A probabilistic approach for adversarial perturbation of integer-constraint features for Jet-Flavour Tagging Algorithms in the CMS Experiment

B A C H E L O R T H E S I S

Martin Waletzko

August 2025

prepared at the

Physics Institute III A

submitted to the

Faculty of Mathematics, Computer Science and Natural Science

of the RWTH AACHEN UNIVERSITY

written under the supervision of

Prof. Dr. rer. nat. Alexander Schmidt
Physics Institute III A

Second Examiner:

Prof. Dr. rer. nat. Johannes Erdmann
Physics Institute III A

ABSTRACT

Adversarial attacks pose a significant challenge to the robustness of deep-learning models used in high-energy physics analyses. This thesis investigates the vulnerability of jet-flavour tagging algorithms within the Compact Muon Solenoid experiment to perturbations of specifically discrete input features. In addition to standard gradient-based methods such as Projected Gradient Descent (PGD), PIP is introduced – a probabilistic integer attack designed for discrete-valued physics inputs. The attack leverages instantaneous gradient information to assign feature-specific flip probabilities, enabling efficient perturbations without continuous relaxations. A combined PIP-PGD attack is further studied to target both feature domains simultaneously. Evaluation on DeepJet demonstrates that PIP can induce competitive degradation relative to PGD, with complementary effects when applied in conjunction. Adversarial training experiments show improved robustness to the targeted attack types, though with limited generalisation across unseen perturbations. These results highlight the importance of accounting for discrete-feature vulnerabilities in the development of resilient high energy physics machine learning models.

Disclaimer
This thesis acknowledges the use of various artificial intelligence technologies, including but not limited to language models and coding assistants. These tools were exclusively employed for the enhancement of the clarity and comprehensibility of the text. The final content, including all conclusions and interpretations, is the sole responsibility of the author.

ACKNOWLEDGEMENTS

I would like to take this opportunity to express my sincere gratitude to all those who supported and assisted me throughout the preparation of this thesis.

My deepest thanks go to Prof. Dr. Alexander Schmidt for providing me with the opportunity to prepare this thesis under his supervision, and for his trust while I was new to deep learning. His guidance and encouragement made it possible to pursue this work on adversarial robustness and jet-flavour tagging to completion.

I am also grateful to Prof. Dr. Johannes Erdmann for kindly agreeing to serve as the second examiner of this thesis and for his constructive feedback.

Moreover, I would like to thank Ulrich Willemsen and Alexander Jung for their assistance, insightful discussions, and the supportive working environment. In particular, I am thankful for their patience, valuable ideas, and technical support.

Finally, I would like to thank my family and friends for their encouragement and careful proofreading

TABLE OF CONTENTS

LIST OF ILLUSTRATIONS

LIST OF TABLES

ACRONYMS

SM	Standard Model
HEP	High-Energy Physics
LHC	Large Hadron Collider
CMS	Compact Muon Solenoid
ECAL	Electromagnetic Calorimeter
HCAL	Hadron Calorimeter
HLT	High-Level Trigger
L1	Level-1 Trigger
ML	Machine Learning
LSTM	Long Short-Term Memory
ReLU	Rectified Linear Unit
FGSM	Fast Gradient Sign Method
PGD	Projected Gradient Descent
PIP	Probabilistic Integer Perturbation
PIP-PGD	Probabilistic Integer-Perturbed Projected Gradient Descent
ROC	Receiver Operating Characteristic
AUC	Area Under the Curve
JSD	Jensen–Shannon Distance
KL	Kullback–Leibler (divergence)
PF	Particle-Flow
CPF	Charged Particle-Flow
NPF	Neutral Particle-Flow
SV	Secondary Vertex
PV	Primary Vertex
PUPPI	PileUp Per Particle Identification
CSV	Combined Secondary Vertex
BvsL	Bottom-versus-Light discriminator
CvsB	Charm-versus-Bottom discriminator
CvsL	Charm-versus-Light discriminator
TP	True Positive
TN	True Negative
FP	False Positive
FN	False Negative
TPR	True Positive Rate
FPR	False Positive Rate

INTRODUCTION

Modern particle physics is becoming increasingly dependent on Machine Learning (ML) for essential analytical tasks, ranging from event reconstruction to particle identification [1]. One notable application of this technology is jet flavour tagging, where algorithms aim to distinguish jets that originate from different quark flavours. The identification of bottom-quark jets plays a central role in many precision measurements and searches for new physics at the Large Hadron Collider (LHC) [2]. However, the robustness of these models is critical. In the broader ML community, adversarial attacks — small, deliberately designed perturbations that can mislead a model — have been widely studied, particularly in computer vision [3]. Their potential impact in particle physics remains under-explored though, especially in scenarios involving discrete-valued input features.

