Weighting for Godot: Task-Weighting Methods for the Multi-Task ATLAS MaskFormer model

HFYD0 [1]

MSc Machine Learning

Gabriel Facini

Submission date: 23 09 2024

[1]**Disclaimer:** This report is submitted as part requirement for the Machine Learning MSc at UCL. It is substantially the result of my own work except where explicitly indicated in the text. The report may be freely copied and distributed provided the source is explicitly acknowledged.

Abstract

Flavour tagging aims to distinguish particle collisions that contain heavy-flavour hadrons (b- and c-hadrons) from background events. MaskFormer is a novel multi-task learning model developed by the ATLAS Flavour Tagging team at the Large Hadron Collider (LHC). This model uses binary masks to tackle segmentation problems, and combines flavour tagging with five other tasks, such as reconstructing vertices within each jet (vertex finding), the predicting the physical process from which multiple particles originate (track origin) and regressing the properties of up to five high-momentum particles in the jet (vertex regression), such as mass and transverse momentum. In multi-task learning models, optimal learning can be hindered by tasks that have opposing gradients, or imbalanced gradient magnitudes, leading to a phenomenon called negative transfer. This thesis presents an investigation into the interactions between these tasks, as well as an implementation of 12 weighting algorithms which aim to mitigate negative transfer and boost model performance. We find that both of the mask-related tasks conflict with one another and that one of the mask-related tasks dominates the gradient update in the second half of model training. The vertex regression task is also found to be orthogonal to other tasks. It is found that weighting methods that directly optimise the task losses provide increased flavour tagging performance and the vertex finding task, while either boosting regression, or leading to slight decrease in regression performance. Weighting methods that access task gradients, on the other hand, provide even higher flavour tagging and track origin performance, however, they lead to more pronounced degradation in regression. We finally implement a modification to binary cross-entropy loss, called Poly-1 Loss, and demonstrate that it achieves significant performance gains with minimal code change.

Acknowledgements

I would like to thank my supervisor, Dr. Gabriel Facini for being incredibly generous with his time, always encouraging me to ask questions, fostering an environment where new ideas were met with enthusiasm and interest, and most importantly for all the fun chats before getting into work. I would also like to thank Nikita Pond, without whom I would not have gotten very far in this thesis, for explaining the entire Salt framework to me (many times), fixing my code bugs, and being available at odd hours to discuss ideas. I would like to thank my parents and sister for their unwavering support as I spent the summer hunched over my laptop in various corners of family weddings and holidays. Your love has carried me through the hardest times. I would like to thank Anna for always being in my corner and putting up with my nonsense, and Emma, Alina, Rob, and David for making this course a bit more bearable.

I dedicate my work to Mayyu and Tooms–your light shines on forever.

Contents

Chapter 1

Introduction

The Large Hadron Collider (LHC) has led to the discovery of the Higgs boson, expanding our understanding of the fundamental structure of matter. The LHC produces proton-proton (pp) collisions at very high energies in detectors like ATLAS. A significant technique used in the ATLAS experiment [3] is flavour tagging. This method has been crucial in detecting certain Higgs boson decay and production modes [1, 63].

Flavour tagging is the process of identifying the particle from which a jet originates. A jet is a collimated spray of particles, produced by a cascade of emissions and decays of particles in a process called hadronisation. The properties of a jet, like energy or mass, are influenced by the particle that initiated it. This information is critical for distinguishing collision events of interest from background processes, and increasing tagging performance is the only way to improve the signal-to-noise ratio without increasing the data. High background rejection is thus vital for enhancing sensitivity of measurements and studies conducted on detector data.

Common applications of flavour tagging include distinguishing quark and gluon jets [47] or tagging jets originating from heavy particles with high momentum (e.g., top quark, Higgs boson) [91]. Our focus is on identifying Higgs boson decays into pairs of bottom quarks (or rather, a bottom and anti-bottom quark), referred to as $H \to b\bar{b}$ tagging. Precise identification of these decays can significantly improve the sensitivity of searches for new physics phenomena [62] and the accuracy of Higgs boson property measurements.

Our research centers around the novel model developed by the Flavour Tagging Group (FTag) at ATLAS, which utilises state-of-the-art Mask2Former architecture [104]. This is the successor to the previous jet tagging model, GN2, which employed the use of two auxiliary tasks to aid the network in learning more robust jet representations and enhancing generalisation, resulting in improved flavour tagging performance [66]. In GN2, the auxiliary tasks were track origin prediction, which predicts the part of the jet production process that a track originates from, and vertex finding, the grouping of tracks originating from a common point.

Like GN2, MaskFormer also incorporates multiple tasks, however MaskFormer differs from GN2 by 1) following a multi-task learning (MTL) approach [109], where the aim is to improve performance across *all* tasks, not just jet classification, and 2) introducing a task which regresses the properties of vertices (vertex fitting). In combining flavour tagging with vertex finding and vertex fitting, MaskFormer has the potential to be a comprehensive tool that can enable more

precise identification of heavy-flavour particles and improve our understanding of complex event topologies.

However, one issue that can arise in MTL models like MaskFormer is a phenomenon called negative transfer or task interference [114]. This can occur when some tasks' gradients dominate or conflict with others, leading to sub-optimal learning. One approach to mitigate negative transfer is the use of sophisticated task weighting strategies, by encouraging tasks to learn at similar rates [81], reducing gradient dominance [102], and de-conflicting gradients [111, 80]. As of writing, the MaskFormer weights are static and chosen based on performance. In this work, we aim to find an appropriate task weighting algorithm which removes the need for manually tuning these weights as well as reduces any potential negative transfer between tasks.

Our main contributions are as follows: we firstly analyse task interactions during training to determine the presence of task interference in our MaskFormer model. By performing this analysis, we also aim to provide insight into the Mask2Former architecture itself, as cross-entropy loss is often combined with the Dice loss [105] in these models due to their complementary nature [61, 48]. However, to our knowledge, no studies into the gradient interactions between these two losses have yet been performed. As such, we aim to provide new insights on the Dice-CE loss, and to shed some light on why these losses work well when combined.

Following this, we implement 12 different MTL weighting algorithms to optimise the performance of our model across its jet classification, vertex finding and vertex fitting tasks. These algorithms can be broadly categorised into two groups: loss-based and gradient-based methods. Loss-based methods involve weighting the losses of individual tasks before aggregating them into a single loss term and optimising this scalar value. Gradient-based methods, on the other hand, directly modify the gradients during optimisation, and represent a more granular approach to task weighting.

We then explore the potential for incorporating new loss functions into the model given the limitations of traditional cross-entropy loss in handling imbalanced data and optimizing complex tasks. This is motivated by the development of PolyLoss [71] which can be used as a systematic framework within which new loss functions can be created. To this end, we implement the simple yet effective Poly-1 Loss which introduces one hyperparameter while showing promising performance gains in many works [44, 20].

Our final contribution is an open-source modularised implementation of all methods used in this work. Our code is available at `https://gitlab.cern.ch/atlas-flavor-tagging-tools/algorithms/salt`.

Chapter 2 provides the theoretical background necessary for understanding the problem. It provides an introduction to the Standard Model of particle physics, an overview of the ATLAS experiment and proton-proton collisions, and finally moves to b-jets and the history of flavour tagging at ATLAS.

Chapter 3 reviews the literature relevant to our work. It outlines the MaskFormer model architecture, presents the domain of multi-task learning and the factors that lead to task interference, motivating the need for appropriate task weighting methods. Lastly, it presents an overview on emerging loss functions and the PolyLoss framework.

In **Chapter 4**, we describe the methods used in our work. We first motivate the need for task-weighting algorithms after presenting a description of the ATLAS MaskFormer model. Next,

we describe the implementation of our methods, including the data, resources, and training framework. We also provide a summary of the most common flavour tagging and vertexing metrics at ATLAS.

Chapter 5 contains details of all our experiments, including results and analysis. We first analyse the task interactions in the model, then proceed to implement various different weighting methods and compare their performance.

Chapter 6 contains a summary of our contributions and our suggestions for future work.

Chapter 2

Background

2.1 The Standard Model

The Standard Model of particle physics is a highly successful theory describing the fundamental constituents of the Universe and their interactions. It classifies elementary particles into three groups: leptons, quarks, and bosons, as shown in Figure 2.1. These particles and their interactions define the core laws of nature, explaining three of the four fundamental forces: electromagnetic, strong nuclear, and weak nuclear forces. These forces are mediated by an exchange of gauge bosons.

At the atomic level, negatively charged electrons (e^-) orbit a nucleus composed of positively charged protons (p^+) and electrically neutral neutrons (n). The electromagnetic interaction, mediated by photons, binds electrons to atoms, while the strong nuclear force, mediated by gluons, binds protons and neutrons within the nucleus. The weak force, responsible for β-decay and nuclear fusion, is carried by charged W^+ and W^- bosons and the neutral Z boson.

In 1964, it was discovered that protons and neutrons are not elementary, but are composite particles formed by three quarks. Quark interactions, like the particles that they make up, are also mediated by the strong interaction. Six quark "flavours" are known: up (u), down (d), charm (c), strange (s), top (t), and bottom (b). Together with the electron and electron neutrino (ν_e), up and down quarks form the first generation of elementary particles, shown in the leftmost column of Figure 2.1. Two additional generations exist at higher energy scales, comprising heavier versions of these particles, such as the tau-lepton (τ^-). During this period, the Higgs boson was also theorised [42, 57, 51], which grants other particles their mass and occupies the unique group of scalar bosons.

Each electrically charged particle also has an antiparticle counterpart, such as positrons (e^+) and antiquarks (e.g., anti-bottom quark $\bar{b}$). Additionally, quarks and gluons carry a property called colour charge, which can be red, green or blue. According to quantum chromodynamics (QCD), which is the study of particles with colour charge, free quarks and gluons are never observed in nature. Particles with colour charge exhibit a phenomenon called colour confinement, meaning they can only be observed in bound, colour-neutral states known as hadrons.

The Standard Model's predictive power has been repeatedly confirmed through experiments, however, it remains an incomplete theory. The observed matter-antimatter asymmetry in the

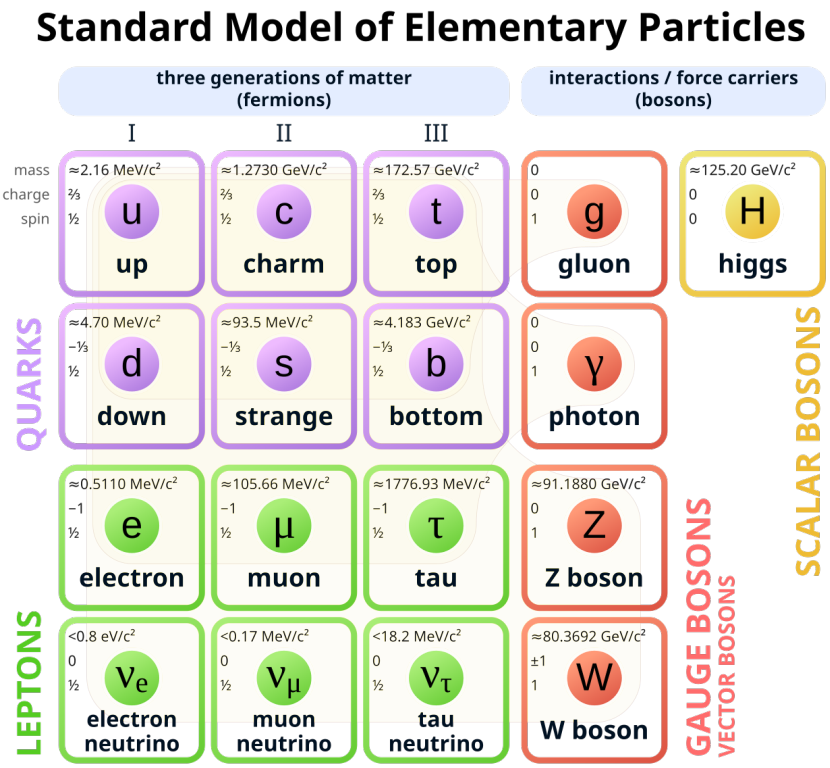


Figure 2.1: All elementary particles described by the Standard Model. Brown loops indicate which bosons couple to which fermions (quarks and leptons) [34].

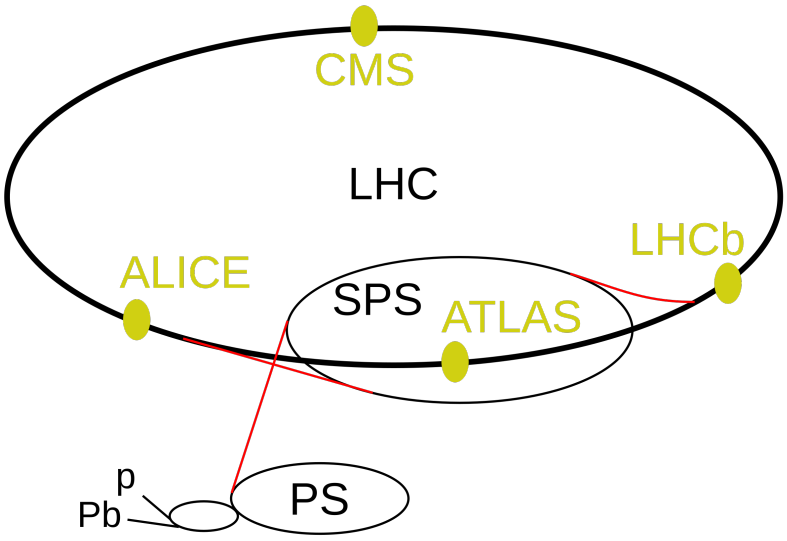


Figure 2.2: Diagram of the main LHC experiments. [58]. The path of protons begins at linear accelerators (marked p and Pb, respectively) and continue to the booster, then to the Proton Synchrotron (PS), then to the Super Proton Synchrotron (SPS) and finally into the 27-km-long LHC tunnel. The 4 largest experiments in the LHC are marked with yellow dots.

universe is as yet unexplained, posing a fundamental question about the early Universe's evolution. Moreover, the Standard Model fails to account for dark matter and dark energy, which together constitute about 95% of the universe's content. Future research in particle physics aims to address these outstanding questions and potentially uncover new fundamental principles governing the universe. For these reasons, probing physics phenomena with high precision is crucial as it can lead to a model which better explains our Universe.

2.2 The ATLAS Experiment

The European Organisation for Nuclear Research, known as CERN, has been at the forefront of particle physics research since its inception in 1954, hosting a series of increasingly powerful accelerators. Notable among these were the Proton Synchrotron in 1959, the Super Proton Synchrotron in 1976, and the Large Electron-Positron Collider which operated from 1989 to 2000 [14, 107]. These machines facilitated numerous groundbreaking discoveries, including the observation of neutral currents in 1973 [52, 85] leading to the discovery of W and Z bosons in 1983 [9, 39].

The Large Hadron Collider (LHC) at CERN is the most powerful accelerator in the world. With a 26.7 km circumference superconducting proton-proton (pp) ring, the LHC is designed to reach extremely high energies of up to 14 TeV in the center-of-mass frame [3]. The LHC's primary goals include exploring the fundamental nature of matter and energy, searching for physics beyond the Standard Model (BSM), and investigating the properties of the strong interaction at extreme energy densities. The path of particles within the LHC is shown in Figure 2.2.

The LHC operates on a cyclical schedule of runs followed by long shutdowns for maintenance

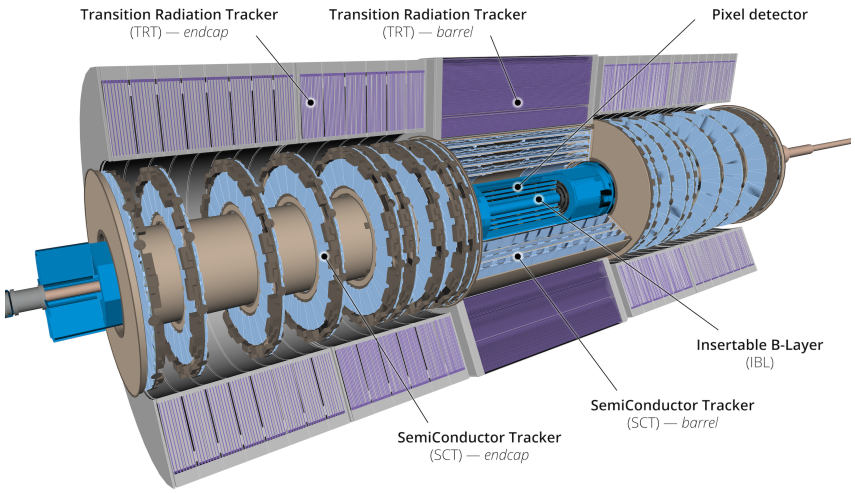


Figure 2.3: 3D Cut-away view of the ATLAS Inner Detector [33], which consists of a silicon pixel detector at the innermost layer with an insertable B-layer at its core, surrounded by a semiconductor tracker and a straw-tube detector, the TRT.

and upgrades. Run 1 spanned from 2009 to 2013, achieving collisions at 7-8 TeV center-of-mass energy and culminating in the discovery of the Higgs boson. Run 2, from 2015 to 2018, saw energy increased to 13 TeV. The LHC is currently in Run 3 (2022-2025), operating at 13.6 TeV.

The ATLAS (A Toroidal LHC ApparatuS) experiment is one of two general-purpose detectors at the LHC, alongside CMS. ATLAS was proposed in 1994 and officially approved in 1996, with construction completed in 2008 [3]. The detector measures 46 m long, 25 m high, and weighs approximately 7,000 tonnes. It consists of several subsystems, including an inner detector for tracking charged particles, electromagnetic and hadronic calorimeters for energy measurements, and a muon spectrometer.

ATLAS has been instrumental in numerous significant discoveries and measurements. Most notably, it played a crucial role, along with CMS, in the discovery of the Higgs boson in 2012, leading to the Nobel Prize in Physics in 2013 for François Englert and Peter Higgs [2]. Beyond the Higgs discovery, ATLAS has contributed to precision measurements of Standard Model parameters, searches for supersymmetry and studies of other BSM phenomena [27].

At its core is the Inner Detector (ID), comprising three main components: the Pixel Detector, the Semiconductor Tracker (SCT), and the Transition Radiation Tracker (TRT). This is illustrated in Figure 2.3. The Inner Detector is highly compact, consisting of four layers of silicon pixels, each pixel smaller than a grain of sand. This detector lies 33 mm from the beam line, allowing researchers to determine the origin and momentum of particles with high precision. The innermost layer of silicon pixels–called the Insertable B-layer (IBL)–was installed between Run 1 and Run 2. Both the pixel hits and SCT measure the location of charged particles, which are then reconstructed as tracks. Lastly, the TRT is made up of straws filled with gas mixture that gets ionised as charged particles cross through the straws. Recording measurements at a large radius aids in more precise momentum measurements. As charged particles travel through the Inner Detector, they leave energy deposits, or "hits", which are used to reconstruct the particles'

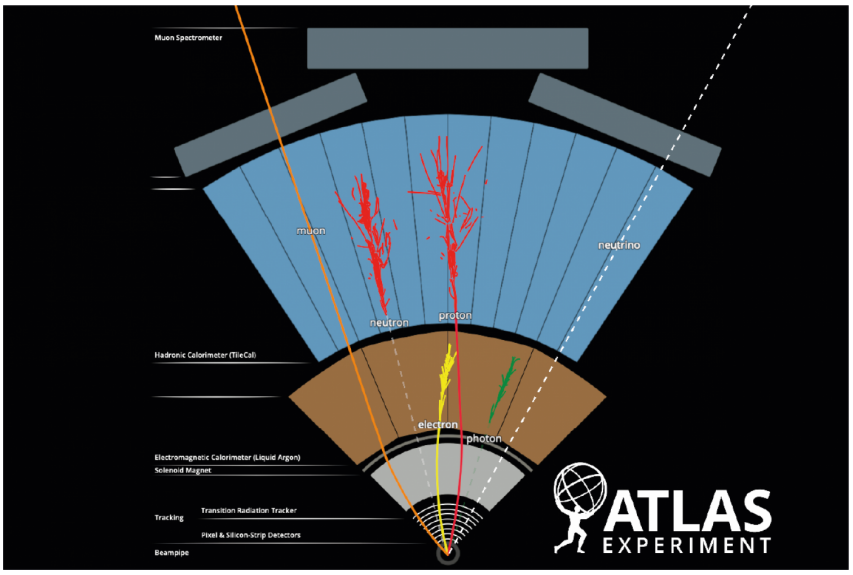


Figure 2.4: 2D schematic view of all layers of the ATLAS detector, with different types of particles shown at the points where they are measured within the detector [35].

trajectories, known as tracks.

Surrounding the ID are the calorimeter systems: the Liquid Argon electromagnetic calorimeter and the hadronic calorimeter. These measure the energies of particles passing through by absorbing them completely. The outermost layer is the Muon Spectrometer, which identifies and measures the momenta of muons, particles that typically pass through the inner layers due to their high mass. The entire detector is permeated by a 2T magnetic field generated by a system of superconducting magnets, which bends the paths of charged particles, allowing for momentum measurements. This layered structure enables ATLAS to identify and measure a wide range of particles with high precision, essential for studying the complex outcomes of proton-proton collisions. A 2D cross-section of the entire ATLAS detector is shown in Figure 2.3, including the ID, calorimeters and muon spectrometer.

2.2.1 Particle Collisions

Proton-proton (pp) collisions at the LHC are governed by the principles of quantum chromodynamics (QCD), which is the study of the strong interaction between quarks and is mediated by gluons. The parton model, developed by Feynman and Bjorken [45], provides a framework for understanding these collisions.

In this model, protons contain a "sea" of quarks, antiquarks and gluons (collectively called partons). The majority of a proton's mass is due to QCD energy, including the kinetic energy of the quarks and the binding energy of the gluon fields. The three quarks that make up the proton (two *up*, one *down*) are the valence quarks, which are more energetically stable than the sea quarks. Collisions occurs between the partons rather than the protons as a whole. As a consequence, the parton distribution functions (PDFs), which describe the probability of finding a parton with a given momentum fraction within the proton, are not precisely known and must be determined experimentally.

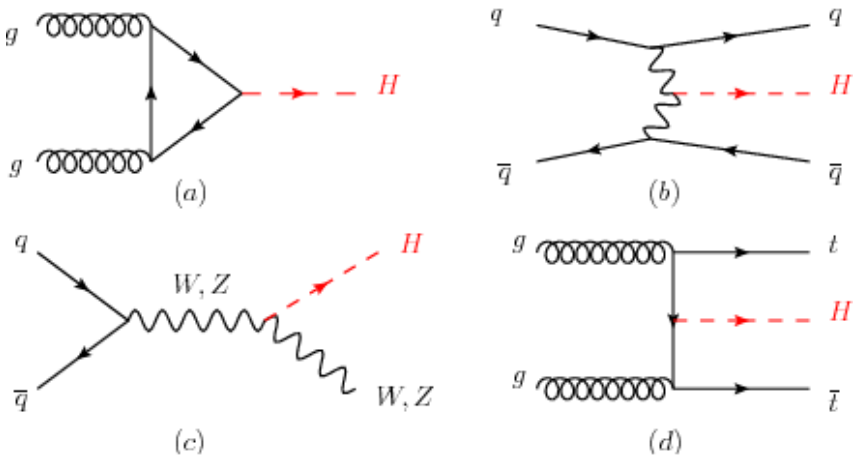


Figure 2.5: Feynman diagrams for the four most common Higgs production modes [50], with a) gluon-gluon fusion, (b) vector-boson fusion, (c) Higgs-strahlung (or associated production with a gauge boson) and (d) associated production with top quarks.

The probability of a given physical process occurring is called the cross-section, and is calculated using the experimental PDF. Secondly, due to color confinement, the partons from a collision must undergo hadronisation, forming jets of particles. Additionally, the strong coupling constant of QCD varies with energy scale (a property known as asymptotic freedom), making calculations challenging at low energies. Lastly, multiple proton-proton interactions occur in a single bunch crossing (where two bunches of protons collide) at the LHC, leading to a phenomenon called pileup. These factors create a complex environment that requires sophisticated modelling to interpret experimental data accurately.

Monte Carlo (MC) event generators are used to simulate complex QCD processes, providing important truth information that can be used to train large machine learning models. The most widely used generators for LHC physics include Pythia, Herwig, and Sherpa [15]. MC generators also account for the underlying event, simulating the additional soft interactions between proton remnants that occur alongside the main hard scattering process.

Proton-proton (pp) collisions are conducted for several reasons, one of which is to observe properties of the Higgs boson. The dominant Higgs production mode is gluon-gluon fusion ($gg \to H$), accounting for almost 87% of the cross-section, where two gluons from the colliding protons fuse to create a Higgs boson, often through a top quark loop, as seen in Figure 2.5. Vector boson fusion ($qq \to qqH$) is the second most common process, involving the fusion of two vector bosons (W or Z) emitted by quarks in the protons, followed by Higgs-strahlung or associated production with a vector boson ($q\bar{q} \to VH$, where $V = W$or Z). The last mode, associated production with top quark pairs ($gg \to t\bar{t}H$) comprises only 1% of all Higgs production events.

2.2.2 B-jets

The Higgs boson has a very short lifetime of 1.6×10^{-22} s before decaying, too short to measure the particle directly. On the other hand, weakly decaying heavy-flavour hadrons have non-negligible lifetimes, such as 1.5 ps for b-hadrons; this is roughly 10^{10} times greater than the lifetime of the Higgs. As the dominant decay mode (occurring approximately 58% of the time) for the Higgs

boson is to a pair of bottom (b) quarks, $H \to b\bar{b}$, the study of heavy-flavour b-quarks is crucially important in understanding the Higgs particle.

