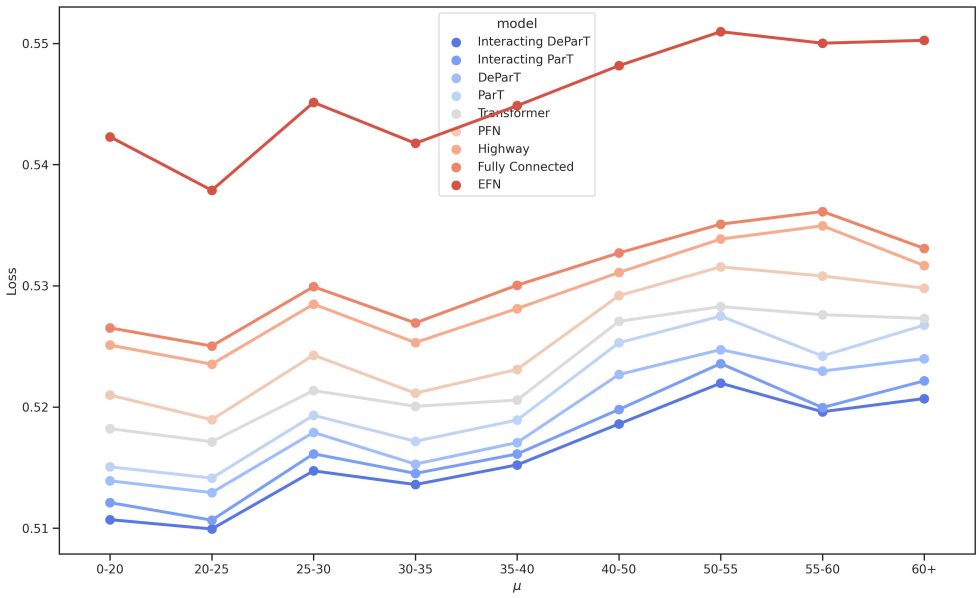


Figure C.28 Loss as a function of pileup.