Jet-tagging algorithms such as DeepJet [4] process hundreds of features, including continuous detector observables and discrete quantities (e.g. counts and category identifiers). Most adversarial studies affect only continuous inputs, overlooking the unique behaviour and vulnerabilities of models when discrete inputs are perturbed. This gap offers an opportunity to probe robustness in a more comprehensive way.

This thesis addresses this challenge by introducing Probabilistic Integer Perturbation (PIP), an adversarial attack tailored to discrete features. PIP uses gradient information in a one-step heuristic to assign feature-specific probabilities for discrete changes, ensuring that all perturbations remain physically meaningful. This method is furthermore combined with the Projected Gradient Descent (PGD) attack [5], targeting both continuous and discrete domains simultaneously and providing a broader evaluation of model vulnerabilities.

Empirically, this study uses the DeepJet architecture [4] trained on approximately ten million simulated jets across all major flavour categories. In addition to the susceptibility for perturbation of continuous values, the results show that DeepJet is vulnerable to degradation through discrete perturbations too. The combined Probabilistic Integer-Perturbed Projected Gradient Descent (PIP-PGD) attack amplifies this effect. Adversarial training with PIP-PGD attacks is presented as a solution, offering robustness in the respective regimes.

Although discrete perturbations do not directly correspond to realistic detector effects, they serve as a diagnostic tool to reveal model dependencies on discretised inputs. Insights that remain hidden in continuous-only studies are revealed through the use of these perturbations. The work presented here contributes to the development of a more complete understanding of adversarial robustness in High Energy Physics (HEP) machine learning and outlines approaches for developing models that are both accurate and resilient in demanding scientific applications.

The thesis is structured as follows: Chapters 2–3 provide the background and baseline, with Chapter 2 reviewing the Standard Model, the LHC, and the CMS detector, and Chapter 3 surveying ML in HEP and adversarial ML while introducing the DeepJet tagger used throughout. Chapter 4 formalises the dataset, evaluation metrics, and threat model, and presents the first methodological contribution: Probabilistic Integer Perturbation (PIP), a gradient-guided attack for discrete inputs that preserves integer constraints and physical bounds, together with the joint PIP–PGD attack that perturbs discrete and continuous features in conjunction. Chapter 5 constitutes the main experimental contribution: a systematic robustness study on DeepJet quantifying attack severity (via Jensen–Shannon distance), classifier performance (ROC/AUC), iteration depth, and PIP sharpness—and a defence via adversarial training, demonstrating that training with PIP–PGD achieves the most balanced cross-robustness while maintaining strong nominal performance; additionally, we examine transferability and cross-robustness across attack types. Chapter 6 summarises the findings, discusses limitations, and outlines directions for robust, discrete-aware jet tagging in future CMS analyses.

Together, these results argue for discrete-aware robustness assessments in HEP ML and provide practical methods to develop taggers that are not only accurate but also resilient in demanding scientific applications.

THE STANDARD MODEL AND CMS

2.1 The Standard Model of Particle Physics

The Standard Model (SM) provides a comprehensive framework that connects the known fundamental particles to three of the four fundamental forces: the strong force, the weak force, and the electromagnetic force. It organises the elementary particles—specifically, the matter particles (quarks and leptons) and the force-carrying particles—under a unifying principle of symmetry. This symmetry ensures that the laws of physics remain consistent even as particles interact through these forces [6].

In the SM, the strong force, which holds quarks together to form protons and neutrons, is carried by particles called gluons. The weak force, responsible for processes like radioactive decay, is mediated by particles known as the W and Z bosons. Meanwhile, the electromagnetic force, which controls how charged particles interact, is carried by the photon [6]. These forces are tied together through the SM's symmetry, which helps explain why particles behave the way they do in nature.

The matter particles in the SM are divided into quarks and leptons. Quarks are the building blocks of protons and neutrons, while leptons include familiar particles like electrons and their neutral partners, neutrinos. These particles are arranged into three generations, each containing two quarks and two leptons (see figure 2.1). The first generation includes the lightest and most stable particles, such as the up and down quarks (which make up protons and neutrons) and the electron and its neutrino. The second and third generations contain heavier, less stable particles that quickly decay into first-generation particles. This repeating family structure explains why protons and neutrons are made of first-generation quarks, while heavier particles are short-lived [7].