Heavy-flavour hadrons are generally produced in what are called *hard interactions*, the primary collision in an event. Bottom quarks are either created in the initial interaction, or top (t) quarks or the Higgs boson (H) are created first, then decay directly to b-quarks ($t \to bW$, $H \to b\bar{b}$). The resulting b-quarks hadronise into b-hadrons, which collimate to form jets; this is why the jets that result from b-quark hadronisation are called *b-jets*. This naming convention also holds true for *c-jets*. Furthermore, b-hadrons commonly decay to c-hadrons, leading to additional vertices within b-jets as a result of these secondary $b \to c$ decays.

The b-quark from the hard process often carries a substantial fraction of the jet energy, and the decay vertex of the resulting b-hadron is significantly displaced from the primary vertex, which is a consequence of its significant lifetime. This location is called the secondary vertex (SV), and the transverse displacement between the primary vertex and secondary vertex is denoted by L_{xy}. After the decay products from the b-hadron move through the detector, each of these particles leaves behind a track. These tracks are lines that indicate the paths the particles from the decay (and earlier hadronisation) followed. These tracks typically show a large displacement from the primary vertex if they originated from a b-jet, which is measured by the impact parameter d_0, defined as the minimum distance of the track from the primary vertex. Conversely, particles from lighter jets typically decay right at the primary vertex, meaning their tracks have small or negligible impact parameters, making them easier to distinguish from the longer-lived b-hadrons. The process of b-jet production from a hard interaction is illustrated in Figure 2.2.2.

The difference between azimuthal angle in the transverse plane (perpendicular to the beam axis) $d_\phi = \phi_i - \phi_2$ and the pseudorapidity (the angle of a particle relative to the beam axis) for two particles $d_\eta = \eta_1 - \eta_2$ are used to calculate angular distances from the jet axis dR, which is measured as $\sqrt{d_\phi^2 + d_\eta^2}$. A particle is considered to be associated to a jet if it is within dR of that jet, in a process called geometrical matching. At high transverse momentum $p_\mathbf{T}$, which is the component of a particle's momentum perpendicular to the beam axis, the jet becomes more highly collimated, leading to increased difficulty in distinguishing decay products.

The second type of pp collisions are *soft interactions* (low p_T), which are known as pile-up. Heavy-flavour jets typically contain one to three secondary vertices, while light-jets typically contain fewer displaced vertices with a lower mass.

2.3 Jet Tagging Algorithms

Jet tagging, particularly the identification of jets originating from heavy-flavour quarks, has been a crucial aspect of high-energy physics experiments for decades. As early as 1990, Lönnblad et al. proposed a neural network to separate gluon and quark jets [82].

At ATLAS, jet tagging algorithms have been developed to classify jets originating from b- and c-quarks. Early approaches relied on impact parameter-based taggers and secondary vertex reconstruction. The incorporation of machine learning techniques marked a significant advancement in jet tagging performance. Initially, shallow learning methods such as artificial neural networks and boosted decision trees were employed. These algorithms combined various low-level features derived from track and vertex information to improve discrimination power.

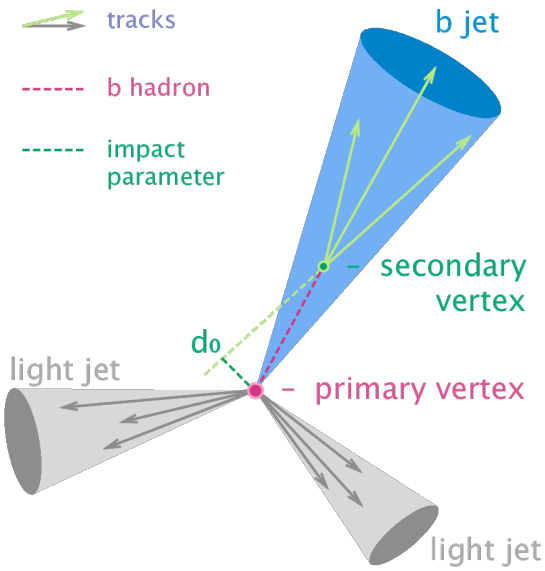


Figure 2.6: Diagram that demonstrates b-jet production from a primary pp collision. b-quarks are produced in the initial interaction and travel a measurable distance before decaying again at the secondary vertex. The resulting b-jet is depicted in blue. Light jets are also produced in the primary interaction (grey). The impact parameter d_0 is the shortest distance between the closest approach of a track and the primary vertex. The impact parameter shown here is for the rightmost track in the b-jet.

A paradigm shift occurred with the adoption of deep learning techniques. In 2018, ATLAS introduced DL1, a deep neural network-based b-tagging algorithm that significantly outperformed previous methods, however this tagger still relied on manually optimised low-level algorithms. Recent years have seen the use of graph neural networks (GNNs) for jet tagging, with the low-level algorithms being replaced by auxiliary tasks to improve performance.

2.3.1 Low-Level Algorithms

Low-level jet tagging algorithms can be grouped into three broad categories: Impact Parameter (IP) based and Secondary Vertex (SV) based.

IP-based taggers exploit the large impact parameters d_0 of tracks originating from b-jets compared to light-jets, by computing the likelihood of a given track associated with a jet originating at the primary vertex [25]. Two such taggers, IP2D and IP3D, calculate discriminants from log-likelihood ratios, using flavour hypotheses for b-, c-, and light-jets (p_b, p_c, and p_l, respectively) computed from summed track contributions extracted from simulations.

Recurrent Neural Network Impact Parameter (RNNIP) [28] is a deep learning approach which feeds sequences of track impact parameters and other track properties into a Recurrent Neural Network, capturing spatial and kinematic signatures of b-jets. The Deep Impact Parameter Sets (DIPS) [31] algorithm builds upon RNNIP by implementing the Deep Sets architecture [115] which is physically motivated by its permutation invariance for track ordering, leading to a significant decrease in training time.

SV-based taggers look for displaced vertices. SV1 is a secondary vertex reconstruction algo-

rithm which uses a likelihood ratio test to discriminate between light and b-jets [54]. SV1 focuses on reconstructing a single displaced inclusive secondary vertex within a jet by grouping together b-hadron decay products, including tracks from b-hadron decays and BC ($b \to c$) decays. The reconstruction process begins by identifying potential two-track vertices, which are formed using all tracks that are associated with the jet. The algorithm then operates with iterative refinement, where each iteration attempts to fit and accurately reconstruct a single displaced secondary vertex, representing the point where the secondary particle decay occurs within the jet.

JetFitter [30] aims to explicitly reconstruct the complete b-hadron decay chain within a jet. This algorithm capitalises on the characteristic structure of weak b- and c-hadron decays, utilising a Kalman filter to identify a line which lies between the primary vertex, secondary vertex, and tertiary vertex. Unlike SV1, JetFitter does not group together b vertices with $b \to c$ vertices, but they are instead reconstructed separately. The algorithm processes six variables that encapsulate both decay topology and vertex information, feeding them into a neural network with three output nodes corresponding to b-, c-, and light-jet hypotheses (p_b, p_c, and p_l, respectively).

2.3.2 High-Level Algorithms

MV1 is a neural network that utilises the outputs of IP3D, SV1 and JetFitter as inputs [26]. Its successor, MV2, is a gradient boosted decision tree that performs a binary classification of jets into b and non-b [29], and outperformed previous methods. Unlike MV1, MV2 directly takes as input the variables employed as inputs to the low-level taggers.

DL1 represented the state-of-the-art in b-tagging for the ATLAS experiment during Run 2 of the LHC [69]. This model is a neural network that incorporates inputs from multiple low-level taggers, such as IP-based taggers (IPxD), SV1, and JetFitter. Variants of DL1 also leveraged outputs from RnnIP or DIPS as inputs, namely in DL1r and DL1d, respectively, with the latter achieving the highest performance of the three DL1 models.

The first algorithm to eliminate the use of low-level algorithms is GN1 [32], a multimodal multitask model, utilising a graph neural network architecture. It processes up to 40 tracks per jet, with corresponding track and impact parameters, along with jet p_{T} and η. The model outputs three jet class probabilities, for b-jets, c-jetsand light-jets. GN1 incorporates two auxiliary training objectives: track origin classification and vertex classification (grouping tracks into vertices), which replace the previous low-level algorithms. Training data comprises 30M jets from $t\bar{t}$(low p_{T}) and $Z \to q\bar{q}$(high p_{T}) samples. GN1 shows significant performance gains over DL1d, particularly in background rejection for both low and high p_{T} jets.

GN2 builds upon GN1 with several enhancements, primarily its size, through the incorporation of layer normalisation and dropout for training stability. With 2.6 million parameters, GN2 is more than three times larger than GN1, and 20 times larger than DL1d. It uses an improved learning rate optimisation with a learning rate scheduler and employs a new, more efficient framework, enabling high-statistics training with 192M jets, a substantial increase from GN1's 30M. The attention mechanism is also updated from a Graph Attention Network to a Transformer architecture. Like GN1, GN2 processes jet and track variables, outputting probabilities for different jet origins. GN2 demonstrates further performance improvements over both DL1d and GN1.

The model architecture consists of a per-track initialiser network, followed by a Transformer

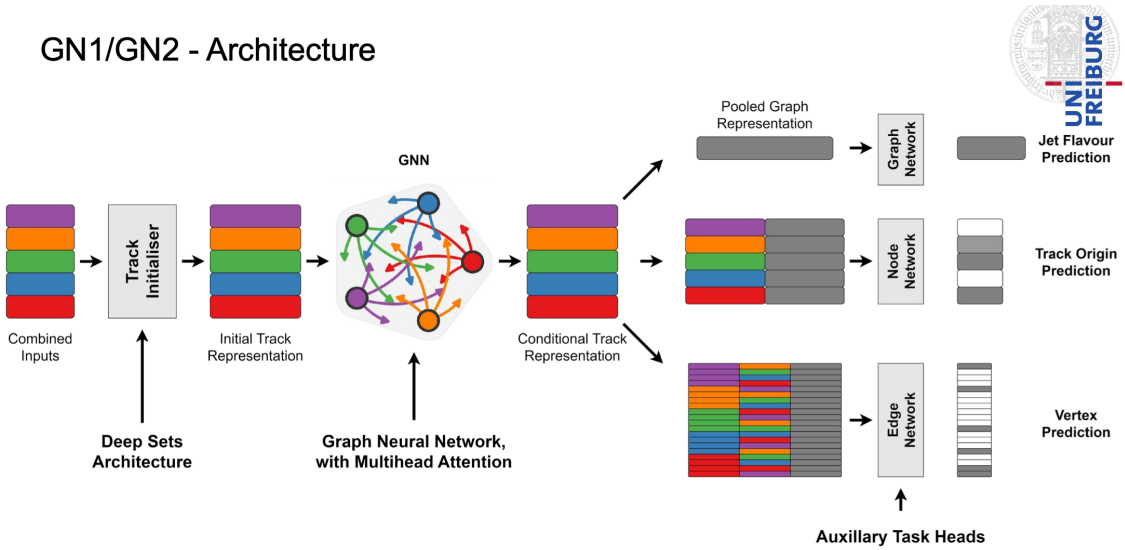


Figure 2.7: GN1 architecture [32]. Inputs are fed into an initialisation network, which outputs an embedding for each track. The node features of a fully connected graph network with attention are then populated with these embeddings. Lastly, the resulting node representations are used to predict the three outputs (jet classification, track origin and vertexing).

Encoder with 6 encoder blocks and 4 attention heads. GN2 forms a global jet representation using a weighted sum over track representations, with attention weights learned during training. GN2 utilises the same auxiliary tasks as GN1, with each auxiliary task using 3 hidden layers (128, 64, 32 neurons). GN2X is the same as GN2, but is specifically designed to discriminate jets from boosted Higgs decays ($H \to b\bar{b}$ and $H \to c\bar{c}$) against backgrounds from multijet processes and hadronic top-quark decays.

For more detailed descriptions on ATLAS jet flavour tagging algorithms, see here [70] and here [89].

Chapter 3

Related Work

3.1 MaskFormer

The latest addition to the series of jet tagging algorithms discussed in 2.3 is the ATLAS MaskFormer model, which represents a groundbreaking achievement in combining jet classification, vertexing finding and vertex fitting under one framework, and will be studied in this work.

This model is based on Mask2Former [23], the successor to Segment Anything [67] (MaskFormer), a segmentation model released in 2023. This model represents a significant advancement in image segmentation by unifying the three main image segmentation tasks under a single framework. Semantic segmentation assigns a category label to each pixel, instance segmentation detects and delineates each distinct instance of an object, and panoptic segmentation encompasses the two. Prior to MaskFormer, these tasks were typically approached with separate, specialised models, such as UNet [95] and SegNet [8] for semantic segmentation and Yolact for instance segmentation tasks [13].

The architecture of Mask2Former reframes segmentation as a mask classification problem, bridging the gap between different segmentation paradigms. This architecture takes images as input and returns detected objects. The detected objects are outputted as class labels for semantic segmentation, and binary pixel masks for instance segmentation. This model combines these two segmentation types by assigning different masks and a common class label for multiple objects of a single category.

The model consists of three main components: a pixel-level encoder, a Transformer decoder, and a segmentation head. The backbone extracts features from the the input image $\mathbf{I} \in \mathbb{R}^{C \times H \times W}$, where C is the number of input channels (3 for RGB images), and H, W are the image dimensions. The multi-scale features generated by the backbone are processed through an image feature encoder (typically a convolutional neural network), which enhances and refines these features. In Mask2Former, this refinement is performed by a transformer architecture that incorporates Deformable Attention.

Next, multi-scale features are upsampled using a mask generator, producing pixel embeddings represented as matrices of dimension $\varepsilon \times H \times W$. Each mask corresponds to an object, an image region, or "nothing," with ε being the embedding size for each mask. To decode each mask and predict its class, a transformer decoder with trainable query embeddings is used. This decoder

processes both the queries and the multi-scale features (which are iteratively refined from the encoder) to output the final mask embeddings (E_{mask}) with dimensions $N \times \varepsilon$, where N represents the number of predefined queries. A dot product is performed between the mask embeddings and pixel embeddings to produce binary masks of size $N \times H \times W$. To classify each mask, a classifier is applied to the mask embeddings, resulting in the final class probability scores for each mask, with dimension $N \times (C + 1)$.

Mask2Former uses a Hungarian matching algorithm to match object queries to the ground truth objects (or to a NULL/OTHER class) by computing the Hungarian assignment between ground truth boxes and model predictions, as in the the DEtection TRansfomer (DETR) model [17]. More specifically, the Hungarian matcher minimises a matching loss that constructs the cost matrix for bipartite matching. The cost function for the Mask2Former matching algorithm combines cross-entropy loss and the Dice [105] loss:

$$L_{\text{mask}} = \lambda_{\text{CE}} L_{\text{CE}} + \lambda_{\text{Dice}} L_{\text{Dice}}. \tag{3.1}$$

where λ_{CE} and λ_{Dice} are hyperparameters which are both set to 5.0. Auxiliary losses can also be used to compute the Hungarian matching cost, by adding them to L_{mask}. More information on the Hungarian method can be found here [17]. This cost function is later combined with the classification loss to produce a final loss as follows:

$$L_{\text{Total}} = L_{\text{mask}} + \lambda_{\text{class}} L_{\text{class}} \tag{3.2}$$

where λ_{class} is a hyperparameter set as 2.0 for objects with a corresponding ground truth, and 0.1 for NULL/OTHER objects. While Mask2Former adopts the same architecture as MaskFormer, the key difference between the two models its the incorporation of a masked attention (MaskAttention) operator in the Transformer decoder, which sparsifies the attention matrix to allow the model to attend only to regions that are important for predicting a given mask.

3.2 Multi-Task Learning

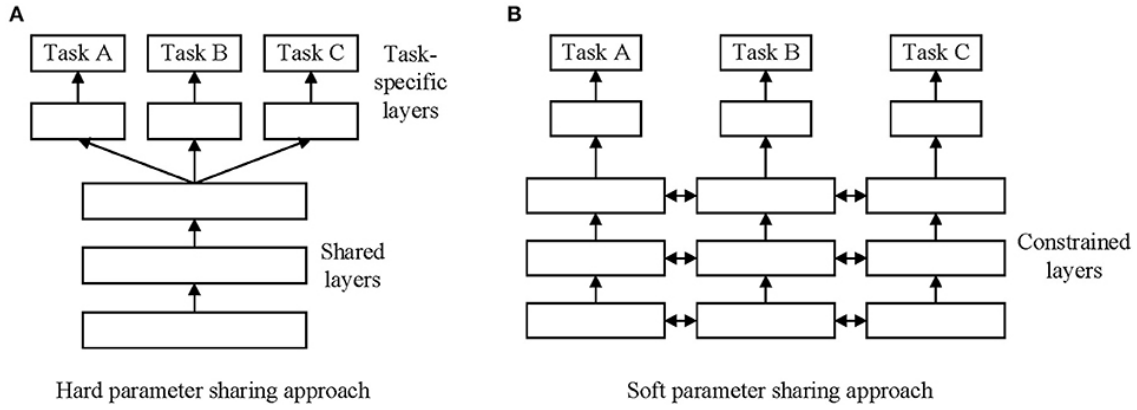


Figure 3.1: Illustration of hard parameter sharing (a) and soft parameter sharing (b) [37].

Multi-task learning (MTL) is a sub-field of machine learning in which multiple related tasks are

learned by a single model to leverage useful information between them [18]. This is in contrast with single-task learning (STL) which only learns a single task. MTL has been studied extensively in the fields of machine vision [12], bioinformatics [112, 116], speech [36], reinforcement learning [108, 16], and natural language processing [99, 19]. MTL is considered an attractive paradigm because it reduces the need for multiple single-task models, reducing the overall number of parameters that would be needed for each individual task. Secondly, it has been shown that well-chosen tasks can work synergistically to generalise better to unseen data, often with comparable or better performance to single-task learning (STL) and with far less data per-task [98].

Concretely, MTL optimises multiple objectives by combining the loss functions of each task. The simplest objective function for a K-task MTL model is the sum of the task losses:

$$L_{\text{Total}} = L_1 + L_2 + \cdots + L_K \tag{3.3}$$

An importance design choice in MTL models is how to manage the sharing of parameters across tasks. This has led to the development of two primary paradigms: hard parameter sharing and soft parameter sharing, which are shown Figure 3.1. In hard parameter sharing, first introduced in early neural network-based multitask models [18], the lower layers of the model (often referred to as the "backbone") are shared across all tasks, while task-specific heads are built on top of these shared layers. These task-specific layers only deal with the output or final task-specific processing, while the shared layers learn common embeddings across all tasks.

This approach forces the model to learn features that are useful for all tasks, thereby reducing the risk of overfitting, and [11] demonstrated that the likelihood of overfitting shared parameters decreases by a factor of K (number of tasks), compared to overfitting the task-specific parameters, such as the output layers. Hard parameter sharing is also computationally efficient, as multiple tasks are processed with a single forward and backward pass through the shared layers. The most popular hard-parameter sharing MTL architecture is the Multi-Task Attention Network (MTAN) [81], which contains a shared network to learn a feature pool across tasks, and K task-specific attention networks which learn task-specific information. The MaskFormer model used in this research implements a hard parameter sharing approach.

However, a major limitation of hard parameter sharing is its inability to handle tasks that are not strongly correlated. If some tasks require significantly different features, the model may struggle to find a shared representation that works well for all tasks. To address this, soft parameter sharing was proposed, where each task has its own set of model parameters and is trained independently. Cross-stitch networks, for example, start out with a soft parameter approach, ie. with independent model architectures, then use "cross-stitch units" to allow dynamic feature sharing across tasks [84].

Another important choice for multi-task learning is model architecture, which can be broadly categorised into encoder-focused and decoder-focused architectures. Encoder-focused methods share information during the encoding stage, enabling feature representation across multiple tasks, with task-specific heads responsible for predicting individual outputs. On the other hand, decoder-focused architectures promote information sharing between task-specific layers. The MaskFormer model studied here lies somewhere between the two, where some tasks pass through the decoder and others pass through dense task-specific heads for output prediction from the encoder directly.

3.3 Task Weighting Methods

In multi-task learning, there are three conditions which can pose significant challenges to model performance. The first is conflicting task gradients, indicated by a negative inner product:

$$\phi_{ij} = \frac{G_i^\top G_j}{\|G_i\|\|G_j\|} < 0 \tag{3.4}$$

where ϕ_{ij} is the cosine similarity between task i and task j.

The second factor is task dominance, where if one task's gradient significantly exceeds the other in magnitude, it will dominate the overall gradient. Gradient magnitude similarity $\mathcal{S}$ is defined as follows:

$$\mathcal{S}(G_i, G_j) = \frac{2\|G_i\|_2\|G_j\|_2}{\|G_i\|_2^2 + \|G_j\|_2^2}. \tag{3.5}$$

where G_i is the gradient of the i-th task's loss with respect to the model parameters $\nabla_\theta L_i(\theta)$. When the magnitude of two gradients are equal, this value is 1. As the gradient magnitudes become increasingly different, $\mathcal{S} \to 0$.

The final factor is high positive curvature along the dominating task gradient direction, leading to overestimation of its improvement on the overall learning objective.

$$\mathcal{H}(L; \theta^{(t)}, \theta^{(t+1)}) = \int_0^1 \nabla L(\theta^{(t)})^\top \nabla^2 L(\theta^{(t)} + \alpha(\theta^{(t+1)} - \theta^{(t)})) \nabla L(\theta^{(t)})\, d\alpha > C \tag{3.6}$$

which is the averaged curvature of L between θ and θ' in the direction of the multi-task gradient $\nabla L(\theta)$, and C is a large positive constant.

These factors are not always inherently an issue, but they become problematic when a subset of tasks consistently suffers from sub-optimal learning as a result [79], a phenomenon called negative transfer or task interference. When combined, these three factors can constitute a "tragic triad" and cause negative transfer between tasks.

One way to mitigate the effect of task interference is an appropriate task combination strategy. Most simply, task losses are combined via an unweighted sum, or equal unitary weighting, as shown in 3.3. While there are arguments in favour of this scheme [68], equal weighting has been shown to produce unsatisfactory performance across some or all tasks, likely due to its assumption that all tasks equally contribute to the objective function. As shown in 3.2, it is not uncommon for a subset of tasks to dominate the gradient update and to therefore potentially harm the parameter update through negative transfer. With the increased popularity of MTL in recent years, this has motivated the development of several task weighting methods, which take three general approaches.

The most common approach to task weighting is scalarisation. Scalarisation is any operation which combines a vector of objectives into a scalar form, and is commonly used in multi-objective optimisation (MOO). A weighted sum is the simplest form of scalarisation: linear scalarisation. Other methods include Pascoletti-Serafini [5] and conic goal programming scalarisation [96]. In loss-based task weighting methods, all objectives are scalarised in some way, through combining individual task losses into a single total loss upon which backpropagation is performed.

However, some loss-based approaches that go beyond pure scalarisation are called *adaptive*

loss-based methods [78]. They involve computing new weights at each model iteration based on the model's performance on each task, before combining the weighted losses through a scalarisation (usually linear). Such methods enjoy the same low computational cost as scalarisation, while also being capable of adapting to changing task importances over time, where scalarisation may fall short.

Finally, gradient-based task weighting techniques strive to tackle the adverse effects of conflicting and dominating gradients by directly modifying the gradients themselves. Gradient-based methods were first introduced for MOO problems by Désidéri [38] with their derivation of the Multiple Gradient Descent Algorithm (MGDA), which makes a step towards the minimum norm of the gradients with respect to each objective function, guaranteeing a Pareto-optimal solution, ie. no other solution in the feasible region dominates this solution (the goal of MOO). The set of all Pareto optimal solutions is called the Pareto set, and its image is the Pareto front or frontier.

In their work, "Multi-task learning as multi-objective optimisation", Sener and Koltun [101] extended this formulation to MTL problems and provided an approximation to MGDA for deep networks with a large number of parameters. The constrained optimisation problem for MGDA can be formulated as follows:

$$\min_{\alpha_1,\ldots,\alpha_K} \left\{ \left\| \sum_{i=1}^{K} \alpha_i \nabla_{\theta_{\text{sh}}} L_i(\theta_{\text{sh}}, \theta_i) \right\|_2^2 \right\} \text{ s.t. } \sum_{i=1}^{K} \alpha_i = 1,\ \alpha_k \geq 0\ \forall k \tag{3.7}$$

where L_i is the loss of the i-th task, θ_{sh} are the shared parameters and θ_i are the task-specific parameters. MGDA uses Karush-Kuhn-Tucker (KKT) conditions to guarantee optimality [46, 100], which are:

- There exist $\alpha_1, \ldots, \alpha_K \geq 0$ such that $\sum_{i=1}^{K} \alpha_i = 1$ and $\sum_{i=1}^{K} \alpha_i \nabla_{\theta_{\text{sh}}} L_i(\theta_{\text{sh}}, \theta_i) = 0$
- For all tasks k, $\nabla_{\theta_k} L_k(\theta_{\text{sh}}, \theta_k) = 0$

All solutions that satisfy the KKT conditions laid out above are referred to as Pareto stationary points [101]. However, it should be noted that while every Pareto optimal point is Pareto stationary, the reverse is not necessarily true. These methods have been shown to fully explore the Pareto frontier, even with non-convexity, which is not the case scalarisation and loss-based methods [59]. One of the drawbacks of gradient-based task weighting, however, is that K backward passes are required to compute the gradients of each individual task with respect to the model parameters.

3.4 Loss functions

Tangential to the question of loss value weighting in multi-task learning, is the choice of loss function itself. Traditional loss functions such as cross-entropy have proven effective for many applications, but they often fall short in scenarios involving imbalanced data, hard-to-classify samples, or tasks like image segmentation, where pixel-wise accuracy is insufficient for assessing model performance. In recent years, several loss functions have been developed to address challenges in machine learning tasks, particularly in classification and segmentation problems.

The Dice loss [105] has gained prominence in medical image segmentation [118, 117, 60] and natural language processing tasks [72]. Derived from the Sørensen–Dice coefficient (DSC) [106, 40],

this loss function effectively handles class imbalance issues often encountered in segmentation problems. The Dice loss measures the overlap between predicted and ground truth segmentation masks, providing a robust alternative to pixel-wise cross-entropy loss. Focal loss, proposed by Lin [77], addresses the challenge of extreme foreground-background class imbalance. This loss function dynamically adjusts the weight of each sample based on the model's confidence in its prediction, allowing the model to focus on hard, misclassified examples.

Polynomial Loss (PolyLoss) [71] was introduced as a comprehensive loss framework through which the aforementioned loss functions can be interpreted, and from which new loss functions can be composed from a principled starting point. For example, loss functions such as CE Loss and Focal Loss are formulated as special cases of the PolyLoss family. This is demonstrated by decomposing these loss functions into a series of polynomial bases (through a Taylor expansion) weighted by coefficients α. When α values are equal for all bases, PolyLoss collapses to a simple Taylor expansion of CE Loss, and when all α values are equally shifted by the Focal focusing parameter γ, PolyLoss similarly collapses to Focal Loss. Shifting α horizontally can reproduce these well-known losses, while shifting α vertically (ie. dropping polynomial bases) can also produce new loss functions. One such function proposed in the paper is Poly-N Loss, which perturbs the leading N polynomial terms by a hyperparameter ϵ while keeping the rest the same:

$$L_{\text{Poly-N}} = \underbrace{(\epsilon_1 + 1)(1 - p_{\text{T}}) + \cdots + \frac{\epsilon_N + 1}{N}(1 - p_{\text{T}})^N}_{\text{perturbed by } \epsilon_j} + \underbrace{\frac{1}{N+1}(1 - p_{\text{T}})^{N+1} + \ldots}_{\text{same as } L_{\text{CE}}} \tag{3.8}$$

$$= -\log(p_{\text{T}}) + \sum_{j=1}^{N} \epsilon_j (1 - p_{\text{T}})^j \tag{3.9}$$

where p_{T} are the outputs (probabilities or logits) returned by the model. The first polynomial is observed to contribute to over half of the CE gradient compared to the rest of the infinitely many terms, motivating the formulation of the simple Poly-1 Loss:

$$L_{\text{Poly-1}} = (1 + \epsilon_1)(1 - p_{\text{T}}) + \frac{1}{2}(1 - p_{\text{T}})^2 + \cdots = -\log(p_{\text{T}}) + \epsilon_1(1 - p_{\text{T}}) \tag{3.10}$$

This loss function has shown impressive performance increases over traditional CE and Focal loss in many settings already [44, 20, 73, 53, 6], with the addition of only one line of code and one additional hyperparameter, the perturbation value ϵ.

Chapter 4

Methods

4.1 Model

Several changes have been made to the original MaskFormer architecture to better align with the specifics of b-jet tagging and vertex reconstruction [104]. Firstly, a set of charged particle tracks is used as input instead of images; this simplifies the Mask2Former architecture by removing the upsampling step. The model's initial stage is a dense embedding layer producing a $d = 256$ dimension vector for each of the M input tracks. This is followed by sinusoidal positional encoding [110] of track impact parameters and angular displacements from the jet axis. The positional encoding allows the self-attention mechanism to identify nearby tracks with low computational overhead [94]. The input embeddings are then fed into a 4-layer self-attention transformer encoder network which performs feature extraction and outputs a second embedding for each track. The encoder has dimension d and an update width of $2d$. ReLU activations are used throughout the model.

The object decoder generates a representation of a set of *object queries*, representing potential output vertices. Object queries are initialised as $N = 5$ learned vectors of dimension d. The number of object queries corresponds to the maximum number of target vertices per jet in the training sample. In each of the three decoder layers, cross-attention is used to exchange information between object queries and track representations from the encoder, and self-attention is used to exchange information between object queries themselves. These attention operations update the object queries by aggregating information about the entire jet efficiently. For cross-attention operations, the MaskAttention operator is used [23]. The final decoder layer's outputs are the updated track representations and object queries, used to predict vertices' presence and properties, along with their track masks, as detailed below.

Task heads in the model consist of task-specific dense networks, each with three hidden layers of 256, 256, and 128 nodes. These networks produce the five outputs of the model: namely vertex class, masks, regressed vertex properties, track origin and jet flavour. The first three of these outputs are produced by the object decoder. Vertex class labels are b-hadron, c-hadron, or Null, with Null indicating no target vertex. Masks are used for vertex finding, ie. assigning input tracks to output vertices. N 1D masks of length M are computed from mask tokens of dimension d and track embeddings. Vertex properties are predicted using a dense, multi-target regression

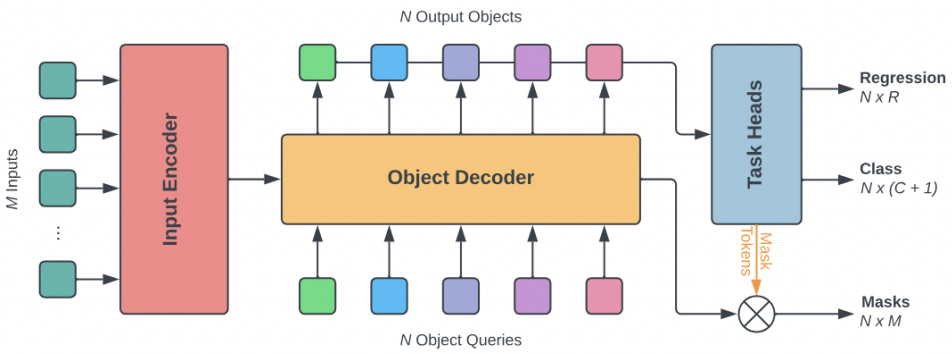


Figure 4.1: Diagram of the MaskFormer network architecture for $N = 5$ object queries. M inputs are fed into a Transformer encoder. The object decoder then takes a set of N object queries and updates them with information from the encoder and from other object queries. Finally, three task heads are used for object-level classification (includes NULL class), regression for R outputs and creating masks for vertexing.

network with five outputs: mass, transverse momentum (p_{T}), radial flight length (L_{xy}), and $d\phi, d\eta$ which, combined, make up angular displacement (dR). These targets are based on simulation-level properties of heavy flavour hadrons.

The vertexing losses are combined as follows [1],

$$L_{\text{Vertexing}} = 2L_{\text{Class}} + \underbrace{10L_{\text{BCE}} + 2L_{\text{Dice}}}_{L_{\text{Mask}}} + L_{\text{Regression}} \tag{4.1}$$

The weights for these losses are also passed into the Hungarian bipartite matching algorithm as matcher weights to compute the matching cost, as shown in Equation 3.1.

For jet flavour prediction, the model outputs probabilities for b-jet, c-jet, and light-jet classes by independently pooling track and vertex representations directly from the encoder (not shown in Figure 4.1) using attention pooling, concatenating the results, and feeding them into the final dense network (ie. task head). The track origin task head is also positioned directly after the encoder, with no pooling. The final loss is calculated as

$$L_{\text{Total}} = L_{\text{Jet Class}} + 0.5L_{\text{Track Origin}} + L_{\text{Vertexing}} \tag{4.2}$$

4.2 Motivation

As shown in 4.1, the losses for each of the MaskFormer tasks are weighted using static task weights, which were chosen based on performance combined linearly through summation. This type of loss weighting has two main drawbacks: the first is that it is difficult to know what the optimal static weight choice is, and finding it through a grid search is both inefficient and would still not address the second key drawback, which is that task importances may vary during different points of the training process [81].

[1] As of writing, the MaskFormer paper is still under review. The weights shown in the paper have since been updated with the ones used in our work as they achieve better performance.

The optimal scenario in a MTL framework is that each task is chosen and weighted in such a way that it boosts, or at the least does not negatively impact, the performance of the other tasks. However, such a scenario is often infeasible due to issues arising from the tragic triad (see Section 3.3).

A third drawback of the current task weighting scheme is the actual choice of the tasks: how can we know that an auxiliary task is beneficial to performance in advance? Task choice is a key component of multi- and auxiliary-task learning paradigms, and can be difficult to adequately choose beforehand. [86] showed that auxiliary tasks can be reverse-engineered by training an auxiliary network to generate auxiliary labels while training the main network to learn both the main task and the auxiliary task. Ablation studies were conducted for GN1, where one or both of the auxiliary tasks were removed to observe changes in performance; it was found that c-jet and light-jet rejection performance was reduced when any of the two auxiliaries were removed [32]. To our knowledge, ablation studies have not yet been conducted for this model to observe how the incorporation of each task impacts overall training. The choice of tasks for MaskFormer is outside the scope of our work and remains an interesting avenue for further research.

The performance of MaskFormer surpasses that of EdgeClassifier, however does not yet surpass the performance of GN2, and it is as yet unknown whether one or more MaskFormer tasks is harming overall training performance. Additionally, it is harder to quantify how a task can be "harmful" to the model in a MTL framework as typically there is no one task that takes precedence over the others, but instead all are deemed equally important. For this reason, when necessary, we will quantify the problem as an auxiliary task learning framework, with jet classification as the main task, as traditionally b-jet classification has been the driving force behind all the machine learning models described so far.

To address the first two issues delineated above, we aim to find a task weighting algorithm that removes the need for manual weight tuning and, in the case of varying task importance throughout the model training, can adaptively change these weights.

4.3 Implementation

4.3.1 Dataset

The training and evaluation data contains jets from top anti-top decays simulated in Pythia 8.307 with a centre of mass energy of $\sqrt{s} = 13$ TeV, with a mean pileup of 50 and a detector response modelled in the Delphes 3.5.0 framework. A detailed summary of this dataset is available at [93].

The input dataset contains jet-, track- and object-level data. The jet-level inputs contain jet pseudorapidity, η, transverse momentum p_{T}, mass and summaries of the total number of hadrons and tracks in each jet. The truth label for jet classification is flavour, which identifies whether a jet is a b-jet (containing at least one b-hadron within radius $dR < 0.4$ of the jet), a c-jet (no b-hadrons but at least one c-hadron), or a light-jet (containing neither b- nor c-hadrons). The jet flavour prediction is outputted by the model as a per-class probability vector for the three jet classes listed above.

The truth label for the track origin task, which classifies what interaction a track originated from, is shown in Table 4.1 below.

Truth Origin	Description
pileup	Originating from a soft interaction
fake	Created from the hits of multiple particles
primary	Originating from primary interaction
from b	Created from the decay of a b-hadron
from bc	Created from a tertiary c-hadron decay, which itself is from a b-hadron decay
from c	Originating from the decay of a c-hadron
from τ	Originating from the decay of a τ
other secondary	Created from other secondary interactions

Table 4.1: Truth origin labels for tracks.

The model processes the 50 tracks with the highest p_{T} for each jet. Each track contributes several input variables, such as the signed impact parameters (d_0 and z_0), where the sign is positive for tracks in front of the primary vertex and negative for those behind, the proportion of the jet's p_{T} carried by the track, and the track's pseudorapidity η and azimuthal angle φ relative to the jet axis. The truth labels for tracks are the truth vertex ID: if two tracks have the same vertex ID, they originated from the same vertex.

The dataset contains up to 5 of the highest p_{T} heavy-flavour hadrons associated with each jet. Hadron properties, namely $p_{\mathrm{T}}, L_{xy}, d\phi, d\eta$, and mass are fed into the model as inputs to be accurately regressed as an output, as well as each hadron's flavour which the model uses to produce a class-probability vector with three values: the probability that the object is a b-hadron, a c-hadron, or a padded (Null) hadron.

The inputs to the model are in the form of HDF5 structured datasets, with each variable summarised in Table 4.2, and the full training dataset consists of 120 million jets, equally distributed among the three jet flavours.

4.3.2 Salt Framework

All ATLAS jet flavour tagging algorithms are contained in `Salt` [10], a general-purpose framework for training models like DL1 and GN2, which is publicly available at `https://gitlab.cern.ch/atlas-flavor-tagging-tools/algorithms/salt`. All models are implemented using Lightning [43], a high-level wrapper for PyTorch [90] that streamlines code organisation and automates much of the training loop process. The framework allows for model configuration using YAML files and Lightning's command-line interface.

Our work draws heavily from the novel open-source Python library for multi-task learning, LibMTL [76]. LibMTL unifies 15 of the most popular task weighting algorithms under a singular PyTorch framework. In our own implementation of these weighting algorithms, we integrated core functions from LibMTL into a modular and user-friendly parent `Weighting` class that can be called from the command-line or the YAML configuration files, and integrates well with the Salt framework. Because of the modular nature of LibMTL, we were also able to include two novel weighting methods not yet included in the LibMTL framework in our own research. Our code is made publicly available at `https://gitlab.cern.ch/isanai/salt`.

For the majority of these weighting algorithms, the core training loop needs to be customised. This was achieved using LightningModule's hooks, which allow us to overwrite PyTorch's standard

Jet Input	
p_{T}	Jet transverse momentum
η	Signed jet pseudorapidity
Track Input	
q/p	Track electric charge divided by momentum (curvature measure)
$\mathrm{d}\eta$	Pseudorapidity of the track, relative to jet η
$\mathrm{d}\phi$	Track azimuthal angle, relative to the jet ϕ
d_0	Closest distance from the track to the PV (longitudinal plane)
$z_0 \sin\theta$	Closest distance from the track to the PV in the transverse plane
$\sigma(q/p)$	Uncertainty on q/p
$\sigma(\theta)$	Uncertainty on track polar angle θ
$\sigma(\phi)$	Uncertainty on track azimuthal angle ϕ
$s(d_0)$	Lifetime signed transverse IP significance
$s(z_0)$	Lifetime signed longitudinal IP significance
nPixHits	Number of pixel hits
nSCTHits	Number of SCT hits
nIBLHits	Number of IBL hits
nBLHits	Number of B-layer hits
nIBLShared	Number of shared IBL hits
nIBLSplit	Number of split IBL hits
nPixShared	Number of shared pixel hits
nPixSplit	Number of split pixel hits
nSCTShared	Number of shared SCT hits
nPixHoles	Number of pixel holes
nSCTHoles	Number of SCT holes

Table 4.2: Jet and Track Input Descriptions

training methods with our own implementations. For example, if an array needs to be re-initialised at the beginning of each training epoch, the relevant hook (which is otherwise empty) can be modified within the LightningModule as follows,

```
def on_train_epoch_start():
    self.array = []
```

and Lightning will call the method at the right time in the training process.[2]

The backward pass also had to be customised for weighting methods that access and modify the gradients directly. This was achieved using Lightning's manual optimisation configuration, which allows for a custom manual backward function and is more similar to traditional PyTorch training.

We also modified the framework from weighting each task loss in its respective script (mask-related losses were previously weighted in the `MaskFormer_loss.py` script, before the jet classification and regression losses, for example) to weighting them all simultaneously at the end of each training step. This allowed us to log the raw task losses using Comet, providing a fair comparison of the training process for each model. Lastly, we logged the weights of each task during training to observe how their importance varied with time.

4.3.3 Training

To account for limited time resources, three key changes are made to the standard MaskFormer model training:

1. The model is compressed by halving the dimension of the initial dense embedding layer to $d = 128$; the resulting small MaskFormer model has 1.8 million trainable parameters as opposed to 6.3 million in the full-sized model [104].

2. The number of training samples is also reduced from 120 million to 20 million, with 1 million validation samples and 500,000 evaluation samples.

3. The training process is set to run for 20 epochs instead of 50.

4. The weights for the Hungarian matcher are held constant at the values shown in Equation 4.1across all trainings. This was done to reduce confounding factors when evaluating different weighting algorithms.

Models are trained on one NVIDIA A100 GPU or on one half of a NVIDIA A100 multi-instance GPU (MIG), which partitions each A100 GPU into two "light" GPU instances. For more information on MIGs, see here [88]. Depending on the weighting algorithm, full training of the model was completed in anywhere between 10 to 30 hours. For optimisation, we employ the AdamW (Adam with weight decay) algorithm, which separates the weight decay from the gradient update, leading to better generalisation, especially in large models [83]. The learning rate is managed by a OneCycle scheduler, which starts training with an initial learning rate of 1×10^{-7}, increasing to a maximum value of 5×10^{-4} once 5% of training has elapsed, and gradually

[2] More details on Lightning's hooks and the pseudocode for where each hook is called can be found at `https://lightning.ai/docs/pytorch/latest/common/lightning_module.html#hooks`

decreasing to 1×10^{-5} by the end of training. For logging and experiment tracking, we use the Comet logger [41]. Evaluation is performed on the epoch with the lowest validation loss.

4.4 Metrics

ATLAS flavour tagging models have a set of commonly used metrics, which are plotted using the Puma Python plotting API package, and will be described here.

The multi-class jet flavour output is combined into a single output score called a discriminant. The b-jet discriminant is calculated as

$$\mathcal{D}_{\mathrm{b}}(f_c) = \log\left(\frac{p_b}{f_c \cdot p_c + (1 - f_c) \cdot p_l}\right), \tag{4.3}$$

where the c-jet fraction f_c tunes the trade-off between c-jet and light-jet rejection (a higher f_c increases c-jet rejection but decreases light-jet rejection), and p_b, p_c and p_l are the probabilities that the jet is a b-jet, c-jet or light-jet, respectively.

This formula can also be used for c-jet tagging by simply flipping all the c values to b in the above equation, as follows

$$\mathcal{D}_{\mathrm{c}}(f_b) = \log\left(\frac{p_c}{f_b \cdot p_b + (1 - f_b) \cdot p_l}\right), \tag{4.4}$$

The discriminant is then used to calculated the efficiency:

$$\varepsilon = \frac{N_{\text{tagged}}^{\text{signal}}}{N_{\text{tagged}}^{\text{signal}} + N_{\text{untagged}}^{\text{signal}}}. \tag{4.5}$$

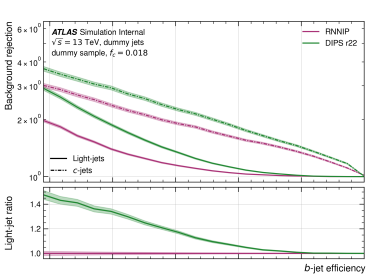
As a function of the discriminant, the efficiency of a some flavour j (b, c, or light) is defined as,

$$\varepsilon^j = \frac{N_{\text{pass}}^j(\mathcal{D} > T_f)}{N_{\text{total}}^j}, \tag{4.6}$$

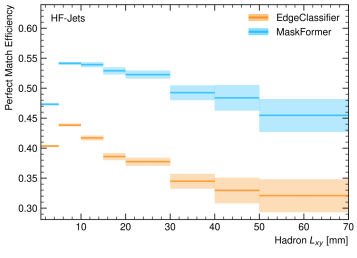
where $N_{\text{pass}}^j(\mathcal{D} > T_f)$ are the number of jets of flavour j beyond the cut T_f on the discriminant $\mathcal{D}$ and N_{total}^j are all jets of flavour j before the cut.

b-jet efficiency is used in practice when evaluating the performance of b-jet tagging algorithms. Receiver Operating Characteristic (ROC) curves are predominantly used for this purpose, to illustrate the trade-off between the b-jet tagging efficiency and the rejection rate of non-$b$jets (light-flavour jets or c-jets). The b-jet tagging efficiency represents the fraction of true b-jets that are correctly identified by the algorithm (Equation 4.5 and 4.6), while the rejection rate indicates the algorithm's ability to correctly exclude non-b jets. The inverse of the non-b jet acceptance rate (background rejection) is often used on the y-axis of the ROC curve. This allows for a clear visualisation of the algorithm's discriminative power across different operating points.

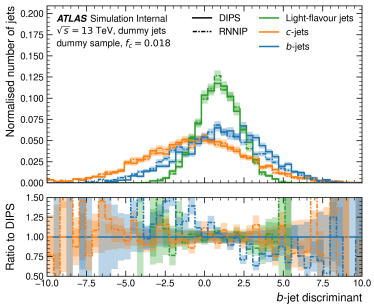
Models are often compared at a fixed b-jet efficiency operating point, and the model with a higher background rejection at this point is deemed to have better performance. There are four common working points for b-tagging in ATLAS, namely at 60%, 70%, 77% and 85% efficiency [4], thus we have chosen a b-jet efficiency range of 49% to 100% for plotting. An example of such an ROC plot is shown in Figure 4.2a.



(a) Receiver Operating Characteristic curve. [55]



(b) Efficiency plot for visualisation of vertexing performance [104].



(c) Regression residual plot [56].

For vertex finding, efficiency plots are used to evaluate performance as shown in Figure 4.2b. These plots typically feature the vertexing efficiency (fraction of correctly predicted vertices) on the y-axis against a relevant variable on the x-axis, such as p_{T} or L_{xy}. The efficiency is often calculated as the ratio of reconstructed vertices to the total number of true vertices within specific bins of the x-axis variable.

For object regression tasks, such as fitting hadron mass and p_{T}, we use residuals to understand the model's performance on a continuous feature space (see Figure 4.2c). Residuals are calculated as the difference between predicted and true values, and the mean and standard deviation of the residuals are important indicators of performance in regression plots. A mean near zero and a low standard deviation implies that the regressed predictions are predominantly clustered near the true value and do not significantly deviate from it.

Additionally, residuals are calculated based on how many objects were correctly classified as "valid". To make a meaningful comparison against the truth hadron data, valid objects are defined as those which 1) have a p_{null} value of less than 0.5, and 2) have corresponding valid hadrons in the truth file. For example, if an object has a predicted p_{null} value of 0.1 but does not have a corresponding truth hadron in the truth dataset, no residual will be computed. Thus, another important feature to consider in residual plots is the height of the residual peak, which represents the volume of objects correctly identified as hadrons; one model may have a lower mean and standard deviation but may also generally identify fewer hadrons for each jet.

Chapter 5

Experiments

5.1 Baseline Experiments

Two initial baseline experiments were run using handpicked task weights, one with the task weightings from 4.1 (which we will refer to as the "default" weighting scheme), the second with equal weights for all tasks. Subsequent model comparisons will focus on improving on these baselines, as it is expected that the performance increase on a small MaskFormer will extend to the standard version. These baseline models are used in the following section to probe into task interactions.

As of writing, MaskFormer falls behind GN2's jet classification performance by 10%-20%. The jet classification performance of the two MaskFormer baseline models (one with unweighted task losses, the other with default weights from Equation 4.1) is also shown against two references of GN2. The first reference is a fully-trained GN2 model, "GN2-Full"; this model has 2.3 million parameters and has been trained with 260 million jets. Given the smaller model and dataset that we use in this work (10× less data), we retrain another GN2 ("GN2-Small") to perform a fairer comparison. This retrained model has 1.7 million parameters and is trained on 30 million training samples. The first training of this model overfit on the validation set, and so the parameters were reduced to 1.4 million and dropout layers were added to the encoder and to each of the task-specific networks to mitigate this.

The baseline models both perform significantly worse than GN2-Full (lower than the drop in performance of 10%-20% for a full-sized MaskFormer) due to the significant reduction in model size, and marginally worse than GN2-Small, as shown in Figure 5.1.

5.2 Task Interactions

The questions we ask in this section are twofold: the first is, how do each of the MaskFormer tasks interact with one another? The second question is why does scaling the weights as in 4.1 benefit each task so much, and how is this performance increase reflected in task interactions?

To answer these questions, we have conducted two of the three tests proposed in [114] to perform a sort of "gradient surgery" on our model. By probing into the gradients of the tasks in our model, we aim to quantify task relatedness outside of their correlation in the realm of particle physics. We first present these metrics for the equal weighting scheme, where the losses remain

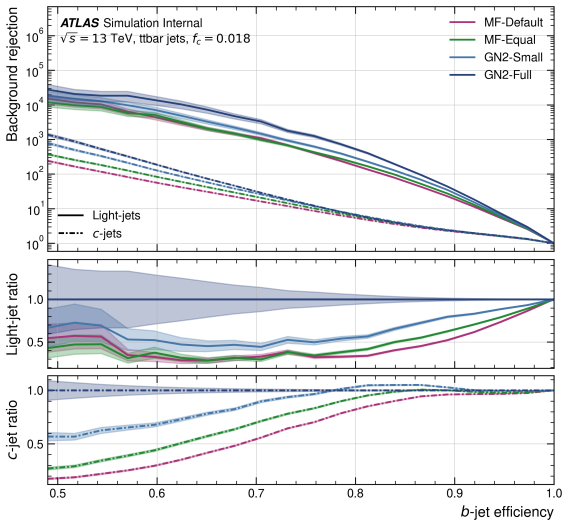


Figure 5.1: Jet classification ROC performance curve as a function of b-jet efficiency for default weights (Equation 4.1) and equal weights, benchmarked against a full-sized GN2 model. A small GN2 model is also shown as a fairer benchmark.

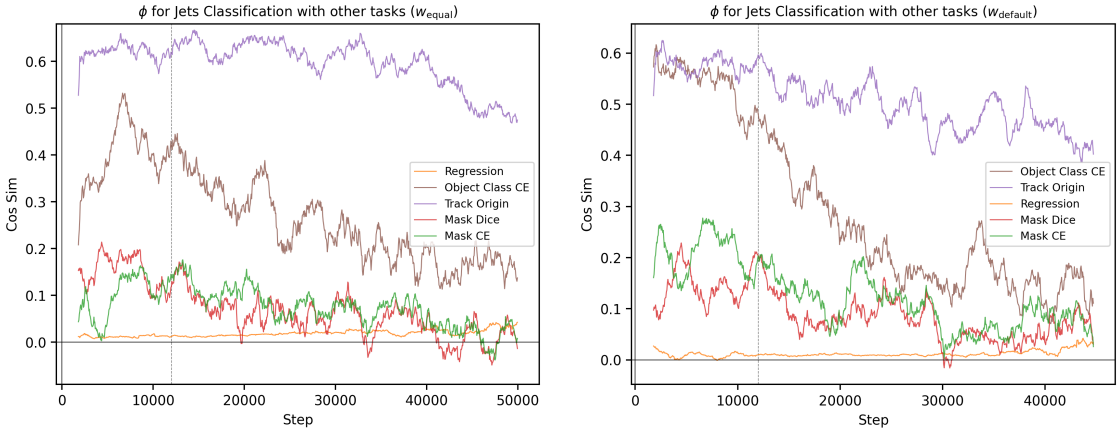
unscaled and are combined through a simple summation. We then compare these values to those of the default weighting scheme and observe how the metrics change as a result, if at all.