A key aspect of the SM is how it accounts for the masses of certain particles. While the symmetry of the SM keeps the photon massless, the W and Z bosons, which are involved in the weak force, need to have mass to match experimental observations. This happens through a process where the symmetry is hidden in the everyday world but still governs the underlying physics, known as spontaneous symmetry breaking [6]. The Higgs field, a field that fills all of space and was proposed in the 1960s by physicists like Englert, Brout, and Higgs, makes this possible. The Higgs field gives mass to the W and Z bosons and, in the process, predicts the existence of a new particle: the Higgs boson [6].

For decades, the Higgs boson remained the missing piece of the SM. In 2012, experiments at CERN's Large Hadron Collider (LHC), conducted by the ATLAS and CMS teams, discovered a new particle with a mass of about 125 GeV [8]. This particle's properties have been studied extensively and align with the Higgs boson predicted by the SM, confirming the mechanism that gives mass to other particles [8].

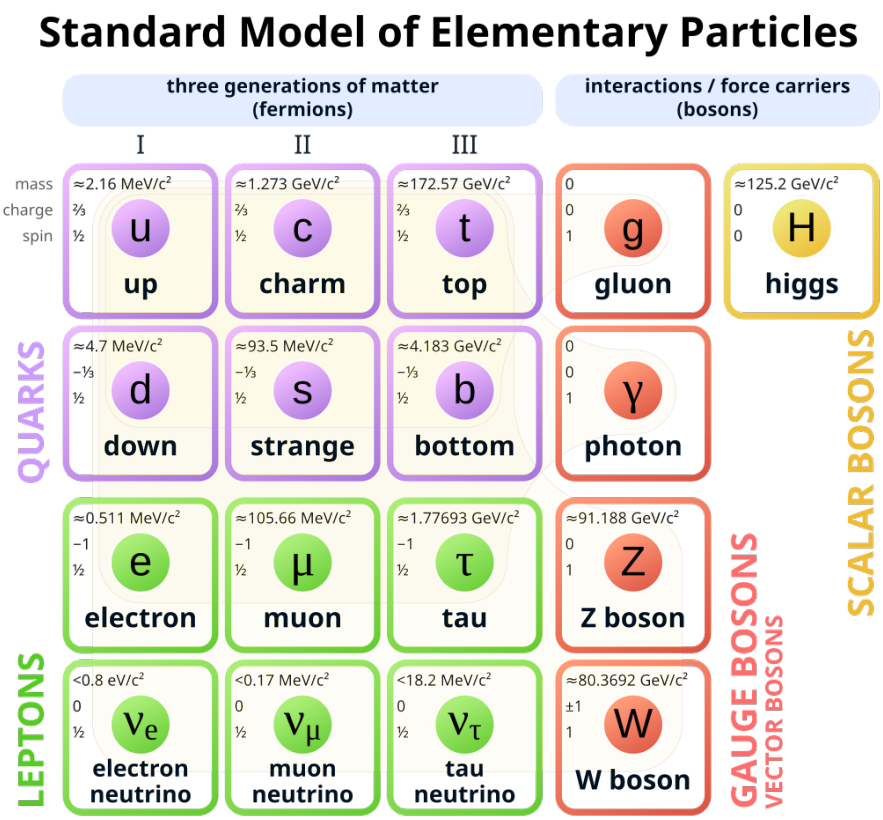


Figure 2.1: Standard model of elementary particles: the 12 fundamental fermions and 5 fundamental bosons [7].

Despite its success in explaining many experimental results, the SM is not a complete theory. It leaves several big questions unanswered. The SM does not explain why there are exactly three generations of particles, each with similar properties but different masses. It also originally assumed neutrinos had no mass, but by now it is known that they have tiny masses, meaning the model needs to be adjusted. The SM does not solve the mystery of why the Higgs boson's mass is so small compared to what quantum effects suggest either — a problem called the hierarchy problem. These gaps suggest the SM is just part of a bigger picture that has not been fully uncovered yet. To do so, particle accelerators are used to challenge the assumptions made in the SM and even go farther — or in this case smaller and into even higher energy scales — and try to look for physics beyond the SM.