Computing the integral for the third element of the tragic triad (high positive curvature 3.6) requires numerical integration using the Trapezoid approximation in the deep learning setting. Although the loss values for the current and next iterations, $\nabla L(\theta^{(t)})$ and $\nabla L(\theta^{(t+1)})$ are already computed by the model, the complicated term is $\nabla L(\theta^{(t)} + \alpha(\theta^{(t+1)} - \theta^{(t)})$. Because we integrate over α, this gradient needs to be computed once for every trapezoid that we choose to step over, leading to a linear increase in training time with number of integral steps. Due to low resources and high number of backward passes needed to compute this metric accurately, we focus on the first two elements of the tragic triad and leave the third as an avenue for future research.

5.2.1 Cosine Similarities

We present here our analysis of the gradient cosine similarities between all task-pairs in the model. Though a total of 30 values (6 parent tasks, 5 cosine similarities) are shown in the graphs below, half of these are repeated because cosine similarity is symmetric, ie $\phi_{ij} = \phi_{ji}$. As expected for a deep neural network, the metrics for each batch of the network are very noisy, making gradient similarity metrics difficult to interpret. We have thus employed a rolling window smoothing to the original data for visualisation purposes, with a window size of 15. The first window of data was discarded for clarity, and the results are shown below.

We first analyse the Jet Classification gradient cosine similarities before comparing the differences observed between the two weighting schemes. Jet Classification's loss gradient is found to be strongly similar to Track Origin ($0.45 < \phi < 0.65$), which intuitively makes sense, as the process from which a track originates is highly correlated with the flavour of the associated jet. There is also a non-negligible cosine similarity with Object Class. This was expected to be higher, as a b-jet is defined by the presence of a high-p_{T} b-hadron within the jet. However, this may be explained



(a) Unitary weights.

(b) Default weights.

Figure 5.2: Cosine similarities for the Jet Classification task loss gradients with the loss gradients for other tasks in the MaskFormer model. The task gradients are computed with respect to shared model parameters.

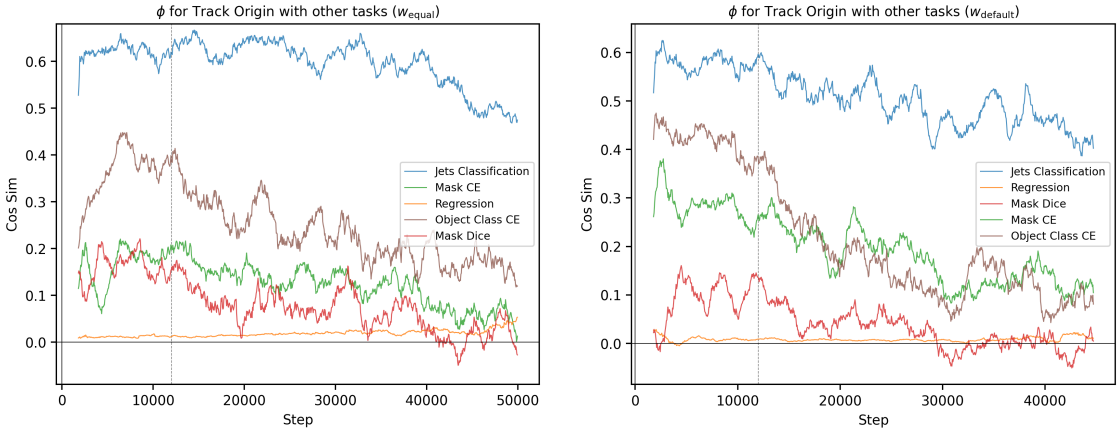
by the fact that the Object Class task regresses not only the leading p_{T} hadron properties, but the leading *five*. If the other hadrons are not indicative of jet flavour, then this may lower the similarity between the two tasks. The two mask tasks have low positive cosine similarity with Jet Classification, while Regression is close to 0.

When comparing how task weights impact ϕ for Jet Classification in Figure 5.2, we can observe that its cosine similarity value with Object Class CE is most affected by altering task weights in 5.2b, increasing from an average of 0.38 to 0.58 across the first 10,000 steps, and then sharply decreasing to approximately 0.1 by the end of training. In the equal weighting scheme, however, the cosine similarity between these two tasks is lower on average and experiences a less significant decrease across training. Otherwise, the remaining tasks' gradient cosine similarity with Jet Classification (except for Regression, which remains almost unchanged) are $0.05 - 0.1$ higher for default weights, indicating that this weighting aligns the task gradient directions with that of Jet Classification more.

Track Origin has very similar $\phi_{\text{Track Origin, Other Tasks}}$ spread as the Jet Classification task in 5.2, and its ϕ value with Object Class CE follows a similar drop throughout training. The cosine similarities with other non-Regression tasks also see an increase with Track Origin when implementing the default weighting approach.

In general, Regression has the lowest correlation with all the other tasks in the network, with $\phi < 0.05$ in all cases. This is somewhat surprising, as from Section 2.2.2, a b-jet is defined as a jet which originates from a b-hadron. Additionally, other regression variables such as hadron mass are signatures of b-hadrons. Thus, it would be expected that regression be more strongly correlated to jet classification. However, the flavour of a jet is usually associated with the leading p_{T} hadron, whereas our model matches the 5 leading p_{T} hadrons, which may partially explain some of this lower correlation.

Though it is unlikely to have a large quantitative impact on model performance, setting default weights suppresses the gradual increase in Regression's cosine similarity with Object Class CE,

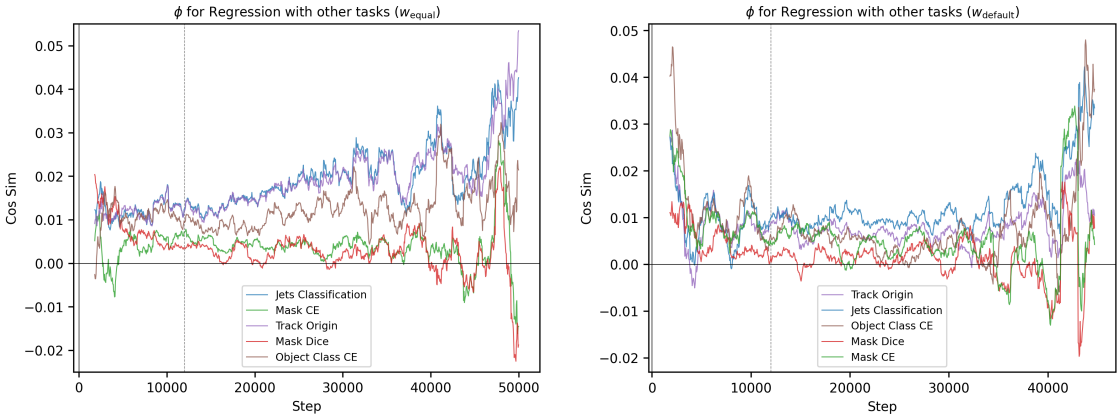


(a) Unitary weights.

(b) Default weights.

Figure 5.3: Cosine similarities for the Track Origin task loss gradient with the loss gradients of the other tasks. The gradients are computed with respect to shared model parameters.

Jet Classification and Track Origin, keeping $\phi < 0.02$ for the majority of training. [114] formulates that only negative gradient cosine similarity is an issue, while Wang et al. [111] suggest that very low positive ϕ between tasks can be detrimental to model performance; in general, it is non-trivial to determine what is the *optimal* value of cosine similarity outside of it being non-negative. We suggest that tasks that have ϕ values that are too high (closer to 1) would lead to a degenerate problem, where there is no information gain from training multiple tasks together, and the multi-task model could effectively be collapsed into a a singe-task model as a result. However, it is as yet unclear whether a value closer to 0 would be more beneficial by encouraging more exploration of the Pareto frontier without harming other tasks, or whether, alternatively, strong orthogonality is harmful and makes it more difficult to reach Pareto optimality.



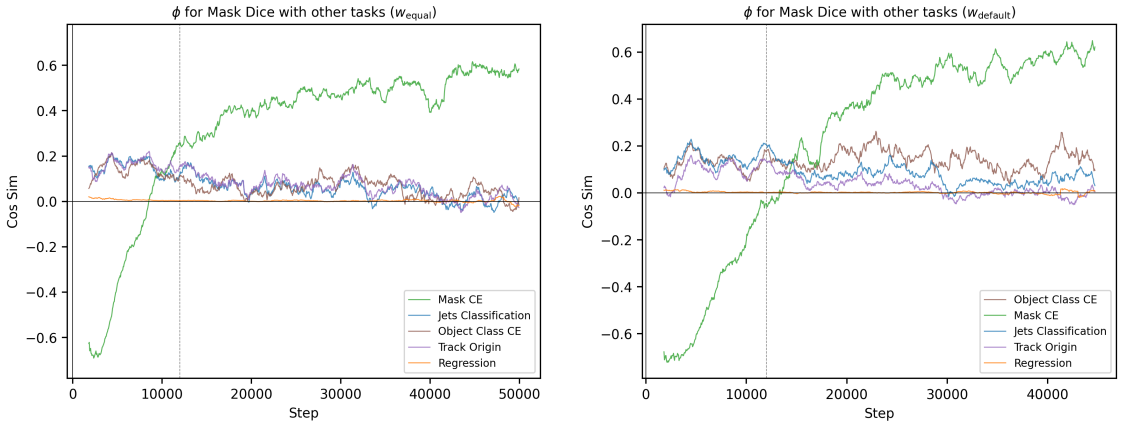
(a) Unitary loss weights.

(b) Default loss weights.

Figure 5.4: Cosine similarities for the Regression task loss gradients and the gradients of other task losses. The task gradients are computed with respect to shared model parameters.

It is also observed across Figures 5.4a and 5.5a that, with equal weighting, Track Origin and Jet Classification have almost identical ϕ values with the corresponding parent task across all training

steps. By contrast, the default weighting scheme in Figures 5.4b and 5.6b seems to decouple these values, by reducing the overall cosine similarity value of Track Origin with Regression and Mask Dice.



(a) Unitary weights.

(b) Default weights.

Figure 5.5: Cosine similarities for the Mask Dice task loss gradients and the gradients of other task losses. The task gradients are computed with respect to shared model parameters.

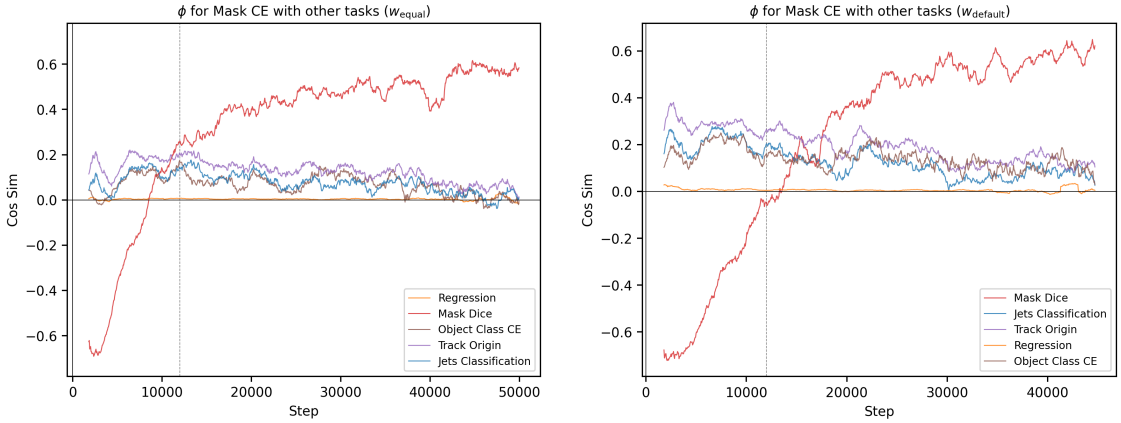
The only clear evidence we have of negative cosine similarities between tasks is between Mask Dice and Mask CE losses, as shown in 5.6 and 5.5, with $-0.6 < \phi_{\text{Mask Dice, MaskCE}} < 0$ in the first few thousand batches. We theorise that this could be due to one of two reasons:

1. This MaskFormer model has only 21% of the parameters relative to the full model. Thus, tasks that are more closely-related, such as mask losses, might compete more strongly over limited resources in the earlier stages of training.

2. In the literature, the combination of these two loss functions has been shown to improve performance in many segmentation [61, 92] and MTL [48] models. However, to our knowledge, the interplay between them, through cosine similarity metrics or other, has not yet been studied. Thus, it may be the case that the initial negative cosine similarity between them might actually be the reason why these two loss functions tend to work better synergistically. Since negative cosine similarity in itself is not necessarily harmful, it should be considered as part of a bigger picture and may even encourage better learning, analogous to how exploration is encouraged in the first few episodes of reinforcement learning models despite sacrificing short-term greedy rewards.

5.2.2 Gradient Magnitudes

In this section, gradient magnitudes are compared visually based on their L_2 gradient norms. L_2 norms suppress gradients lower than 1, which are less expected to contribute to task dominance and amplify the presence of larger gradients. The results for L_1 gradients are shown in Appendix A). For the default weighting scheme, gradient magnitudes are computed for the gradients of the task losses *before* they are scaled at each iteration as well as after. This is to study the impact of



(a) Unitary weights.

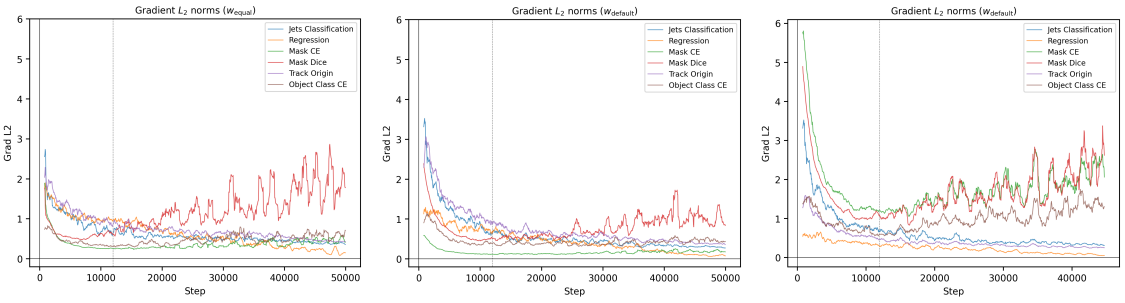
(b) Default weights.

Figure 5.6: Cosine similarities between MaskCE loss gradients and the gradients of other task losses. The task gradients are computed with respect to shared model parameters.

the scaling on the actual training dynamics, as well as their impact on the final backpropagated scaled gradients.

Before comparing both weighting methods, we observe in Figure 5.7 that at the point where ϕ between MaskCE and Mask Dice crosses 0 (which for the unweighted losses is at 9,000 iterations in Figure 5.5a, and for default weights is at 14,000 iterations in Figure 5.5b), the Mask Dice gradient starts to blow up and dominate the task gradients for the rest of training.

The impact of using non-unitary loss weights can especially be seen when evaluating gradient magnitudes. Using the weights from 4.1 leads to a steep increase in the gradient magnitudes for MaskCE and Object Class CE especially after the first 10,000 iterations.



(a) Equal weights.

(b) Default weights (pre-scaling).

(c) Default weights (post-scaling).

Figure 5.7: L_2 Gradient norms $\|G_k\|_2$ of MaskFormer tasks. The gradient magnitudes for an equal weighting scheme are shown on the left. The gradient magnitudes of the raw unscaled losses outputted during training with default weights are shown in the center. On the right, the gradient magnitudes obtained after scaling the losses are shown.

Notably, the default scheme *suppresses* Regression and the dominating Mask Dice gradients while *amplifying* MaskCE and Track Origin, as shown in Figure 5.7b. In doing so, the gradients for the masking tasks are made almost equal during training after scaling (see Figure 5.7c), potentially enhancing the MaskFormer's inherent masking capability. This change in Regression gradient

magnitude will be discussed further in our analysis of results.

5.3 Loss-based Weighting

In this section, we first present each of the loss-based weighting methods used in this work, and present our results in Section 5.6.

5.3.1 Geometric Loss Strategy

One of the drawbacks of combining losses through summation is differences in the scale of the individual losses. The Geometric Loss Strategy (GLS) is a scalarisation method which addresses this issue through a common geometric programming approach [24]. For a K-task problem with losses $L_1, L_2, ..., L_K$, the total loss is expressed as:

$$L_{\text{GLS}} = \prod_{i=1}^{K} \sqrt[K]{L_i} \tag{5.1}$$

5.3.2 Random Loss Weighting

Random loss weighting assigns random weights to tasks at each training iteration [75]. The weights are typically drawn from a Normal distribution, but can also be drawn from Dirichlet, Bernoulli and Uniform distributions. In our work, we use the standard Normal distribution, as it was shown to yield the highest performance increase with the shortest convergence time. Lastly, a softmax function is applied to constrain the weights to positive values, and the total loss function is computed using a standard weighted sum of the individual task losses.

$$\mathbf{w} = \sigma(\mathbf{z}) = \begin{bmatrix} \sigma(z_1 \sim \mathcal{N}(0,1)) \\ \vdots \\ \sigma(z_k \sim \mathcal{N}(0,1)) \\ \vdots \\ \sigma(z_K \sim \mathcal{N}(0,1)) \end{bmatrix} \tag{5.2}$$

$$L_{\text{RLW}} = \mathbf{w}^T\mathbf{L} = \sum_{i=1}^{K} w_i L_i \tag{5.3}$$

This method introduces stochasticity in the training process, and the loss function can be viewed as doubly stochastic when employing a variant of stochastic gradient descent for optimisation, such that the randomness comes from both the mini-batch data sampling and the loss weights [74]. This stochasticity helps the model navigate multimodal loss landscapes and avoid local minima, and it has been shown to reach comparable performance to more sophisticated weighting methods. RLW was included in our work to benchmark more complex weighting algorithms against a simple yet effective one that involves no prior tuning.

5.3.3 Uncertainty Weighting

Uncertainty weighting draws upon the concept of homoscedastic uncertainty in Bayesian modelling [65]. This is a type of aleatoric (or "irreducible") uncertainty that cannot be explained away with increased training data [49]. Furthermore, homoscedastic uncertainty is also independent of model inputs, making it interpretable as a task-dependent uncertainty. The authors propose to use this uncertainty to weight losses by introducing learnable uncertainty parameters, optimising them alongside model parameters. The resulting loss function is based on maximising the Gaussian likelihood with task-dependent uncertainty:

$$L_{\mathrm{UW}}(\theta, \sigma_1, \sigma_2, ..., \sigma_K) = \sum_{i=1}^{K} \frac{1}{2\sigma_i^2} L_i(\theta) + \log \sigma_i \tag{5.4}$$

where σ_i represents the task-specific uncertainty parameter, L_i is the task-specific loss, and θ are the model parameters. If a task has a higher uncertainty σ_i, this will lower the task's contribution to the overall loss function. To prevent the model from arbitrarily increasing each σ value to infinity to bring the total loss close to 0, the $\log \sigma$ term penalises large values of the uncertainty parameter.

5.3.4 Smooth Tchebysheff (Chebyshev) Scalarisation

Chebyshev scalarisation is a technique commonly used in multi-objective optimisation to transform a vector of multiple objective functions into a single scalar objective [64]. The primary goal of Chebyshev scalarisation is to generate Pareto-optimal solutions by systematically balancing different objectives, under the guidance of a preference vector λ.

This scalarisation method is based on the Chebyshev norm (or infinity norm) and emphasises the worst performance among the objectives. Concretely, given a set of objective functions $\{L_i\}_{i=1}^{K}$, Chebyshev scalarisation seeks to minimise the maximum deviation from a reference point z^*, typically representing an ideal point. This ideal point is often the best known value for each objective, which for our use case is the maximum loss across all the tasks, $\max_i L_i^{(t)}$.

The general form of the Chebyshev scalarisation can be expressed as:

$$\min \left\{ \max_{i \in \{1,\ldots,m\}} (w_i \cdot |L_i - z_i^*|) \right\} \tag{5.5}$$

This formulation minimises the largest weighted deviation from the ideal objective values across all objectives. Lin et al. [78] extend this algorithm from the domain of MOO to MTL by proposing a smoothed Chebyshev scalarisation as a fast and low-cost loss combination method which does not require gradient manipulation. The algorithm is shown in Algorithm 1.

5.3.5 Fast Adaptive Multitask Optimisation

FAMO [79] seeks to update the parameters in such a way that results in the largest worst-case improvement rate across all tasks. The rate of improvement for task k at a given iteration t is

Algorithm 1 Smooth Tchebysheff (Chebyshev) Scalarisation

Input: Preference λ, Initial x_0, Step Size $\{\eta^{(t)}\}_{t=0}^{T}$
for $t = 1$ to T **do**
  $L_{\text{STCH}}^{\mu} = \mu \log \sum_{i=1}^{m} \exp \lambda_i (L_i^{(t)} - \max_i L_i^{(t)})/\mu$
  $\theta^{(t)} = \theta^{(t-1)} - \eta^{(t)} \nabla_\theta L_{\text{STCH}}^{\mu}$
end for
Output: Final Solution $\theta^{(T)}$

$$r_k = \frac{L_k^{(t)} - L_k^{(t+1)}}{L_k^{(t)}} \tag{5.6}$$

By maximising the minimal improvement rate r across all tasks and subtracting $\frac{1}{2}\|\Delta\theta^{(t)}\|$ to prevent an under-specified system where the objective can be infinitely large, the optimisation problem for FAMO is as follows:

$$\max_{\Delta\theta^{(t)} \in \mathbb{R}^m} \min_{k \in [K]} \frac{1}{\alpha} r_k - \frac{1}{2}\|\Delta\theta^{(t)}\|$$

Instead of solving the primal optimisation problem, where $d \in \mathbb{R}^m$ (in the case of deep networks, m can be in the millions or billions), the dual problem is solved instead. The dual problem is equivalent to optimising the log-objective of MGDA [101], and the algorithm for FAMO is shown in Algorithm 2 below[1].

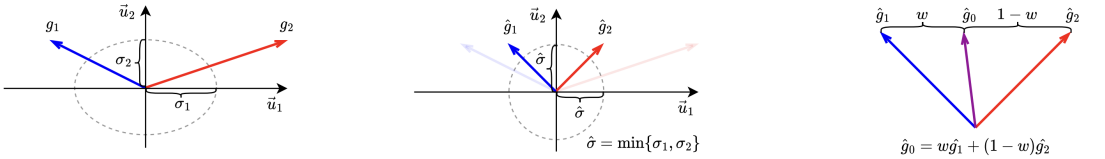
Algorithm 2 Fast Adaptive Multitask Optimisation (FAMO)

Input: Input: $\theta^{(0)}$ – initial parameter vector, $\{L_i\}_{i=1}^{K}$ – differentiable loss functions
Initialise: $L_{\min}$ array (set to 0s for loss functions that return positive values and ignored here for simplicity)
Initialise: m model parameter with corresponding optimiser m_{opt}
for $t = 1, \ldots, T$ **do**
  procedure WEIGHT LOSS($L^{(t)}$)
    Compute softmax of m: $\mathbf{z} \leftarrow \text{softmax}(m)$
    Compute task loss weights: $w^{(t)} \leftarrow \log(L^{(t)}) \cdot \mathbf{z} / \sum_i (\mathbf{z_i}/L_i^{(t)})$ and weight losses $L^{(t)}$
  end procedure
  procedure UPDATE WEIGHTS(m)
    Run second forward pass after backpropagation
    Compute difference: $\delta \leftarrow \log(L^{(t)} - \log(L^{(t+1)})$
    Compute gradient: $d \leftarrow \nabla_m \text{softmax}(m)$ with δ as output
    Update m using gradient d with optimiser m_{opt}
  end procedure
end for

5.3.6 Dynamic Weighting Average

The Dynamic Weight Average (DWA) algorithm adjusts task weights based on the relative rate of change of losses [81]. This loss change is computed at the end of every epoch. For task k at

[1] A positive non-zero ϵ value is added to each $L^{(t)}$ during computations to ensure numerical stability, but it is not shown here for brevity.



(a) Initial gradients (b) Gradients aligned via Aligned-MTL (c) Accumulated MTL gradient

Figure 5.8: Illustration of AlignedMTL procedure on two tasks. (a) Initial task gradients g_1 and g_2 have differing magnitudes and are conflicting. (b) Aligning principal components u_1, u_2 of initial linear system of gradients, ie. re-scaling axes of a coordinate system set by principal components. Aligned $\hat{g}_1$, $\hat{g}_2$ are orthogonal and of equal magnitude. (c) Aligned gradients are aggregated with a weighted sum, with pre-defined weights w.

epoch t, the weight is computed as:

$$w_k^{(t)} = \frac{K \exp(r_k^{(t-1)}/\tau)}{\sum_i \exp(r_i^{(t-1)}/\tau)} \tag{5.7}$$

where K is the number of tasks, τ is a temperature parameter controlling the softness of weight assignment, and $r_k^{(t-1)}$ is the relative loss change from one epoch to the next:

$$r_k^{(t-1)} = \frac{L_k^{(t-1)}}{L_k^{(t-2)}} \tag{5.8}$$

$L_k^{(t)}$ is calculated as the average loss of that task across all batches in the epoch t. While loss data is collected for the first two epochs, the weights are kept equal across all tasks. DWA aims to automatically balance tasks by giving more weight to tasks that are learning slowly relative to others, promoting more uniform learning progress across all tasks.

5.4 Gradient-based methods

In this section, we present each of the gradient-based weighting methods used in this work. Our findings are presented in Section 5.6.

5.4.1 Aligned Multi-Task Learning

AlignedMTL proposes using a condition number of a linear system of gradients as a stability criterion of an MTL optimization. The condition number of for the gradient matrix G[2] is defined as

$$\kappa(G) = \frac{\sigma_{max}}{\sigma_{min}} \tag{5.9}$$

where σ_{max} and σ_{min} correspond to the maximum and minimum singular values of the matrix, respectively. A linear system is considered to be stable if $\kappa = 1$.

[2]Different notation is used in this subsection to differentiate between the total cumulative gradient G_{Total} and the gradient matrix G. In the rest of this section, G refers to the total cumulative gradient.

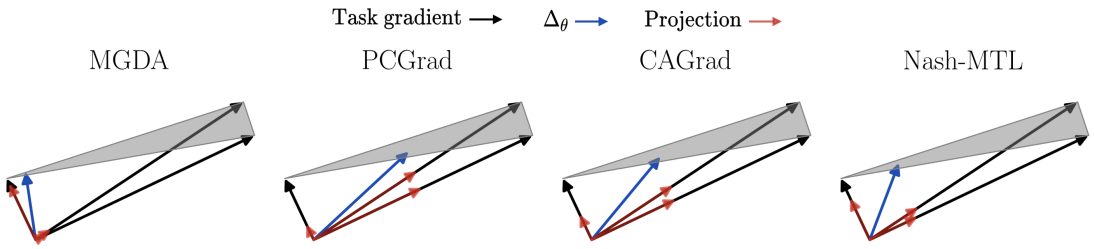


Figure 5.9: Visualization of gradient update direction for (a) MGDA, (b) PCGrad, (c) CAGrad, and (d) NashMTL. The update direction obtained by various methods on three gradients is shown in blue. Update directions are rescaled for better visibility. The size of the projection (red) of the update to each gradient direction (black) is also shown. [87]

Specifically, AlignedMTL aims to find a new cumulative gradient, $\hat{G}_{\text{Total}}$, such that $\|G_{\text{Total}} - \hat{G}_{\text{Total}}\|_2^2$ is minimised, while the overall linear system of gradients is stable ($\kappa(\hat{G}) = 1$). By applying a triangle inequality to the initial problem, this leads to

$$\|G_{\text{Total}} - \hat{G}_{\text{Total}}\|_2^2 \leq \|G - \hat{G}\|_F^2 \|w\|_2^2. \tag{5.10}$$

where G is the gradient matrix pre-alignment, $\hat{G}$ is the gradient matrix post-alignment, and w are the pre-defined task weights. Thereby, we consider the following optimization task:

$$\min_{\hat{G}} \|G - \hat{G}\|_F^2 \quad \text{s.t.} \quad \hat{G}^\top \hat{G} = I \tag{5.11}$$

Singular value decomposition (SVD) is applied to the gradient matrix, after which singular values corresponding to principal components are rescaled so that they are equal to the smallest singular value σ_{min}. This process is illustrated in Figure 5.8.

5.4.2 Projecting Conflicting Gradients

Projecting Conflicting Gradients (PCGrad) modifies conflicting gradients by removing the components of each gradient that conflict with the other task gradients [114]. This is achieved by projecting the conflicting gradients onto the normal plane of one other before summing their values to form the final gradient update.

For tasks i and j, this can be formalised such that if their gradients conflict (i.e., cosine similarity $\phi_{ij} = \langle G_i, G_j \rangle < 0$), the gradient of task i is modified as:

$$G_i^{\text{PCGrad}} = \begin{cases} G_i - \frac{G_i \cdot G_j}{\|G_j\|^2} G_j & \text{if } G_i \cdot G_j < 0 \\ G_i & \text{otherwise} \end{cases} \tag{5.12}$$

where G_i and G_j are gradients of different tasks, and G_i^{PCGrad} is the modified gradient for task i. This process is repeated for all task pairs, resulting in a new set of gradients that aims to reduce negative transfer between tasks, and is shown in Figure 5.9 (b).

5.4.3 Conflict-Averse Gradient Descent

Conflict-Averse Gradient Descent (CAGrad) makes use of the concept from MGDA of finding a parameter update $\Delta\theta$ that maximises the minimum decrease across the losses [80]. Concretely, this allows the average total gradient $\overline{G}^{(t)}$ to decrease while also decreasing each individual gradient, $G_i^{(t)}$. To do so, it defines an optimal update vector within an ϵ-ball centred at the average gradient, creating a constrained optimisation problem at each iteration as follows:

$$\max_{\Delta\theta\in\mathbb{B}_\epsilon} \min_{i\in[K]} \langle G_i^{(t)}, \Delta\theta^{(t)} \rangle, \tag{5.13}$$

$$s.t. \|\Delta\theta^{(t)} - \overline{G}^{(t)}\| \leq c\|\overline{G}^{(t)}\| \tag{5.14}$$

where $c \in [0, 1)$ is a hyperparameter which determines how quickly the optimiser converges to a solution. This minimises the average loss while leveraging the worst improvement of each task as a form of regularisation. The gradient direction at each step t is then calculated as:

$$\Delta\theta^{(t)} = -\ \overline{G}^{(t)} - c\|\overline{G}^{(t)}\| \left(\sum_{i=1}^{K} \lambda_i^{(t)} G_i^{(t)} \right) \tag{5.15}$$

where $\lambda_i^{(t)}$ is the Lagrange multiplier from the Lagrangian obtained from the optimisation problem in 5.13. The gradient update for CAGrad is shown in Figure 5.9 (c).

5.4.4 Nash-Bargaining Multi-Task Learning

NashMTL [87] formulates multi-task learning problems as a Nash bargaining game [7] with K players, each with their own utility function u_i derived from their gradient G^i. The players negotiate to an agreed gradient direction update that is proportionally fair, meaning that no one large gradient can dominate the update. If no agreement point is found, the players settle at the disagreement point, $\Delta\theta = 0$.

Similarly to CAGrad, NashMTL also searches for an update $\Delta\theta$ within an ϵ-ball, however here it is centred around the origin. The constrained optimisation problem is given by

$$\max_{\Delta\theta\in\mathbb{B}_\epsilon} \sum_i log(\Delta\theta^\top G_i) \tag{5.16}$$

$$\text{s.t. } G^\top G\alpha = \frac{1}{\alpha} \tag{5.17}$$

which can be solved by $\Delta\theta = \sum_i \alpha_i^{(t)} G_i^{(t)}$. We provide the complete NashMTL algorithm in Algorithm 3 below.

NashMTL also includes the option to update the task weights every $1 \leq t < T$ iterations, leading to a reduced computational overhead with relatively small drop in performance. The gradient update for NashMTL is shown in Figure 5.9 (d).

Algorithm 3 Nash-MTL

Input: $\theta^{(0)}$ – initial parameter vector, $\{L_i\}_{i=1}^K$ – differentiable loss functions, η – learning rate
for $t = 1, \ldots, T$ **do**
  Compute task gradients $g_i^{(t)} = \nabla_{\theta^{(t-1)}} L_i$
  Set $G^{(t)}$ the matrix with columns $g_i^{(t)}$
  Solve for α: ${G^{(t)}}^\top G^{(t)} \alpha = 1/\alpha$ to obtain $\alpha^{(t)}$
  Update the parameters $\theta^{(t)} = \theta^{(t)} - \eta G^{(t)} \alpha^{(t)}$
end for
Return: $\theta^{(T)}$

5.4.5 Gradient Vaccine

GradVac [111] is motivated by a drawback of PCGrad 5.4.2, which is its assumption that task-pairs must strictly have a gradient cosine similarity above zero to be modified by the weighting algorithm. As a result, PCGrad does not tackle the cases where the gradient similarity is positive but still detrimentally low to model performance.

GradVac introduces an adaptive gradient cosine similarity objective $\hat{\phi}_{ij}$ that is greater than 0, and is also larger than the cosine similarity between the two tasks i and j, ie. $\hat{\phi}_{ij} > \phi_{ij}$.

$$G_i^{\text{GradVac}} = G_i + \frac{\|G_i\| \left(\hat{\phi}_{ij} \sqrt{1 - \phi_{ij}^2} - \phi_{ij} \sqrt{1 - (\phi_{ij}^T)^2} \right)}{\|G_j\| \sqrt{1 - (\hat{\phi}_{ij})^2}} \cdot G_j \tag{5.18}$$

where ϕ_{ij} represents the cosine similarity between the gradients. The new objective cosine similarity is chosen as an exponential moving average of each task-pair's cosine similarity:

$$\hat{\phi}_{ijk}^{(t)} = (1 - \beta) \hat{\phi}_{ijk}^{(t-1)} + \beta \phi_{ijk}^{(t)}, \tag{5.19}$$

where i, j are a task-pair and k is the parameter (layer) granularity at which GradVac is calculated.

5.4.6 Gradient Normalisation

The GradNorm algorithm adjusts the task weights such that all tasks train at similar rates [21]. This is done in order to mitigate the effect of different training speeds and imbalanced gradient magnitudes as a result, which, as shown in Section 3.3, can cause negative transfer. The authors define a loss ratio which compares the loss of each task L_k with its value at the initial training step:

$$\tilde{L}_k^{(t)} = \frac{L_k^{(t)}}{L_k^{(0)}}$$

$\tilde{L}_k(t)$ represents the inverse training rate of task i, where lower values indicate a faster training rate for task i, and this ratio allows for relative gradient comparison through the relative inverse training rate for task i, $r_k^{(t)} = \frac{\tilde{L}_k^{(t)}}{\mathbb{E}_{\text{task}}[\tilde{L}_k^{(t)}]}$. Each task's r value is then scaled by the average gradient norm across tasks, providing a sensible common direction and scale for all tasks to move in. This method finally proposes an L1 loss function between the task gradient and its relative inverse

training rate (scaled by the average gradient norm), as follows:

$$L_{\text{GradNorm}}^{(t)} = \sum_i \left| \|G_W^i{}^{(t)}\|_2 - \|\overline{G}_W^{(t)}\|_2 \times \left[r_i^{(t)}\right]^\alpha \right|_1$$

where $G_W^i{}^{(t)} = \nabla_W w_i^{(t)} L_i^{(t)}$, the gradient with respect to W of i-th weighted task loss at time t, $\overline{G}_W^{(t)}$ is the average of these gradients across all tasks at time t and α is a hyperparameter which controls how strongly each task gradient is pulled in the direction of $\overline{G}_W^{(t)}$ (the strength of the "restoring" force).

5.5 Loss Function Modification

5.5.1 PolyLoss

In this section, we propose the use of a modified MaskCE loss function, Poly-1 MaskBCE (or Poly-MaskCE for short). Our contribution was also modularised, for reproducibility and future use. Poly-MaskCE is implemented by adding one line of code to the current MaskCE loss; our modification is shown below in cyan:

```
def mask_ce_loss(inputs: Tensor, labels: Tensor):
    pt = torch.sigmoid(inputs)
    pt = torch.where(labels == 1, pt, 1 - pt)
    epsilon = -1
    CE = F.binary_cross_entropy_with_logits(inputs, labels, reduction="none")
    loss = CE + epsilon * (1 - pt)
    # find the mean loss for each mask
    loss = loss.mean(1)

    # take the average over all masks
    return loss.sum() / len(inputs)
```

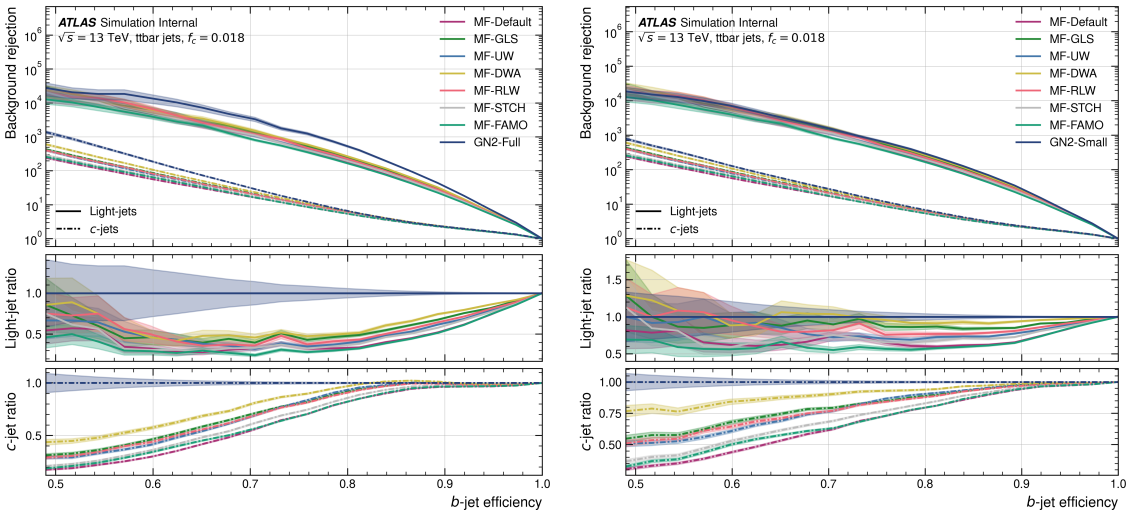
We then train four MaskFormers with this new loss function: two with the Default weighting ($\epsilon = 1$, $\epsilon = -1$), one with DWA weighting, and one with GLS scalarisation and compare their performance to the regular cross-entropy loss models.

5.6 Results

5.6.1 Jet Classification

The majority of loss-based weighting methods are comparable to or better than the default baseline in jet classification.[3] The only methods that represent a drop in performance from the default baseline are FAMO in c-jet rejection and STCH at certain b-jet efficiencies. In [79], the FAMO weighting algorithm surpassed three of the loss-based methods discussed in our work when applied to the NYUv2 segmentation and depth estimation dataset [103]. However, in our MaskFormer

[3]The jet classification task is tested using the `Salt` evaluation script found in commit hash `6c6fbc09`.



(a) Benchmarked against a full-sized GN2 model.

(b) Benchmarked against a small GN2 model.

Figure 5.10: Jet classification ROC performance curve as a function of b-jet efficiency for loss-based task weighting methods.

model, FAMO consistently falls behind all other schemes in light-jet rejection, despite having a similar weight update logic to DWA. STCH also represents a less favourable result, which could be an effect of multi-objective optimisation (MOO) algorithms not necessarily translating favourably to the MTL paradigm [97]. Both of these methods only improve upon the default MaskFormer in c-jet rejection, and that too at a limited range of b-jet efficiencies between 55% and 68%.

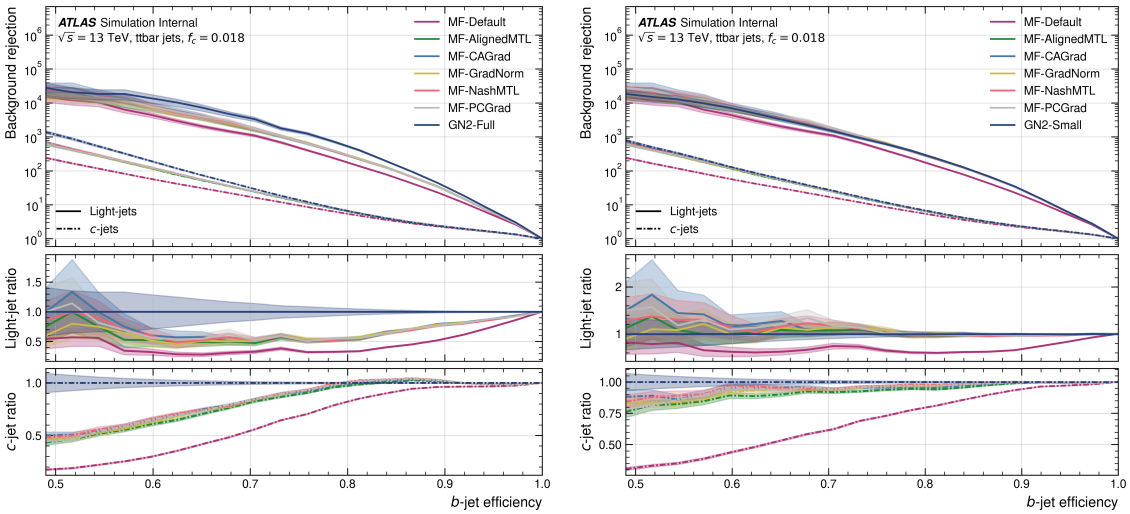
We find that DWA represents the highest improvement in jet classification, with a maximum 53.2% increase in light-jet rejection (77% operating point), and an 87% increase in c-jet rejection at the 60% operating point as shown in Table B.1. At lower efficiencies, DWA achieves close to GN2-Full performance and surpasses its smaller counterpart. GLS leads to a roughly identical performance increase to DWA in light-jet rejection. Though not as stark an improvement, GLS also increases significantly above baseline performance in c-jet rejection at all b-jet efficiencies. This is especially remarkable given the simplicity of the GLS weighting, which uses no dynamic logic, uses unitary weights, and requires no hyperparameter tuning. u-jet and c-jet rejection performance for the four key working points are summarised in Table 5.1.

In general, gradient-based methods offer much higher light-jet and c-jet rejection compared to loss-based methods. At low b-jet efficiencies, CAGrad and PCGrad even outperform a fully trained GN2 model in light-jet rejection (Figure 5.11a). This shows that gradient-based methods have strong potential in jet classification, as our MaskFormer has only 10% of the training data compared to GN2-Full and less than 80% of the model size. This indicates that a full-sized MaskFormer trained with gradient-based weighting could outperform GN2-Full for a greater range of efficiencies and with even higher background rejection. However the increase of performance drops more sharply with increased b-jet efficiency compared to loss-based methods.

Despite overall improvement over baseline with all models, it seems that AlignedMTL falls behind other methods in c-jet rejection and u-jet rejection, with GradNorm following closely behind. The methods which offer the biggest improvement at the 60% working point are CAGrad

	Method	u-jet rejection				c-jet rejection			
		60%	70%	77%	85%	60%	70%	77%	85%
	Default	4385.97	1157.41	316.26	62.38	56.86	17.08	7.58	3.27
Loss-based	GLS	5952.39	1449.28	457.88	84.18	87.21	22.22	8.69	3.51
	UW	6410.27	1173.71	355.37	74.24	77.70	21.24	8.89	3.54
	DWA	6172.85	1618.13	484.50	93.63	106.56	25.19	9.37	3.62
	RLW	6666.68	1333.34	394.01	78.91	82.43	21.43	8.63	3.50
	STCH	4273.51	1207.73	334.67	73.00	68.45	19.03	7.97	3.38
	FAMO	3875.98	863.56	290.87	59.61	64.23	17.51	7.58	3.28
Grad-based	AlignedMTL	6413.35	1650.96	513.07	99.37	113.76	25.57	9.48	3.61
	CAGrad	**7940.33**	1852.74	529.36	**103.25**	122.19	**26.46**	9.62	3.67
	GradNorm	6669.88	1832.38	**539.63**	98.38	118.71	26.20	9.77	3.67
	NashMTL	7249.87	1873.56	536.16	97.17	**123.19**	26.30	9.81	**3.68**
	PCGrad	**7940.33**	**1961.73**	514.65	97.23	121.92	26.32	**9.82**	3.67
	GradVac	6410.27	1543.21	451.67	93.74	100.71	24.57	9.40	3.62
PolyLoss	Default ($\epsilon = 1$)	5952.39	1449.28	428.45	83.21	86.76	22.25	8.66	3.46
	Default ($\epsilon = -1$)	6944.46	1587.3	500.5	94.48	98.39	23.06	8.87	3.51
	DWA ($\epsilon = -1$)	7575.77	1773.05	525.76	97.81	112.54	25.84	9.57	3.63
	GLS ($\epsilon = -1$)	6944.46	1515.15	456.62	85.12	87.81	21.76	8.52	3.44

Table 5.1: u-jet and c-jet rejection for all task weighting methods at 60%, 70%, 77% and 80% operating points.

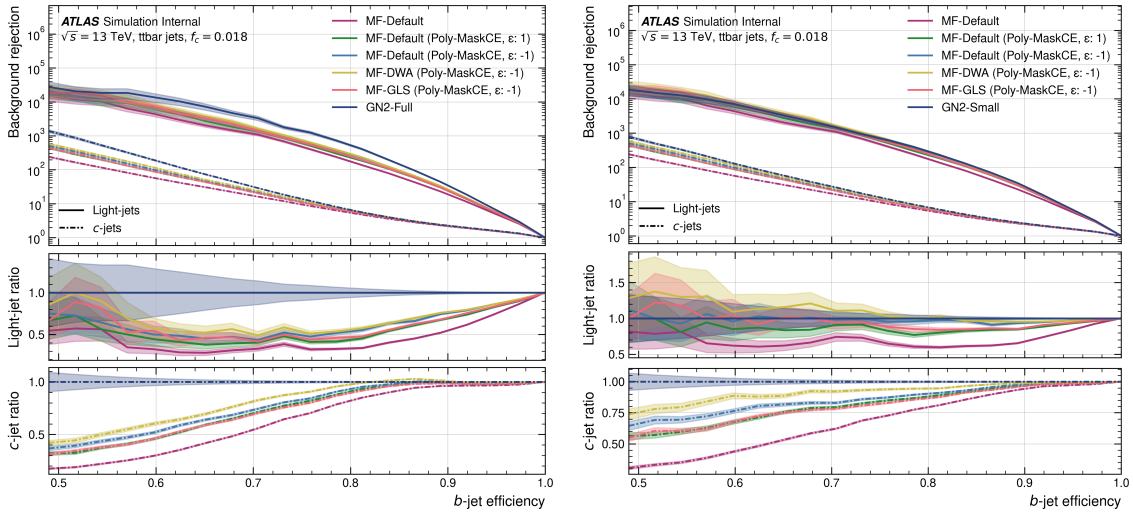


(a) Benchmarked against a full-sized GN2 model.

(b) Benchmarked against a small GN2 model.

Figure 5.11: Jet classification ROC performance curve as a function of b-jet efficiency for various gradient-based task weighting methods.

and PCGrad with both models showing an 81% improvement in u-jet rejection and 114-115% increase in c-jet rejection.



(a) Benchmarked against a full-sized GN2 model.

(b) Benchmarked against a small GN2 model.

Figure 5.12: Jet classification ROC performance curve as a function of b-jet efficiency for various gradient-based task weighting methods.

PolyLoss models are found to have improved rejections compared to their standard CE-loss counterparts. Only light-jet rejections are discussed at the 60% working point in the interest of brevity, however these performance increases are reflected across all working points for the majority of PolyLoss models.

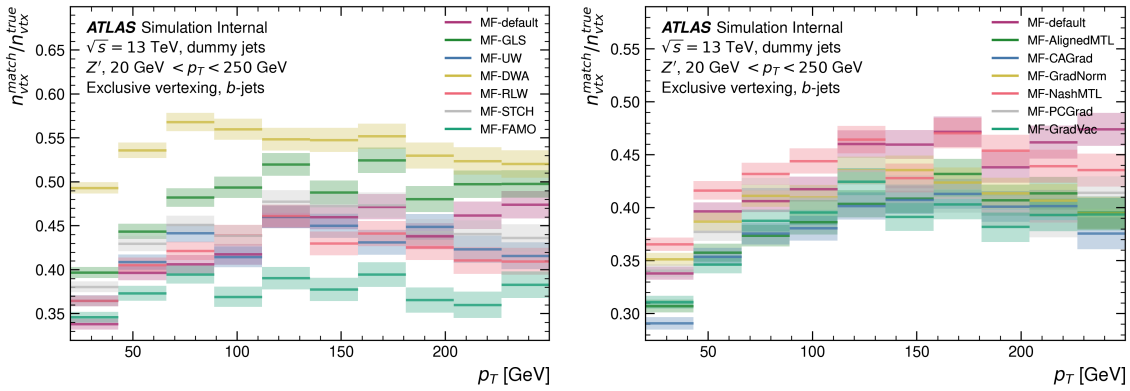
We first analyse a PolyLoss model with default weighting and $\epsilon = 1$, and find that it achieves the same performance increase as the GLS model (with a standard CE-loss function). By changing the perturbation hyperparameter from 1 to -1, it is evident that even greater performance increases are possible: light-jet rejection increases by 58% when compared with the default weighting model trained with standard CE-loss.

Furthermore, when combining PolyLoss *and* GLS weighting (Poly-GLS), there is a light-jet rejection increase of 17% over the standard GLS model. For Poly-DWA, the model achieves the second-best c-jet and light-jet rejection across all models, despite being much less involved and time-consuming than gradient-based models. Thus, the incorporation of PolyLoss serves as a performance booster, whether used with static weights or in conjunction with an appropriate task weighting algorithm.

5.6.2 Vertex Finding

The masks outputted by the model are used to assign input tracks to output vertices. There are $N = 5$ masks (one mask for each possible output vertex) of length M (number of possible tracks tracks associated to the vertex). A value of 1 in the i-th element of mask j means that the i-th track is assigned to vertex j, while a 0 means no assignment. For example, if there is a value of 1 in the 3rd element ($i = 3$) of the first mask ($j = 1$), this means that the third input track is associated to the first output vertex.

Two vertexing approaches are considered: inclusive and exclusive vertexing. Inclusive vertexing, such as SV1, aims to reconstruct a single inclusive vertex within the jet. To achieve this, tracks from the b-hadron decay are grouped with tracks from tertiary $b \to c$ decays. Exclusive vertexing, on the other hand, aims to resolve each individual displaced vertex in the jet. In order to fairly compare the performance of our models against a benchmark of SV1, inclusive vertex finding is studied as well as exclusive vertexing (without a reference). Naturally, inclusive vertexing is easier than exclusive vertexing, and this is reflected in the significantly higher inclusive vertexing efficiencies.



(a) Loss-based weighting methods.

(b) Gradient-based weighting methods.

Figure 5.13: Exclusive vertex finding efficiency as a function of p_{T} for b-jets evaluated on a Z' sample.

The vertexing task is evaluated on a Z' sample as it provides clearer visualisation of model performance.[4] For loss-based methods, there seems to be a high correlation between jet classification and vertex finding performance, as DWA and GLS are once again consistently best-performing across the full p_{T} range for exclusive vertexing, as demonstrated in Figure 5.13a. DWA in particular demonstrates a significant efficiency increase at low p_{T}, improving upon the default baseline by roughly 14-17% for $p_{\mathrm{T}} < 100$ GeV. At the lowest p_{T} range, all loss-based weighting methods, even those considered poorly performing in jet classification (STCH and FAMO), improve upon the baseline. However, the improvements in performance are less stark at higher p_{T}, with methods like RLW and UW falling behind the baseline in the $p_{\mathrm{T}} > 100$ GeV range.

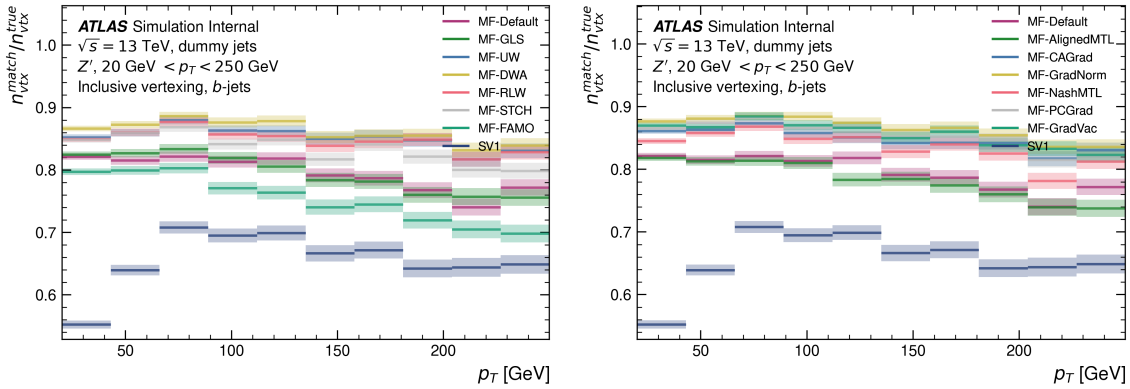
Surprisingly, gradient-based methods are not nearly as successful in improving the exclusive vertexing task. In fact, some methods such as GradVac, CAGrad and AlignedMTL consistently perform worse than the default weighting across the full p_{T} range. Notably, the aforementioned methods provided some of the highest jet classification rejection rates, surpassing GN2-Full at low efficiencies. The only method which surpasses the performance of MF-Default here at multiple p_{T} ranges is NashMTL, however the increase is only on the order of 2-3%.

This is intriguing, as we do not observe significant gradient conflict between jet classification and the two mask losses that contribute to the vertex finding task output (Mask CE and Mask Dice). However, as discussed in Section 5.2.1, this may be a case where even a low positive gradient cosine similarity is harmful to the model. Nonetheless, if that were the case, we would

[4]The vertexing task and subsequent tasks in this section are tested using the latest `Salt` evaluation script which can be found in commit hash `5d8f5c20`.

expect GradVac to perform well on this task as it aims to tackle the very issue of conflicting gradients arising from low positive cosine similarity.

Another reason that this task is less effectively learned by gradient-based methods may be the conflict between mask losses in the first half of training. It was proposed earlier that this may be a synergistic conflict, which allows the model to better explore the Pareto front. From this angle, the impaired performance on this task can be explained from an intuitive perspective. Because gradient-based methods seek to mitigate conflicting gradients, they may naively force an artificial alignment between the mask losses, potentially disrupting the natural balance that allows them to complement each other effectively. More gradient tests need to be conducted with these two losses to confirm this hypothesis and understand if their conflict does indeed lead to improved performance.



(a) Loss-based weighting methods.

(b) Gradient-based weighting methods.

Figure 5.14: Inclusive vertex finding (reconstructing a single vertex within the jet) efficiency as a function of p_{T} for b-jets evaluated on a Z' sample.

During inclusive vertexing evaluation, performance increases are also noted. All methods outperform SV1, with DWA, UW, RLW and even STCH improving upon the default baseline. GLS is not as successful in inclusive vertexing, however, given that it excels at the more difficult exclusive vertexing, this is not necessarily a bad result. Gradient-based methods offer slightly higher exclusive vertexing efficiency when compared with loss-based methods, and all methods except GradVac outperform the default baseline.

5.6.3 Track Origin Prediction

To measure the track origin classification performance, the area under the curve (AUC) of the ROC curve is computed for each origin class (see Table 4.1). We perform this classification using a one-vs-all approach, where we take one class as signal and the rest as background, for each class. Tracks originating from c-hadron decay are most difficult to classify across all models (see Appendix D.1), possibly due to similarities with certain b-hadron species' tracks. In general, we find no significant performance differences across weighting algorithms, with all models achieving AUC > 0.9, except FAMO which shows degraded performance across all classes.

The AUC, Precision, Recall and F1 scores for different truth origins are averaged using a weighted mean, which weights each class score by the relative number of tracks with that class

label. Precision and Recall are calculated using true positives (TP), false positives (FP) and false negatives (FN):

$$\text{Precision} = \frac{TP}{TP + FP} \tag{5.20}$$

$$\text{Recall} = \frac{TP}{TP + FN} \tag{5.21}$$

The F1 score then combines these two metrics as below,

$$\text{F1 Score} = 2 \cdot \frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}} \tag{5.22}$$

We provide a summary of the lowest and highest weighted mean values for the three broad approaches studied in our work in Table 5.2.

	AUC		**Precision**		**Recall**		**F1**	
Group	Low	High	Low	High	Low	High	Low	High
Loss-based	0.92	0.94	0.8	0.82	0.47	0.54	0.58	0.63
Gradient-based	0.94	0.94	0.82	0.82	0.53	0.54	0.63	0.63
PolyLoss	0.93	0.93	0.82	0.94	0.52	0.54	0.62	0.63

Table 5.2: Weighted averages of AUC, Precision, Recall and F1 for the three approaches investigated, showing the lowest and highest values for each metric within each group.

Excluding FAMO, to two decimal places, all models have the same weighted mean AUC and Precision values. As the different weighting methods have minimal effect on this task, we will not analyse the results further as objective model performance is outside the scope of this thesis.

5.6.4 Hadron Classification

To analyse vertex (hadron) classification performance, we first analyse the models' ability to correctly differentiate "valid" (non-NULL) hadrons from padded hadrons, which can be seen in Table 5.3. This is synonymous to b-jet efficiency used for ROC calculation, however in the context of regression we do not use a discriminant $\mathcal{D}_b$. Instead, we simply choose the class with $p > 0.5$ from the per-class probability output vector.

Next, we look at the classification accuracy. In collisions, c-hadrons are produced around 2.2 times more than b-hadrons. Thus, if a model classifies more b hadrons correctly these make up a smaller portion of the overall correctly classified hadrons in the dataset. This is shown in Table 5.4. Again, we find that GLS and the Default weighting scheme perform best on hadron classification, with DWA performing the worst. Despite poor performance in jet classification, FAMO shows the second best performance amongst loss weighting methods, indicating that the algorithm may have caused the model to over-emphasise regression performance at the expense of flavour tagging. However, we also note that RLW has very similar performance to FAMO, suggesting that FAMO is not weighting regression any better than choosing at random.

While DWA proved to have the highest improvement in jet classification, hadron classification performance drops quite sharply compared to the default weighting scheme. The fraction of

		TPR (%) ↑	FPR (%) ↓	FNR (%) ↓	TNR (%) ↑
	Default	71.06	6.25	28.94	93.75
Loss-based	GLS	70.11	7.01	29.89	92.99
	STCH	32.16	16.53	67.84	83.47
	UW	53.12	10.62	46.88	89.38
	DWA	28.68	18.42	71.32	81.58
	RLW	55.81	9.98	44.19	90.02
	FAMO	56.75	8.53	43.25	91.47
Grad-based	AlignedMTL	55.33	9.88	44.67	90.12
	CAGrad	52.29	10.03	47.71	89.97
	GradNorm	38.70	14.07	61.30	85.93
	NashMTL	39.28	14.40	60.72	85.60
	PCGrad	38.74	14.24	61.26	85.76
PolyLoss	Default ($\epsilon = 1$)	**72.29**	**5.23**	**27.71**	**94.77**
	Default ($\epsilon = -1$)	71.65	6.03	28.35	93.97
	DWA ($\epsilon = -1$)	33.52	16.60	66.48	83.40
	GLS ($\epsilon = -1$)	70.58	6.62	29.42	93.38

Table 5.3: Binary hadron classification (bc vs NULL) predictions for different weighting methods. False positives are when the model has $p_b > 0.5$ or $p_c > 0.5$ when the true hadron class is NULL, and the converse is true for false negatives.

		bc hadron flavour accuracy (%) ↑	b hadron flavour accuracy (%) ↑	c hadron flavour accuracy (%) ↑
	Default	60.62	70.49	56.07
Loss-based	GLS	62.21	**77.42**	55.20
	STCH	7.66	3.04	9.79
	UW	38.60	59.33	29.04
	DWA	16.81	5.50	22.03
	RLW	18.36	1.58	26.09
	FAMO	51.80	72.73	42.15
Grad-based	AlignedMTL	51.00	69.53	42.46
	CAGrad	46.37	59.93	40.12
	GradNorm	28.26	4.75	39.10
	NashMTL	31.80	67.83	15.19
	PCGrad	28.88	4.99	39.89
PolyLoss	Default ($\epsilon = 1$)	61.55	69.02	58.11
	Default ($\epsilon = -1$)	**63.72**	73.63	**59.15**
	DWA ($\epsilon = -1$)	21.27	5.46	28.56
	GLS ($\epsilon = -1$)	61.49	77.41	54.14

Table 5.4: Hadron flavour classification predictions for different weighting methods. The class probabilities are converted to predictions as $p_{\text{class}} > 0.5$.

correctly identified valid hadrons is only 28.68% with only 5.5% of b-hadrons being classified correctly.

We know from Table 5.3 that dynamic models like DWA and FAMO predict very few valid hadrons correctly. For gradient-based methods, we find a similar drop in performance for vertex classification. While AlignedMTL classifies valid hadrons more accurately than other gradient-based methods, the model only correctly identifies b and c hadrons in more than half of all instances. When compared with the default weighting, performance is severely degraded; as expected, the inverse relationship between jet classification and object regression performances is evident here. However, despite similar flavour tagging performance for PCGrad and CAGrad, the former performs much worse at hadron classification.

5.6.5 Vertex Regression

We now turn our attention to the regression performance by looking at the residuals' mean and standard deviation, shown for all five regressed variables in Table 5.5.

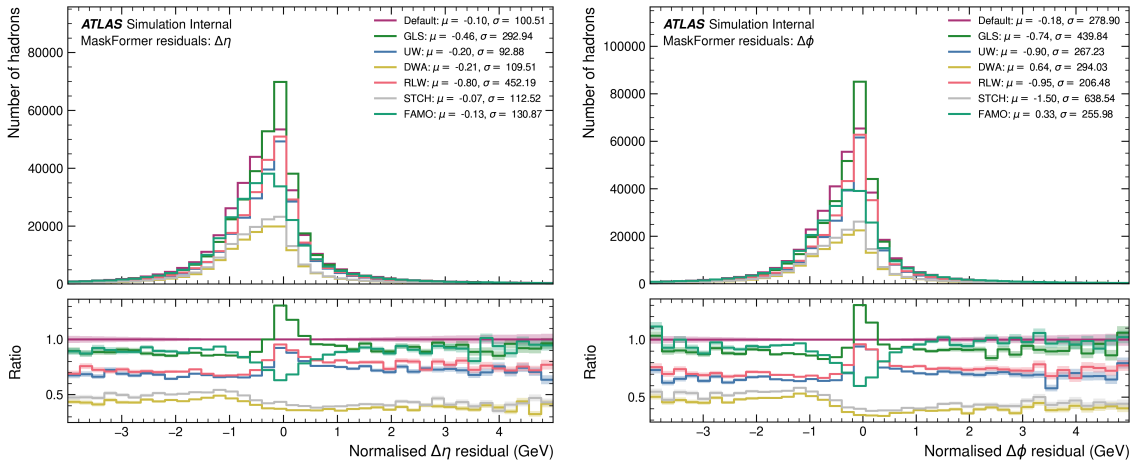
		$\mu_{\Delta \mathrm{m}}$	$\sigma_{\Delta \mathrm{m}}$	$\mu_{\Delta p_{\mathrm{T}}}$	$\sigma_{\Delta p_{\mathrm{T}}}$	$\mu_{\Delta L_{xy}}$	$\sigma_{\Delta L_{xy}}$	$\mu_{\Delta d\eta}$	$\sigma_{\Delta d\eta}$	$\mu_{\Delta d\phi}$	$\sigma_{\Delta d\phi}$
	Default	0.02	0.36	0.07	0.46	0.05	0.99	-0.12	96.86	-0.18	278.9
Loss-based	GLS	0.03	0.38	0.08	0.44	**0.04**	0.88	-0.12	83.43	-0.74	439.84
	STCH	**0.0**	0.25	0.09	0.47	0.07	0.99	-0.31	**47.08**	-1.5	638.54
	UW	0.03	0.37	**0.06**	0.42	**0.04**	0.84	-0.05	113.69	-0.9	267.23
	DWA	0.02	0.27	0.08	0.46	0.07	0.95	-0.14	110.15	0.64	294.03
	RLW	0.02	0.36	0.07	0.44	0.07	1.16	-0.3	208.87	-0.95	206.48
	FAMO	0.03	0.3	0.08	0.5	0.07	0.85	-0.16	130.06	0.33	255.98
Grad-based	AlignedMTL	0.03	0.32	0.12	0.7	0.05	0.86	-0.06	130.72	**-0.01**	188.65
	CAGrad	0.03	0.28	0.11	0.72	0.05	0.91	**0.03**	124.74	0.09	201.21
	GradNorm	-0.01	**0.23**	0.12	0.63	0.07	1.02	-0.15	127.57	-0.02	283.58
	NashMTL	0.06	0.39	0.11	0.74	0.05	1.19	0.04	146.84	-0.88	426.69
	PCGrad	**-0.0**	0.24	0.12	0.84	0.08	1.29	-0.15	109.57	-0.03	403.53
	GradVac	**0.0**	0.23	0.13	0.8	0.07	0.97	-0.23	97.67	0.24	297.87
PolyLoss	$\mathrm{Default}_{\epsilon=1}$	0.03	0.37	0.07	0.45	0.05	1.01	-0.42	186.4	-0.19	234.18
	$\mathrm{Default}_{\epsilon=-1}$	0.02	0.34	**0.06**	**0.41**	**0.04**	**0.72**	-0.16	170.62	-0.21	203.26
	$\mathrm{DWA}_{\epsilon=-1}$	**0.0**	0.24	0.09	0.5	0.07	0.84	-0.2	111.72	-0.2	312.17
	$\mathrm{GLS}_{\epsilon=-1}$	0.04	0.4	0.07	0.44	**0.04**	1.1	-0.63	322.1	-0.23	**153.99**

Table 5.5: Normalised regression residual means and standard deviations for all task weighting methods. L_{xy} residuals are for $L_{xy} > 1$ mm.

As can be seen in Table 5.5, it is difficult to ascertain which model is better at the task of regression as a whole, with different models performing better on different regression outputs. Overall, based on the relative abundance of best regressed outputs (ie. mean and standard closest to 0), PolyLoss models seem to perform best at regression, followed by loss-based weightings then gradient-based weightings.

In the interest of brevity, for the three broad approaches in our work (loss-based methods, gradient-based methods and loss modification), we present a subset of regression residual graphs. The remaining graphs for all our methods are displayed in Appendix E.

Angular variables ($d\phi, d\eta$) seem to be most difficult for the model to regress, with very high standard deviation of the residuals. This makes sense, given that these two variables have very



(a) Pseudorapidity difference $d\eta$ residuals $(d\eta_{\text{truth}} - d\eta_{\text{predicted}})/d\eta_{\text{truth}}$.

(b) Azimuthal angle difference residuals $(d\phi_{\text{truth}} - d\phi_{\text{predicted}})/d\phi_{\text{truth}}$.

Figure 5.15: Normalised $d\eta$ (left) and $d\phi$ (right) residuals for the five leading p_{T} hadrons. The predictions are regressed from MaskFormer models trained with loss-based weighting methods, with the residual's mean and standard deviation displayed in the legend for each method.

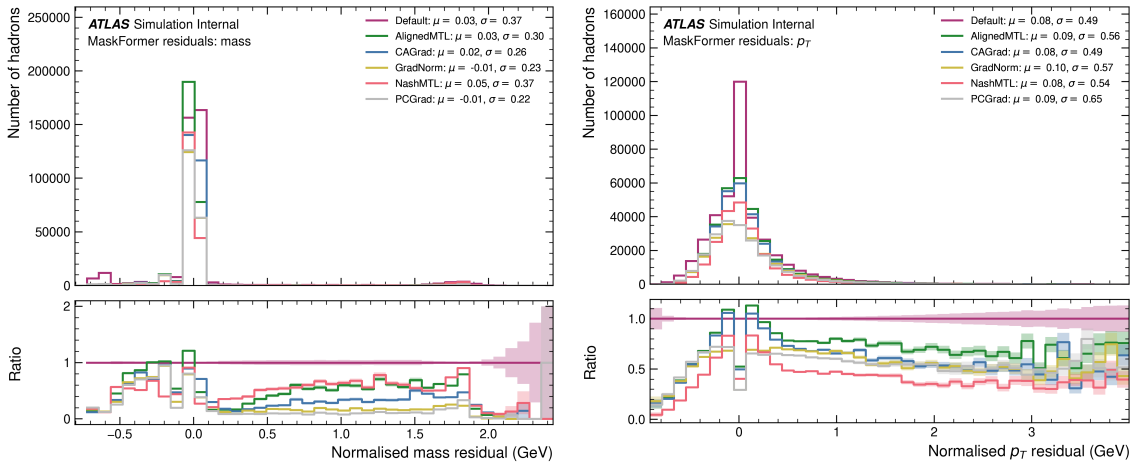
low mean truth values relative to their standard deviation. For the pseudorapidity difference, for example, $\mu_{d\eta} = 9.58 \times 10^{-5}$ and $\sigma_{d\eta} = 0.066$. The ratio of these two values is 685.89. Thus, this may lead the model to struggle to correctly predict these variables because even small errors in the prediction can result in relatively large residuals when normalised by the truth values that are magnitudes smaller.

For mass, p_{T} and L_{xy}, the models seem to over-estimate the target value. The majority of gradient-based weightings seem to underestimate the hadron's mass (Figure 5.16a), with only NashMTL and CAGrad being skewed towards a higher mass prediction; these two models also have the lowest mean mass residuals, however conversely, they also have the highest standard deviation values too. Of all gradient-based methods, NashMTL is the poorest predictor of p_{T}.

These results are especially interesting given that regression does not have a conflicting gradient cosine similarity with jet classification. However, there are clear imbalances in gradient magnitude as seen in 5.2a. More interestingly, regression loss curves are minimally affected by the choice of task weighting algorithm. This leads to the question of how a task that is so insensitive to task weighting from a loss perspective has test results that vary greatly with different models.

One possible argument is that the rate of change of regression loss is lower than for other tasks. Regression has the fastest convergence rate across all tasks, with low change in losses from early on in training, making the loss curve look almost flat after the first few iterations. This may mean that for dynamic loss methods, prioritising classification and mask tasks results in very small absolute changes in regression loss, thereby resulting in low regression performance. However, the FAMO algorithm has the highest regression loss during training (between 13% and 15% higher than for GLS), however it achieves significantly higher performance in valid hadron identification. Thus, loss curves only offer partial insight into this issue.

Another argument may be that this is related to the very low gradient similarity between regression and all other tasks. In Section 5.2.1 and 5.6.3, it was suggested that a very low positive



(a) Mass residuals $(m_{\text{truth}} - m_{\text{predicted}})/m_{\text{truth}}$.

(b) p_{T} residuals $(p_{\text{T}_{\text{truth}}} - p_{\text{T}_{\text{predicted}}})/p_{\text{T}_{\text{truth}}}$.

Figure 5.16: Normalised mass (left) and p_{T} (right) residuals for the five leading p_{T} hadrons. The predictions are regressed from MaskFormer models trained with gradient-based weighting methods, with the residual's mean and standard deviation for each method displayed in the legend.

cosine similarity may still cause conflict in the model. Looking at the regression results presented here further strengthens this argument, and also may explain why the default weighting scheme suppresses the regression task's gradient magnitude during training. We hypothesise that in the best case scenario, regression is mildly harmful to other tasks, and its degradation contributes to the performance of jet classification and vertexing.

Chapter 6

Conclusions

6.1 Summary

In this work, we probe into the ATLAS MaskFormer multi-task learning (MTL) model to investigate the occurrence of negative transfer between tasks and to mitigate its effect through various task weighting methods.

It is found that the MaskFormer model does not suffer from any significant conflicting gradients except between the Mask Dice and Mask CE tasks during the first half of training; however, given the synergy that these tasks have been proven to display when implemented jointly, we theorise that this conflict may help the model better explore the Pareto frontier. However, this is just a hypothesis, and further investigation would be needed to determine this.

We note that regression is almost fully orthogonal to all other tasks, with a cosine similarity close to 0. This finding leads us to suggest that even a low positive cosine similarity value may be harmful to the model, based on the inverse correlation between jet classification and regression performance.

We then implement Random Loss Weighting (RLW), which assigns random weights to each task's loss; Geometric Loss Strategy (GLS), which uses a geometric product scalarisation; Fast Adaptive Multi-Task Optimisation (FAMO), which seeks the largest worst-case improvement rate across all tasks; Smooth Tchebysheff Scalarisation (STCH), which applies a multi-objective optimisation scalarisation to MTL; Uncertainty Weighting (UW), which weights tasks based on their aleatoric (intrinstic) uncertainty; and Dynamic Weight Average (DWA), which adjusts weights dynamically based on the rate of loss change between epochs for each task.

DWA significantly increases flavour tagging and vertex finding performance, however degrades significantly in regression. Across all tasks, GLS seems to perform most successfully despite its simple yet efficient implementation. This may be due to its loss scale invariance; by multiplying losses together instead of summing them, GLS effectively ignores scale differences between models.

For gradient-based methods, we explore NashMTL, which formulates MTL as a multi-player bargaining game and seeks Nash equilibrium; AlignedMTL, which aligns principal components of the gradient matrix to mitigate gradient dominance and conflicting directions; Projecting Conflicting Gradients (PCGrad), which projects conflicting gradients onto the normal plane of each other; and Conflict-Averse Gradient Descent (CAGrad), which seeks to maximise the average total gradi-

ent decrease while also decreasing each task gradient; Gradient Vaccine (GradVac) which softens the conflicting gradients condition proposed in PCGrad, and Gradient Normalisation (GradNorm), which encourages all to tasks train at similar rates.

Gradient-based methods seem to significantly boost the performance of the jet classification task in MaskFormer at the expense of regression tasks. However, this does not preclude the use of these methods for our model. A 3× increase in light-jet rejection at 50% b-jet efficiency is achieved above the baseline model (see Appendix B). This improvement is valuable to the overall MaskFormer model despite degradation in regression. As mentioned previously, jet classification lies at the core of flavour tagging algorithms, and in cases where a Pareto-optimal solution is not achievable, treating the model as an auxiliary task learning model is feasible. Thus, gradient-based methods can be employed as an auxiliary task weighting method, boosting the jet classification task, while the regression task can be improved upon in future work.

One of the issues with gradient-based methods, however, is the increase in training time, which scales linearly with the number of tasks. Additionally, Pytorch Lightning does not function well with automatic mixed precision when using manual optimisation to directly modify gradients. This means that gradient-based methods can only train with full 32-bit precision, which can only be done on full GPUs, as opposed to more resource-efficient Multi-Instance GPUs. Thus, outside of their poor performance on regression, the implementation of these methods also suffers from additional constraints and limitations that negatively affect the training process.

We find that track origin prediction is unaffected by the use of different weighting algorithms. One potential explanation is that this task gets stuck in a local minimum early on in the training process. However, the overall high performance on this task suggests otherwise. Another explanation is that it is an easy task, and is therefore robust against changes in its weight impacting its performance. The track origin loss gradient was also found to have a high cosine similarity to jet classification. This makes sense, given the strong connection between the process from which a track originates from and the flavour of the jet (for example, a track originating from a b-hadron would be expected in heavy-flavour jets only).

In general, no single model evaluated in our work represents a definitive superior method for weighting tasks in the MaskFormer MTL network. Instead, different algorithms seem to represent different optimisation tradeoff points between tasks. As such, these methods can be used to one's advantage, depending on the desired outcome or task that is more important to optimise.

The use of modified loss functions, such as Poly-1 Loss, has promising results when compared with current baselines and can "boost" whatever positive effects are obtained from the chosen weighting methods it was evaluated on, namely GLS and DWA.

6.2 Future Work

Our findings show that the Geometric Loss Strategy (GLS) demonstrates a promising ability to moderately improve upon all tasks within the model without degrading any one task. Given the importance of the b-tagging task in particular, an extension of GLS called the Focused Learning Strategy (FLS) could be implemented in future to prioritise specific tasks, such as jet classification:

$$L_{\text{FLS}} = \prod_{i=1}^{K} \sqrt[K]{L_i} \times \prod_{j=1}^{m} \sqrt[m]{L_j} \tag{6.1}$$

where m ($\leq K$) important tasks are given more weight. This weighting method could be used depending on the task we want to optimise for.

Future work could investigate the set of all optimal solutions, or the Pareto set, for MaskFormer from a more theoretical perspective. Task-specific benchmarks can be produced as a practical alternative to visualising the six-dimensional Pareto front. In future, six single-task learning (STL) models can be trained for each MaskFormer task and then compared against the task performance for different MTL weighting methods. Furthermore, ablation studies could also be conducted to investigate an appropriate choice of tasks, and to confirm whether regression is indeed harmful to other tasks as we propose in our work.

One aspect of the model that was not addressed in our work is the Hungarian matcher weights. As of now, the convention for the ATLAS MaskFormer is to simply set the matcher weights to the loss weights. However, there is little evidence to prove the efficiency of assigning matcher weights in this fashion. In general, the literature on Hungarian matcher weights is sparse. Most works either assign them based on domain knowledge of the underlying problem, or treat them as a hyperparameter and set them seemingly arbitrarily [17, 23]. Future research could build upon our work by leveraging the loss weights derived from weighting algorithms and using them in the matching process. This could be done by feeding them back into the Hungariain matching step in the MaskFormer decoder, creating a dynamic Hungarian matcher that responds to changes in loss or gradient magnitude.

Given the high computational cost of gradient-based methods, approximate gradient-based methods have been proposed, such as MGDA-UB for encoder-decoder MTL models [101]. Such methods compute the gradients of the task losses with respect to the shared representations (from the encoder) instead of the shared parameters, such that the optimisation problem becomes

$$\min_{\alpha_1,\ldots,\alpha_K} \left\{ \left\| \sum_{i=1}^{K} \alpha_i \nabla_{\mathbf{z}} L_i(\theta_{\text{sh}}, \theta_i) \right\|_2^2 \right\} \text{ s.t. } \sum_{i=1}^{K} \alpha_i = 1,\ \alpha_k \geq 0\ \forall k \tag{6.2}$$

where $\nabla_{\mathbf{z}} L_i(\theta_{\text{sh}}, \theta_i)$ can be computed with just one backward pass.

Unlike traditional encoder-decoder multi-task learning algorithms, MaskFormer introduces a more complex architecture, where different tasks share more parameters than others; for example, object classification and object regression (vertex fitting) share the encoder and decoder, whereas these tasks only share the encoder layers with jet classification. Thus, this leads to asymmetrically "shared" representations. This may pose challenges for MGDA-UB-based weighting methods such as GradDrop [22] as the representations are not unified through a single encoder structure, and therefore the concept of shared representations is more ambiguous than in traditional architectures. However, we believe it is still an interesting avenue to pursue for future work, as it can replicate the strong performance improvements found with gradient-based methods with a fraction of the computational cost.

Additionally, Kurin et al. [68] demonstrated that gradient-based algorithms are so effective because they implicitly regularise the model. The authors then showed that if regularisation

is tuned carefully, a model with unitary scalarisation (ie. an unweighted sum of task losses) can match the performance of gradient-based methods. Xin et al. [113] showed that instead unitary scalarisation methods have the potential to perform as well as dynamic methods, but lack the proper tuning. Thus, as gradient-based methods greatly improve upon jet classification performance, in future we believe they can serve as benchmarks for how well our MaskFormer model *could* perform with adequately tuned model parameters and regularisation.

Bibliography

[1] Morad Aaboud, Georges Aad, Brad Abbott, B Abeloos, DK Abhayasinghe, SH Abidi, OS AbouZeid, NL Abraham, H Abramowicz, H Abreu, et al. 2018. Observation of h→ bb decays and vh production with the atlas detector. *Physics Letters B*, 786:59–86.

[2] Georges Aad, Tatevik Abajyan, Brad Abbott, Jalal Abdallah, S Abdel Khalek, Ahmed Ali Abdelalim, R Aben, B Abi, M Abolins, OS AbouZeid, et al. 2012. Observation of a new particle in the search for the standard model higgs boson with the atlas detector at the lhc. *Physics Letters B*, 716(1):1–29.

[3] Georges Aad, Xabier Sebastian Anduaga, S Antonelli, M Bendel, B Breiler, F Castrovillari, JV Civera, T Del Prete, Maria Teresa Dova, S Duffin, et al. 2008. The atlas experiment at the cern large hadron collider.

[4] Georges Aad et al. 2019. ATLAS b-jet identification performance and efficiency measurement with $t\bar{t}$ events in pp collisions at $\sqrt{s} = 13$ TeV. *Eur. Phys. J. C*, 79(11):970.

[5] Fereshteh Akbari, Mehrdad Ghaznavi, and Esmaile Khorram. 2018. A revised pascoletti–serafini scalarization method for multiobjective optimization problems. *Journal of Optimization Theory and Applications*, 178:560–590.

[6] Emine Aşar and Atilla Özgür. 2024. Effect of polyloss function on steel defect detection. In *Steel 4.0: Digitalization in Steel Industry*, pages 143–166. Springer.

[7] Robert J Aumann and Sergiu Hart. 1992. *Handbook of game theory with economic applications*, volume 2. Elsevier.

[8] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. 2017. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE transactions on pattern analysis and machine intelligence*, 39(12):2481–2495.

[9] Marcel Banner, Roberto Battiston, Ph Bloch, F Bonaudi, K Borer, M Borghini, J-C Chollet, AG Clark, C Conta, P Darriulat, et al. 1983. Observation of single isolated electrons of high transverse momentum in events with missing transverse energy at the cern pp collider. *Physics Letters B*, 122(5-6):476–485.

[10] Jackson Barr, Diptaparna Biswas, Maxence Draguet, Philipp Gadow, Emil Haines, Osama Karkout, Dmitrii Kobylianskii, Wei Lai, Matthew Leigh, Ivan Oleksiyuk, Nikita Pond, Sébastien Rettie, Andrius Vaitkus, Samuel Van Stroud, and Johannes Wagner. 2024. Salt:

Multimodal multitask machine learning for high energy physics. *Journal of Open Source Software*.

[11] Jonathan Baxter. 1997. A bayesian/information theoretic model of learning to learn via multiple task sampling. *Machine learning*, 28:7–39.

[12] Hakan Bilen and Andrea Vedaldi. 2017. Universal representations:the missing link between faces, text, planktons, and cat breeds.

[13] Daniel Bolya, Chong Zhou, Fanyi Xiao, and Yong Jae Lee. 2019. Yolact: Real-time instance segmentation. In *Proceedings of the IEEE/CVF international conference on computer vision*, pages 9157–9166.

[14] Maurice Bourquin, RM Brown, Y Chatelus, JC Chollet, A Degre, D Froidevaux, AR Fyfe, J-M Gaillard, CNP Gee, WM Gibson, et al. 1983. Measurements of hyperon semileptonic decays at the cern super proton synchrotron: Ii. the 1-011-011-01decay modes. *Zeitschrift für Physik C Particles and Fields*, 21(1):1–15.

[15] Andy Buckley, Jonathan Butterworth, Stefan Gieseke, David Grellscheid, Stefan Höche, Hendrik Hoeth, Frank Krauss, Leif Lönnblad, Emily Nurse, Peter Richardson, et al. 2011. General-purpose event generators for lhc physics. *Physics Reports*, 504(5):145–233.

[16] Daniele Calandriello, Alessandro Lazaric, and Marcello Restelli. 2014. Sparse multi-task reinforcement learning. *Advances in neural information processing systems*, 27.

[17] Nicolas Carion, Francisco Massa, Gabriel Synnaeve, Nicolas Usunier, Alexander Kirillov, and Sergey Zagoruyko. 2020. End-to-end object detection with transformers.

[18] Rich Caruana. 1997. Multitask learning. *Machine learning*, 28:41–75.

[19] Shijie Chen, Yu Zhang, and Qiang Yang. 2024. Multi-task learning in natural language processing: An overview. *ACM Computing Surveys*, 56(12):1–32.

[20] Yaqi Chen, Hao Zhang, Xukui Yang, Wenlin Zhang, and Dan Qu. 2023. Task-based polyloss improves low-resource speech recognition. In *2023 8th International Conference on Signal and Image Processing (ICSIP)*, pages 516–520. IEEE.

[21] Zhao Chen, Vijay Badrinarayanan, Chen-Yu Lee, and Andrew Rabinovich. 2018. Gradnorm: Gradient normalization for adaptive loss balancing in deep multitask networks. In *International conference on machine learning*, pages 794–803. PMLR.

[22] Zhao Chen, Jiquan Ngiam, Yanping Huang, Thang Luong, Henrik Kretzschmar, Yuning Chai, and Dragomir Anguelov. 2020. Just pick a sign: Optimizing deep multitask models with gradient sign dropout. *Advances in Neural Information Processing Systems*, 33:2039–2050.

[23] Bowen Cheng, Ishan Misra, Alexander G Schwing, Alexander Kirillov, and Rohit Girdhar. 2022. Masked-attention mask transformer for universal image segmentation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition*, pages 1290–1299.

[24] Sumanth Chennupati, Ganesh Sistu, Senthil Yogamani, and Samir A Rawashdeh. 2019. Multinet++: Multi-stream feature aggregation and geometric loss strategy for multi-task learning. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition workshops*, pages 0–0.

[25] ATLAS Collaboration. 2019. Atlas b-jet identification performance and efficiency measurement with tt events in pp collisions at. *Eur. Phys. J. C*, 79:970.

[26] ATLAS collaboration, TA collaboration, et al. 2014. Calibration of the performance of b-tagging for c and light-flavour jets in the 2012 atlas data. ATLAS-CONF-2014-046.

[27] ATLAS collaboration et al. 2014. Search for supersymmetry at sqrt (s)= 8 tev in final states with jets and two same-sign leptons or three leptons with the atlas detector. *arXiv preprint arXiv:1404.2500*.

[28] ATLAS collaboration et al. 2017. Identification of jets containing b-hadrons with recurrent neural networks at the atlas experiment. Technical report, ATL-PHYS-PUB-2017-003.

[29] Atlas Collaboration et al. 2017. Optimisation and performance studies of the atlas b-tagging algorithms for the 2017-18 lhc run. Technical report, ATL-PHYS-PUB-2017-013.

[30] ATLAS collaboration et al. 2018. Topological b-hadron decay reconstruction and identification of b-jets with the jetfitter package in the atlas experiment at the lhc. Technical report, ATL-PHYS-PUB-2018-025.

[31] ATLAS collaboration et al. 2020. Deep sets based neural networks for impact parameter flavour tagging in atlas. Technical report, ATL-PHYS-PUB-2020-014.

[32] Atlas Collaboration et al. 2022. Graph Neural Network Jet Flavour Tagging with the ATLAS Detector. Technical report, CERN, Geneva. All figures including auxiliary figures are available at https://atlas.web.cern.ch/Atlas/GROUPS/PHYSICS/PUBNOTES/ATL-PHYS-PUB-2022-027.

[33] Atlas Collaboration et al. 2023. The atlas experiment at the cern large hadron collider: a description of the detector configuration for run 3. *arXiv preprint arXiv:2305.16623*.

[34] Cush. 2023. Example image from wikipedia. `https://en.wikipedia.org/wiki/File:Standard_Model_of_Elementary_Particles.svg`. Accessed: September 17, 2024.

[35] Patrick Czodrowski. 2009. *Improvement of a Data-Driven Method for the Simulation of the $\tau\tau$-Mass Shape in $Z\rightarrow \tau\tau\rightarrow ee+ 4\nu$ for ATLAS at the LHC*. Ph.D. thesis, TU, Dresden (main).

[36] Amit Das, Mark Hasegawa-Johnson, and Karel Veselý. 2017. Deep auto-encoder based multi-task learning using probabilistic transcriptions. In *Interspeech 2017*, pages 2073–2077.

[37] Changyu Deng, Xunbi Ji, Colton Rainey, Jianyu Zhang, and Wei Lu. 2020. Hard vs soft parameter sharing. `https://www.researchgate.net/figure/Illustration-of-Hard-and-Soft-Parameter-Sharing-A-hard-parameter-sharing-B-soft_fig1_346610484`.

[38] Jean-Antoine Désidéri. 2012. Multiple-gradient descent algorithm (mgda) for multiobjective optimization. *Comptes Rendus Mathematique*, 350(5-6):313–318.

[39] Luigi Di Lella and Carlo Rubbia. 2015. The discovery of the w and z particles. In *60 Years of CERN Experiments and Discoveries*, pages 137–163. World Scientific.

[40] Lee R. Dice. 1945. Measures of the amount of ecologic association between species. *Ecology*, 26(3):297–302.

[41] Angelica Lo Duca and Gideon Mendels. 2022. *Comet for Data Science: Enhance your ability to manage and optimize the life cycle of your data science project*. Packt Publishing Ltd.

[42] François Englert and Robert Brout. 1964. Broken symmetry and the mass of gauge vector mesons. *Physical review letters*, 13(9):321.

[43] William Falcon and The PyTorch Lightning team. 2024. Pytorch lightning.

[44] Jiangfan Feng, Etao Tang, Maimai Zeng, Zhujun Gu, Pinglang Kou, and Wei Zheng. 2023. Improving visual question answering for remote sensing via alternate-guided attention and combined loss. *International Journal of Applied Earth Observation and Geoinformation*, 122:103427.

[45] Richard P Feynman. 1988. The behavior of hadron collisions at extreme energies. *Special Relativity and Quantum Theory: A Collection of Papers on the Poincaré Group*, pages 289–304.

[46] Joerg Fliege and Benar F. Svaiter. 2000. Steepest descent methods for multicriteria optimization. *Mathematical methods of operations research*, 51(3):479–494.

[47] Jason Gallicchio and Matthew D Schwartz. 2011. Quark and gluon tagging at the lhc. *Physical review letters*, 107(17):172001.

[48] Simon Graham, Quoc Dang Vu, Mostafa Jahanifar, Shan E Ahmed Raza, Fayyaz Minhas, David Snead, and Nasir Rajpoot. 2023. One model is all you need: multi-task learning enables simultaneous histology image segmentation and classification. *Medical Image Analysis*, 83:102685.

[49] Ryan-Rhys Griffiths, Alexander A Aldrick, Miguel Garcia-Ortegon, Vidhi Lalchand, et al. 2021. Achieving robustness to aleatoric uncertainty with heteroscedastic bayesian optimisation. *Machine Learning: Science and Technology*, 3(1):015004.

[50] C. Grojean. 2016. Higgs Physics. *arXiv preprint arXiv:1708.00794*, pages 143–158. 12 pages, contribution to the CERN in the Proceedings of the 2015 CERN-Latin-American School of High-Energy Physics, Ibarra, Ecuador, 4 - 17 March 2015.

[51] Gerald S Guralnik, Carl R Hagen, and Thomas WB Kibble. 1964. Global conservation laws and massless particles. *Physical Review Letters*, 13(20):585.

[52] FJ Hasert, S Kabe, W Krenz, J Von Krogh, D Lanske, J Morfin, K Schultze, H Weerts, G Bertrand-Coremans, Jean Sacton, et al. 1974. Observation of neutrino-like interactions without muon or electron in the gargamelle neutrino experiment. *Nuclear Physics B*, 73(1):1–22.

[53] Jingzhen He, Junxia Wang, Zeyu Han, Jun Ma, Chongjing Wang, and Meng Qi. 2023. An interpretable transformer network for the retinal disease classification using optical coherence tomography. *Scientific Reports*, 13(1):3637.

[54] Sebastian Heer. 2017. The secondary vertex finding algorithm with the atlas detector. Technical report, ATL-COM-PHYS-2017-1517.

[55] PUMA HEP. 2024. Roc curve example from puma hep. `https://raw.githubusercontent.com/umami-hep/puma/examples-material/roc.png`. Accessed: September 17, 2024.

[56] PUMA HEP. 2024. Roc curve example from puma hep. `https://raw.githubusercontent.com/umami-hep/puma/examples-material/histogram_discriminant.png`. Accessed: September 17, 2024.

[57] Peter W Higgs. 1964. Broken symmetries and the masses of gauge bosons. *Physical review letters*, 13(16):508.

[58] Arpad Horvath. 2006. Lhc diagram. `https://en.wikipedia.org/wiki/Large_Hadron_Collider#/media/File:LHC.svg`. Created: 3 April 2006, Uploaded: 15 November 2009, Licensed under CC BY-SA 2.5. Accessed: September 17, 2024.

[59] Yuzheng Hu, Ruicheng Xian, Qilong Wu, Qiuling Fan, Lang Yin, and Han Zhao. 2024. Revisiting scalarization in multi-task learning: A theoretical perspective. *Advances in Neural Information Processing Systems*, 36.

[60] Qing Huang, Jinfeng Sun, Hui Ding, Xiaodong Wang, and Guangzhi Wang. 2018. Robust liver vessel extraction using 3d u-net with variant dice loss function. *Computers in biology and medicine*, 101:153–162.

[61] Baixin Jin, Pingping Liu, Peng Wang, Lida Shi, and Jing Zhao. 2020. Optic disc segmentation using attention-based u-net and the improved cross-entropy convolutional neural network. *Entropy*, 22(8):844.

[62] Austin Joyce, Bhuvnesh Jain, Justin Khoury, and Mark Trodden. 2015. Beyond the cosmological standard model. *Physics Reports*, 568:1–98.

[63] Patrick Jussel. 2011. *Flavour Tagging in Hadronic B_s^0 Decays for the ATLAS Experiment at the Large Hadron Collider*. Ph.D. thesis, Innsbruck U.

[64] Refail Kasimbeyli, Zehra Kamisli Ozturk, Nergiz Kasimbeyli, Gulcin Dinc Yalcin, and Banu Icmen Erdem. 2019. Comparison of some scalarization methods in multiobjective optimization: comparison of scalarization methods. *Bulletin of the Malaysian Mathematical Sciences Society*, 42:1875–1905.

[65] Alex Kendall, Yarin Gal, and Roberto Cipolla. 2018. Multi-task learning using uncertainty to weigh losses for scene geometry and semantics.

[66] Alexander Khanov. 2023. Flavour Tagging with Graph Neural Network with the ATLAS Detector.

[67] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C Berg, Wan-Yen Lo, et al. 2023. Segment anything. In *Proceedings of the IEEE/CVF International Conference on Computer Vision*, pages 4015–4026.

[68] Vitaly Kurin, Alessandro De Palma, Ilya Kostrikov, Shimon Whiteson, and M. Pawan Kumar. 2023. In defense of the unitary scalarization for deep multi-task learning.

[69] Marie Christine Lanfermann. 2017. Deep learning in flavour tagging at the atlas experiment. Technical report, ATL-COM-PHYS-2017-1471.

[70] Marie Christine Lanfermann. 2017. Deep learning in flavour tagging at the atlas experiment. Technical report, ATL-COM-PHYS-2017-1471.

[71] Zhaoqi Leng, Mingxing Tan, Chenxi Liu, Ekin Dogus Cubuk, Xiaojie Shi, Shuyang Cheng, and Dragomir Anguelov. 2022. Polyloss: A polynomial expansion perspective of classification loss functions. *arXiv preprint arXiv:2204.12511*.

[72] Xiaoya Li, Xiaofei Sun, Yuxian Meng, Junjun Liang, Fei Wu, and Jiwei Li. 2019. Dice loss for data-imbalanced nlp tasks. *arXiv preprint arXiv:1911.02855*.

[73] Zhuyun Li, Osamu Yoshie, Hao Wu, Xiaoming Mai, Yingyi Yang, and Xian Qu. 2024. Image analysis method of substation equipment status based on cross-modal learning. *IEEJ Transactions on Electrical and Electronic Engineering*, 19(9):1507–1521.

[74] Baijiong Lin, Feiyang Ye, and Yu Zhang. 2021. A closer look at loss weighting in multi-task learning. *ArXiv*, abs/2111.10603.

[75] Baijiong Lin, Feiyang Ye, Yu Zhang, and Ivor W Tsang. 2021. Reasonable effectiveness of random weighting: A litmus test for multi-task learning. *arXiv preprint arXiv:2111.10603*.

[76] Baijiong Lin and Yu Zhang. 2023. Libmtl: A python library for deep multi-task learning. *The Journal of Machine Learning Research*, 24(1):9999–10005.

[77] T Lin. 2017. Focal loss for dense object detection. *arXiv preprint arXiv:1708.02002*.

[78] Xi Lin, Xiaoyuan Zhang, Zhiyuan Yang, Fei Liu, Zhenkun Wang, and Qingfu Zhang. 2024. Smooth tchebycheff scalarization for multi-objective optimization. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pages 30479–30509. PMLR.

[79] Bo Liu, Yihao Feng, Peter Stone, and Qiang Liu. 2024. Famo: Fast adaptive multitask optimization. *Advances in Neural Information Processing Systems*, 36.

[80] Bo Liu, Xingchao Liu, Xiaojie Jin, Peter Stone, and Qiang Liu. 2021. Conflict-averse gradient descent for multi-task learning. *Advances in Neural Information Processing Systems*, 34:18878–18890.

[81] Shikun Liu, Edward Johns, and Andrew J. Davison. 2019. End-to-end multi-task learning with attention. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (CVPR)*.

[82] Leif Lönnblad, Carsten Peterson, and Thorsteinn Rögnvaldsson. 1990. Finding gluon jets with a neural trigger. *Physical review letters*, 65(11):1321.

[83] Ilya Loshchilov, Frank Hutter, et al. 2017. Fixing weight decay regularization in adam. *arXiv preprint arXiv:1711.05101*, 5.

[84] Ishan Misra, Abhinav Shrivastava, Abhinav Gupta, and Martial Hebert. 2016. Cross-stitch networks for multi-task learning. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 3994–4003.

[85] Paul Musset and Jean-Pierre Vialle. 1978. Neutrino physics with gargamelle. *Physics Reports*, 39(1):1–130.

[86] Aviv Navon, Idan Achituve, Haggai Maron, Gal Chechik, and Ethan Fetaya. 2020. Auxiliary learning by implicit differentiation. *arXiv preprint arXiv:2007.02693*.

[87] Aviv Navon, Aviv Shamsian, Idan Achituve, Haggai Maron, Kenji Kawaguchi, Gal Chechik, and Ethan Fetaya. 2022. Multi-task learning as a bargaining game. *arXiv preprint arXiv:2202.01017*.

[88] NVIDIA Corporation. 2021. *NVIDIA Multi-Instance GPU User Guide*. Accessed: September 17, 2024.

[89] Michela Paganini, Atlas Collaboration, et al. 2018. Machine learning algorithms for b-jet tagging at the atlas experiment. In *Journal of Physics: Conference Series*, volume 1085, page 042031. IOP Publishing.

[90] Adam Paszke, Sam Gross, Francisco Massa, Adam Lerer, James Bradbury, Gregory Chanan, Trevor Killeen, Zeming Lin, Natalia Gimelshein, Luca Antiga, et al. 2019. Pytorch: An imperative style, high-performance deep learning library. *Advances in neural information processing systems*, 32.

[91] Jannicke Pearkes, Wojciech Fedorko, Alison Lister, and Colin Gay. 2017. Jet constituents for deep neural network based top quark tagging. *arXiv preprint arXiv:1704.02124*.

[92] Tao Peng, Caishan Wang, You Zhang, and Jing Wang. 2022. H-segnet: hybrid segmentation network for lung segmentation in chest radiographs using mask region-based convolutional neural network and adaptive closed polyline searching method. *Physics in Medicine & Biology*, 67(7):075006.

[93] Nikita Ivvan Pond. 2023. Top quark pair events for heavy flavour tagging and vertexing at the LHC.

[94] Huilin Qu, Congqiao Li, and Sitian Qian. 2022. Particle transformer for jet tagging. In *International Conference on Machine Learning*, pages 18281–18292. PMLR.

[95] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. 2015. U-net: Convolutional networks for biomedical image segmentation. In *Medical image computing and computer-assisted intervention–MICCAI 2015: 18th international conference, Munich, Germany, October 5-9, 2015, proceedings, part III 18*, pages 234–241. Springer.

[96] Sankar Kumar Roy, Gurupada Maity, Gerhard Wilhelm Weber, and Sirma Zeynep Alparslan Gök. 2017. Conic scalarization approach to solve multi-choice multi-objective transportation problem with interval goal. *Annals of Operations Research*, 253:599–620.

[97] Michael Ruchte and Josif Grabocka. 2021. Multi-task problems are not multi-objective.

[98] Sebastian Ruder. 2017. An overview of multi-task learning in deep neural networks.

[99] Victor Sanh, Thomas Wolf, and Sebastian Ruder. 2018. A hierarchical multi-task approach for learning embeddings from semantic tasks.

[100] S. Schäffler, R. Schultz, and K. Weinzierl. 2002. Stochastic method for the solution of unconstrained vector optimization problems. *Journal of Optimization Theory and Applications*, 114(1):209–222.

[101] Ozan Sener and Vladlen Koltun. 2018. Multi-task learning as multi-objective optimization. *Advances in neural information processing systems*, 31.

[102] Dmitry Senushkin, Nikolay Patakin, Arseny Kuznetsov, and Anton Konushin. 2023. Independent component alignment for multi-task learning. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 20083–20093.

[103] Nathan Silberman, Derek Hoiem, Pushmeet Kohli, and Rob Fergus. 2012. Indoor segmentation and support inference from rgbd images. In *Computer Vision–ECCV 2012: 12th European Conference on Computer Vision, Florence, Italy, October 7-13, 2012, Proceedings, Part V 12*, pages 746–760. Springer.

[104] Samuel Van Stroud, Nikita Pond, Max Hart, Jackson Barr, Sébastien Rettie, Gabriel Facini, and Tim Scanlon. 2023. Vertex reconstruction with maskformers.

[105] Carole H Sudre, Wenqi Li, Tom Vercauteren, Sebastien Ourselin, and M Jorge Cardoso. 2017. Generalised dice overlap as a deep learning loss function for highly unbalanced segmentations. In *Deep Learning in Medical Image Analysis and Multimodal Learning for Clinical Decision Support: Third International Workshop, DLMIA 2017, and 7th International Workshop, ML-CDS 2017, Held in Conjunction with MICCAI 2017, Québec City, QC, Canada, September 14, Proceedings 3*, pages 240–248. Springer.

[106] Thorvald Sørensen. 1948. A method of establishing groups of equal amplitude in plant sociology based on similarity of species content and its application to analyses of the vegetation on danish commons. *Biologiske Skrifter*, 5:1–34.

[107] Thomas Taylor and Daniel Treille. 2017. The large electron positron collider (lep): Probing the standard model. *Adv. Ser. Direct. High Energy Phys*, 27:217.

[108] Yee Whye Teh, Victor Bapst, Wojciech Marian Czarnecki, John Quan, James Kirkpatrick, Raia Hadsell, Nicolas Heess, and Razvan Pascanu. 2017. Distral: Robust multitask reinforcement learning.

[109] Simon Vandenhende, Stamatios Georgoulis, Wouter Van Gansbeke, Marc Proesmans, Dengxin Dai, and Luc Van Gool. 2021. Multi-task learning for dense prediction tasks: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(7):3614–3633.

[110] A Vaswani. 2017. Attention is all you need. *Advances in Neural Information Processing Systems*.

[111] Zirui Wang, Yulia Tsvetkov, Orhan Firat, and Yuan Cao. 2020. Gradient vaccine: Investigating and improving multi-task optimization in massively multilingual models. *arXiv preprint arXiv:2010.05874*.

[112] Christian Widmer, Jose Leiva, Yasemin Altun, and Gunnar Rätsch. 2010. Leveraging sequence classification by taxonomy-based multitask learning. In *Research in Computational Molecular Biology: 14th Annual International Conference, RECOMB 2010, Lisbon, Portugal, April 25-28, 2010. Proceedings 14*, pages 522–534. Springer.

[113] Derrick Xin, Behrooz Ghorbani, Justin Gilmer, Ankush Garg, and Orhan Firat. 2022. Do current multi-task optimization methods in deep learning even help? *Advances in neural information processing systems*, 35:13597–13609.

[114] Tianhe Yu, Saurabh Kumar, Abhishek Gupta, Sergey Levine, Karol Hausman, and Chelsea Finn. 2020. Gradient surgery for multi-task learning. *Advances in Neural Information Processing Systems*, 33:5824–5836.

[115] Manzil Zaheer, Satwik Kottur, Siamak Ravanbakhsh, Barnabas Poczos, Russ R Salakhutdinov, and Alexander J Smola. 2017. Deep sets. *Advances in neural information processing systems*, 30.

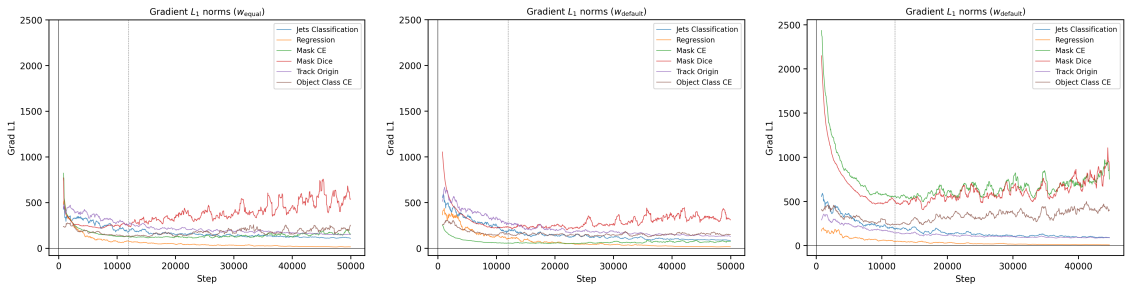
[116] Kai Zhang, Joe W Gray, and Bahram Parvin. 2010. Sparse multitask regression for identifying common mechanism of response to therapeutic targets. *Bioinformatics*, 26(12):i97–i105.

[117] Yue Zhang, Shijie Liu, Chunlai Li, and Jianyu Wang. 2021. Rethinking the dice loss for deep learning lesion segmentation in medical images. *Journal of Shanghai Jiaotong University (Science)*, 26:93–102.

[118] Rongjian Zhao, Buyue Qian, Xianli Zhang, Yang Li, Rong Wei, Yang Liu, and Yinggang Pan. 2020. Rethinking dice loss for medical image segmentation. In *2020 IEEE International Conference on Data Mining (ICDM)*, pages 851–860. IEEE.

Appendix A

Task Interactions



(a) Equal weights. (b) Default weights (pre-scaling). (c) Default weights (post-scaling).

Figure A.1: L_1 Gradient norms $\|G_k\|_1$ of MaskFormer tasks. The gradient magnitudes for an equal weighting scheme are shown on the left. The gradient magnitudes of the raw unscaled losses outputted during training with default weights are shown in the center. On the right, the gradient magnitudes obtained after scaling the losses are shown.

Appendix B

Jet Classification

		u-jet rejection (% increase)				c-jet rejection (% increase)			
	Method	60%	70%	77%	85%	60%	70%	77%	85%
Loss-based	GLS	35.71	25.22	44.78	34.95	53.38	30.13	14.72	7.09
	STCH	-2.56	4.35	5.82	17.04	20.37	11.43	5.16	3.22
	UW	46.15	1.41	12.37	19.02	36.64	24.38	17.27	8.07
	DWA	40.74	39.81	53.20	50.11	87.40	47.50	23.71	10.56
	RLW	52.00	15.20	24.59	26.52	44.96	25.48	13.92	6.84
	FAMO	-11.63	-25.39	-8.03	-4.43	12.95	2.56	0.10	0.31
Grad-based	AlignedMTL	46.22	42.64	62.23	59.31	100.05	49.73	25.08	10.23
	CAGrad	**81.04**	**60.08**	67.38	**65.53**	114.89	**54.97**	26.92	11.97
	GradNorm	52.07	58.32	**70.63**	57.72	108.76	53.43	28.96	12.10
	NashMTL	65.30	61.88	69.53	55.79	**116.64**	53.99	29.44	**12.30**
	PCGrad	**81.04**	69.49	62.73	55.88	114.41	54.13	**29.66**	12.22
	GradVac	46.15	33.33	42.82	50.27	77.12	43.85	24.01	10.70
PolyLoss	Default ($\epsilon = 1$)	35.71	25.22	35.48	33.40	52.58	30.27	14.33	5.55
	Default ($\epsilon = -1$)	58.33	37.14	58.26	51.47	73.02	35.05	17.09	7.12
	DWA ($\epsilon = -1$)	72.73	53.19	66.25	56.81	97.91	51.32	26.31	10.74
	GLS ($\epsilon = -1$)	58.33	30.91	44.38	36.47	54.43	27.45	12.44	4.98

Table B.1: Percentage Increase in Rejection Rates for u-jets and c-jets at Various Signal Efficiencies

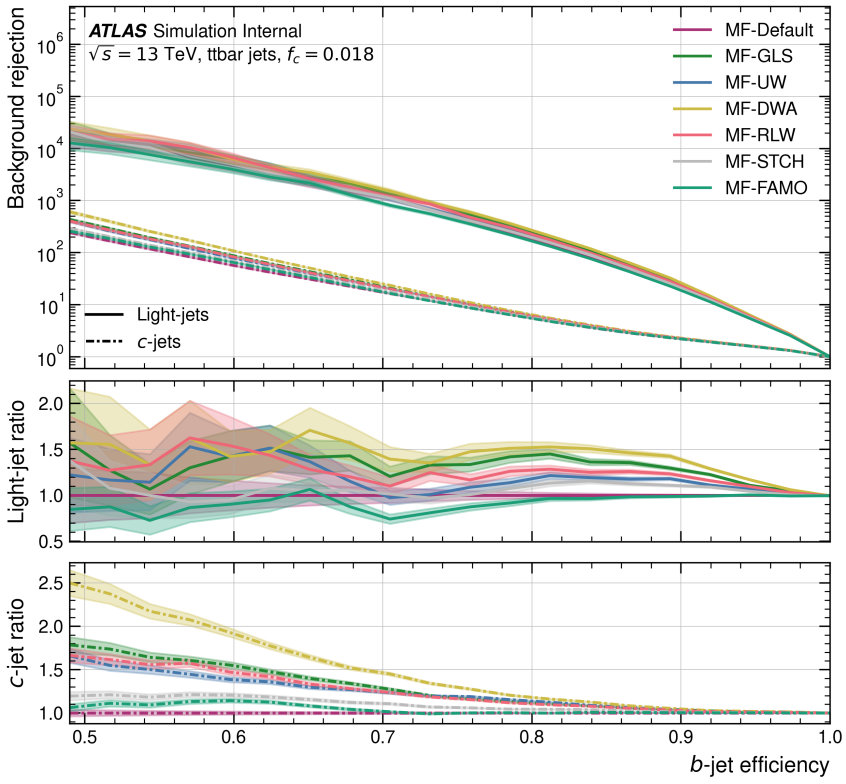


Figure B.1: Jet classification ROC performance curve as a function of b-jet efficiency for various loss-based task weighting methods, benchmarked against the default weighting MaskFormer model.

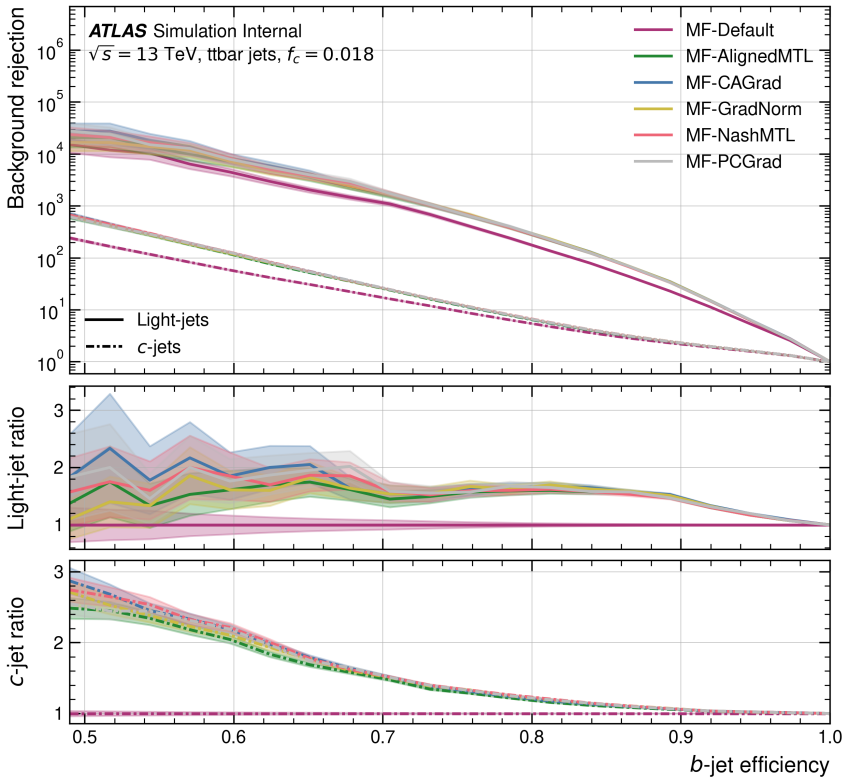


Figure B.2: Jet classification performance against b-jet efficiency for various gradient-based task weighting methods, benchmarked against the default weighting MaskFormer model.

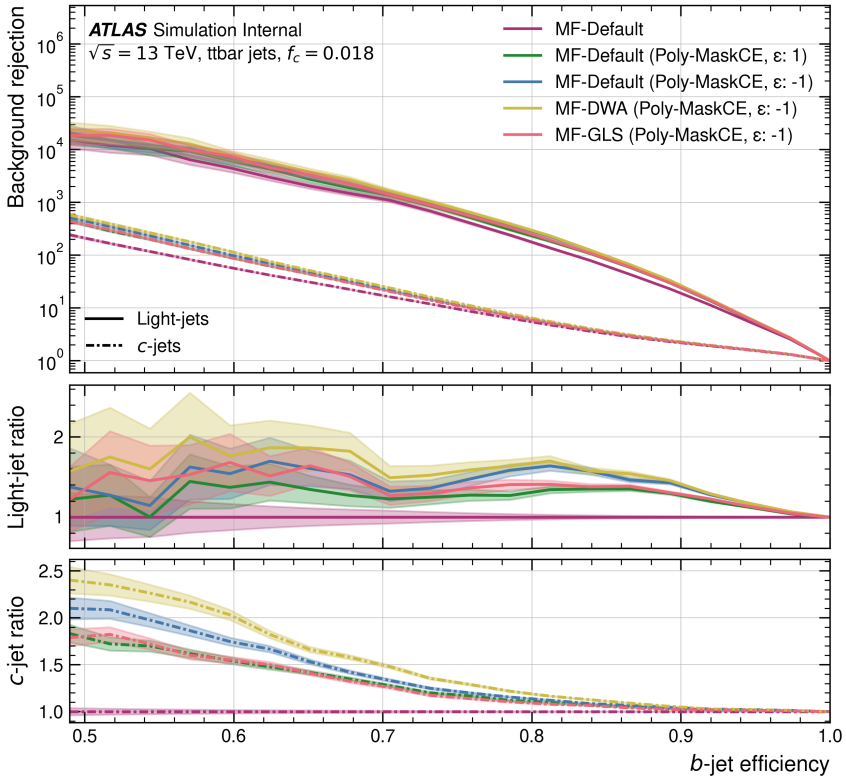
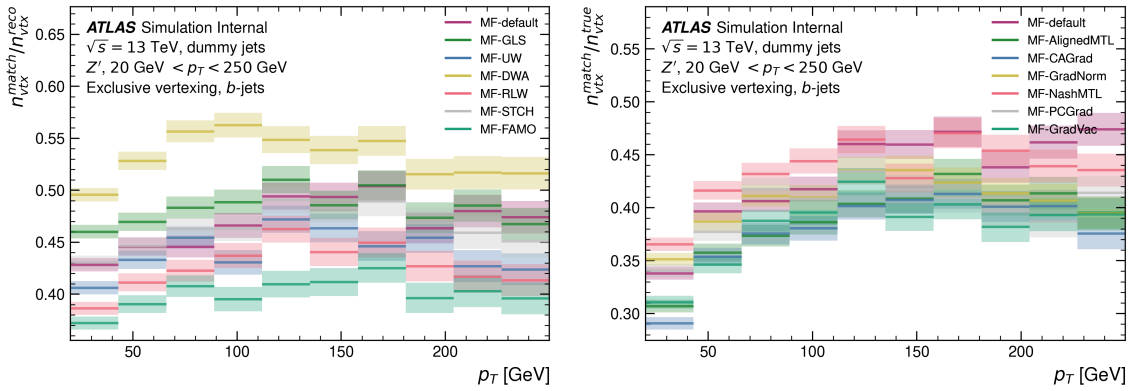


Figure B.3: Enter Caption

Appendix C

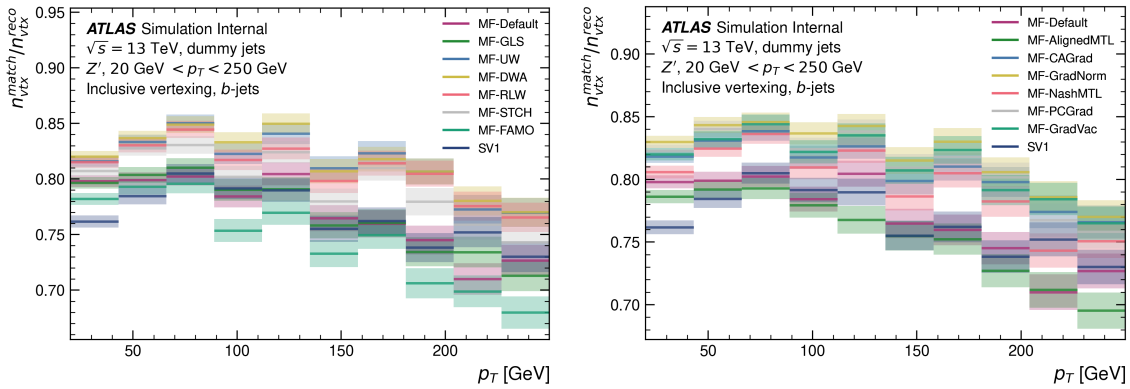
Vertex Finding



(a) Loss-based weighting methods.

(b) Gradient-based weighting methods.

Figure C.1: Exclusive vertex finding efficiency as a function of p_{T} for b-jets evaluated on a Z' sample.



(a) Loss-based weighting methods.

(b) Gradient-based weighting methods.

Figure C.2: Inclusive vertex finding (reconstructing a single vertex within the jet) purity as a function of p_{T} for b-jets evaluated on a Z' sample.

Appendix D

Track Origin Classification

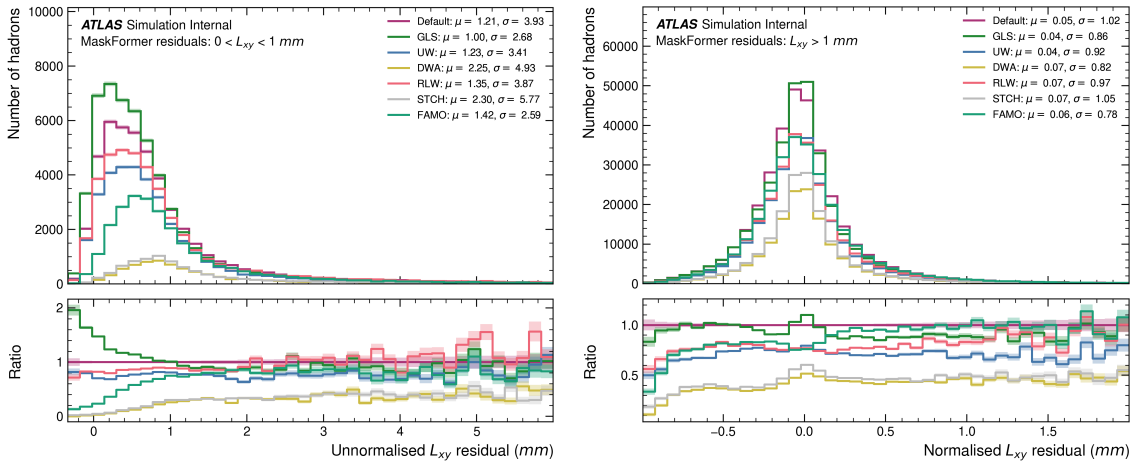
	Method	Pileup	Fake	Primary	FromB	FromBC	FromC	FromTau	OS
	Default	0.97	0.96	0.95	0.90	0.93	0.86	0.90	0.95
Loss-based	GLS	0.97	0.96	0.95	0.91	0.93	0.87	0.91	0.95
	STCH	0.97	0.96	0.95	0.90	0.93	0.87	0.91	0.95
	UW	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	DWA	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	RLW	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	FAMO	0.95	0.91	0.94	0.89	0.92	0.85	0.77	0.92
Grad-based	AlignedMTL	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	CAGrad	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	GradNorm	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	NashMTL	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	PCGrad	0.97	0.97	0.95	0.91	0.93	0.87	0.90	0.95
PolyLoss	Default ($\epsilon = 1$)	0.97	0.96	0.95	0.90	0.93	0.86	0.90	0.95
	Default ($\epsilon = -1$)	0.97	0.96	0.95	0.91	0.93	0.87	0.91	0.95
	DWA ($\epsilon = -1$)	0.97	0.97	0.95	0.91	0.93	0.87	0.91	0.95
	GLS ($\epsilon = -1$)	0.97	0.96	0.95	0.90	0.93	0.86	0.91	0.95

Table D.1: AUC values for all task weighting methods in track origin classification. A one-vs-all binary classification approach is used, where each class is treated as signal and the rest as background. OS refers to OtherSecondary origin.

Appendix E

Vertex Regression

E.1 Loss-based methods



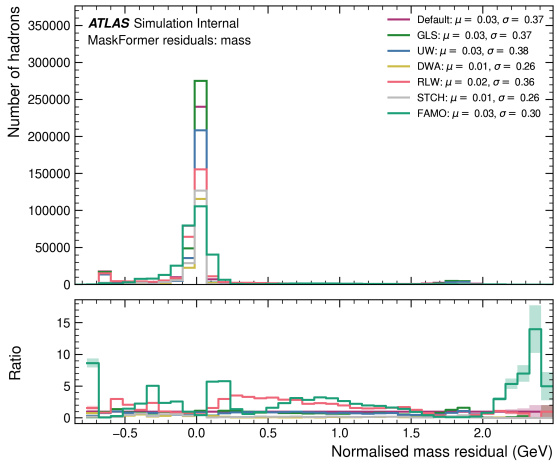
(a) Unnormalised L_{xy} residuals for the five leading p_{T} hadrons in the range [0.0 mm, 1.0 mm].

(b) Normalised L_{xy} residuals for the five leading p_{T} hadrons for $L_{xy} > 1.0$ mm.

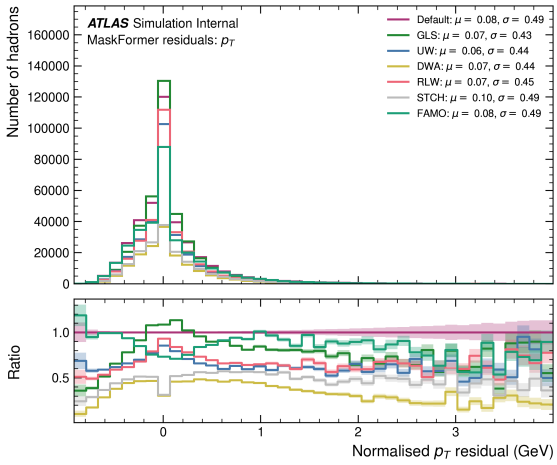
Figure E.1: L_{xy} residuals $(L_{xy_{\mathrm{truth}}} - L_{xy_{\mathrm{predicted}}})/L_{xy_{\mathrm{truth}}}$ for the five leading p_{T} hadrons. The predictions are regressed from MaskFormer models trained with loss-based weighting methods, with the residual's mean and standard deviation displayed in the legend for each method.

E.2 Gradient-based methods

E.3 PolyLoss

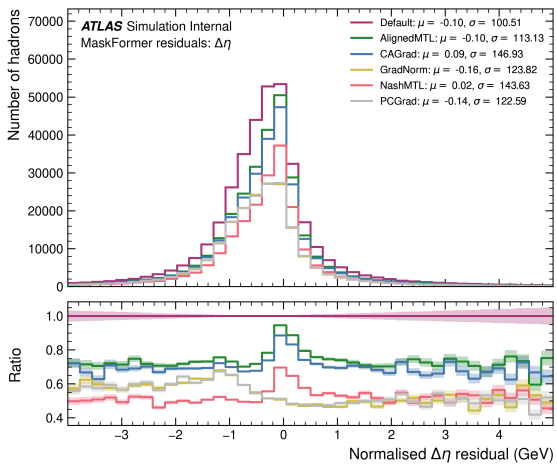


(a) Mass residuals $(m_{\text{truth}} - m_{\text{predicted}})/m_{\text{truth}}$.

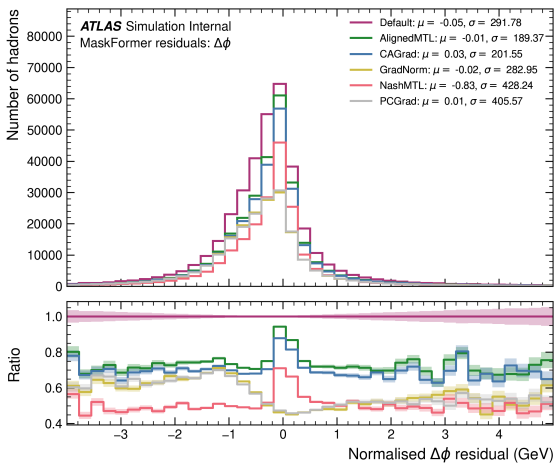


(b) p_{T} residuals $(p_{\text{T}_{\text{truth}}} - p_{\text{T}_{\text{predicted}}})/p_{\text{T}_{\text{truth}}}$.

Figure E.2: Normalised mass (left) and p_{T} (right) residuals for the five leading p_{T} hadrons. The predictions are regressed from MaskFormer models trained with loss-based weighting methods, with the residual's mean and standard deviation for each method displayed in the legend.

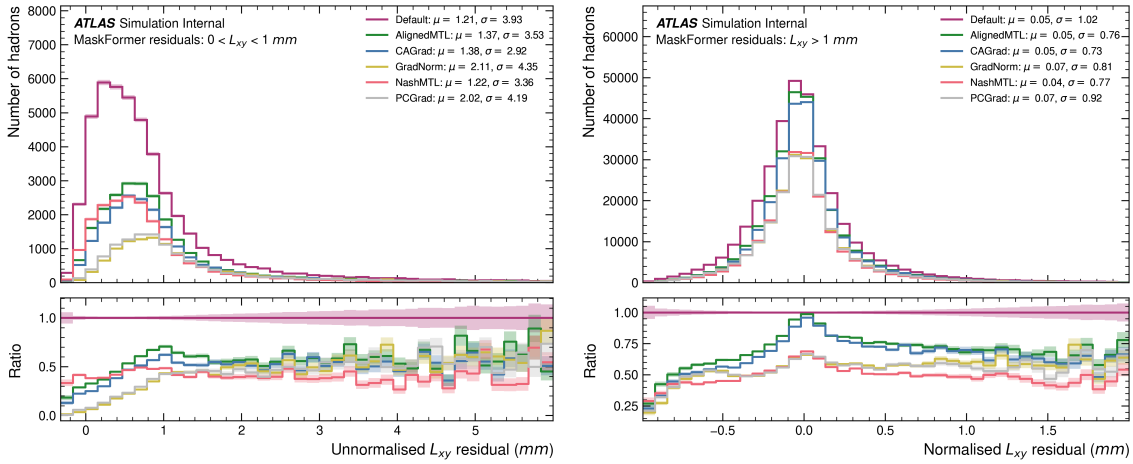


(a) Pseudorapidity difference $d\eta$ residuals $(d\eta_{\text{truth}} - d\eta_{\text{predicted}})/d\eta_{\text{truth}}$.



(b) Azimuthal angle difference residuals $(d\phi_{\text{truth}} - d\phi_{\text{predicted}})/d\phi_{\text{truth}}$.

Figure E.3: Normalised $d\eta$ (left) and $d\phi$ (right) residuals for the five leading p_{T} hadrons. The predictions are regressed from MaskFormer models trained with gradient-based weighting methods, with the residual's mean and standard deviation displayed in the legend for each method.



(a) Unnormalised L_{xy} residuals for the five leading p_{T} hadrons in the range [0.0 mm, 1.0 mm].

(b) Normalised L_{xy} residuals for the five leading p_{T} hadrons for $L_{xy} > 1.0$ mm.

Figure E.4: L_{xy} residuals $(L_{xy_{\mathrm{truth}}} - L_{xy_{\mathrm{predicted}}})/L_{xy_{\mathrm{truth}}}$ for the five leading p_{T} hadrons. The predictions are regressed from MaskFormer models trained with gradient-based weighting methods, with the residual's mean and standard deviation displayed in the legend for each method.