2.2 The Large Hadron Collider

To explore phenomena at distance scales far below 10^{-18} m (i.e. at extremely high energy scales), physicists rely on high-energy particle collisions. The LHC at CERN is the world's largest and most powerful particle accelerator, designed to probe such distance scales by colliding protons at unprecedented energies. It is a 27 km circumference circular accelerator that accelerates two

counter-rotating beams of protons to nearly the speed of light and brings them into head-on collision at four interaction points. The LHC, whose design centre-of-mass energy is 14 TeV (7 TeV per beam) [9], was built to achieve a peak instantaneous luminosity of 1×10^{34} cm^{-2}s^{-1} [9]. In its successful Run-1 (2010–2013) and Run-2 (2015–2018) operations, the LHC reached collision energies up to 13 TeV and even exceeded the design luminosity – achieving about 2×10^{34} cm^{-2}s^{-1} in 2018 [10]. The machine is now in Run-3 (2022–present) with a slight energy increase (13.6 TeV). It is being upgraded for the High-Luminosity LHC (HL-LHC) era later this decade, which aims to further boost the luminosity by about an order of magnitude [11].

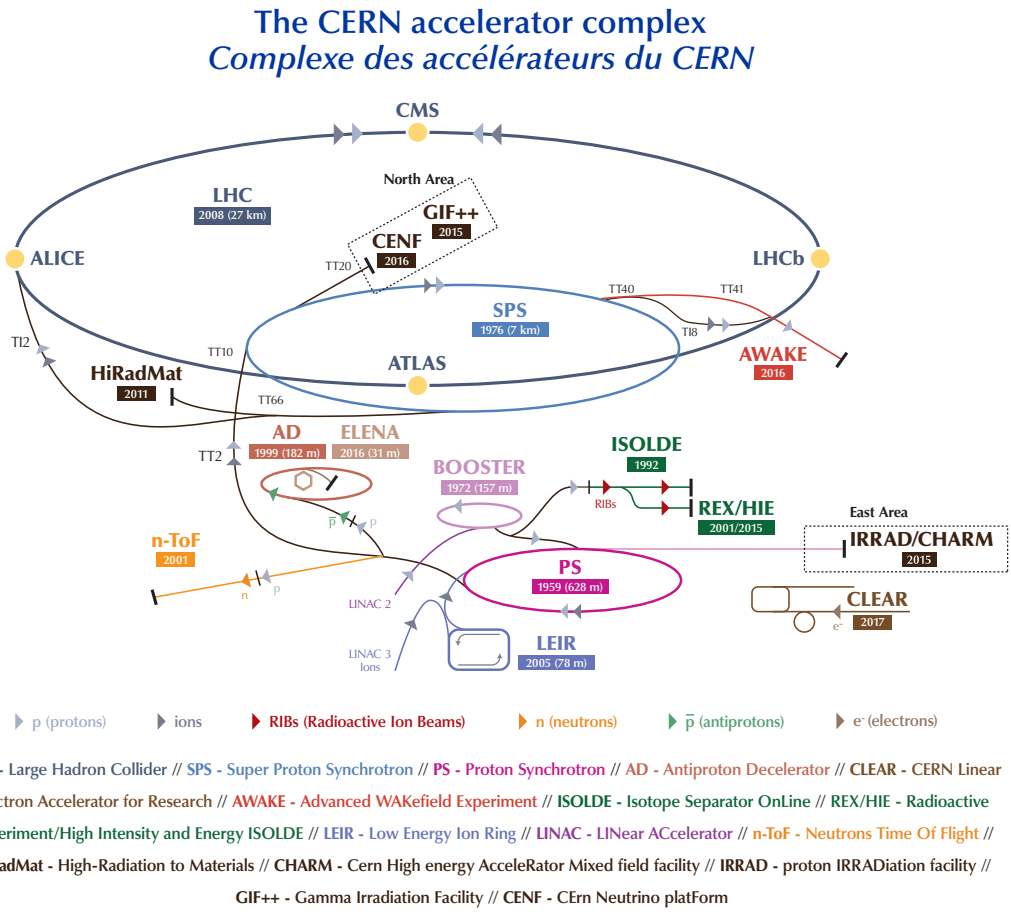


Figure 2.2: Sketch of the CERN accelerator complex [12].

Figure 2.2 shows the structure of the CERN complex with the LHC at its heart. On the discovery front, the LHC's first triumphant success was the Higgs boson. Ongoing searches continue for signs of physics beyond the SM – for example, heavy supersymmetric particles, extra dimensions, or new force carriers – across many possible decay channels. So far, no clear evidence of new particles has appeared; extensive searches in the most promising channels have found no statistically significant deviations from SM expectations [13].

2.3 The CMS Experiment

Collision events in the LHC produce quarks and gluons that cannot exist as free particles (colour confinement [6]). Instead, the outgoing parton radiates additional quarks and gluons, forming a parton shower whose virtuality cascades down to $\mathcal{O}(\text{GeV})$, where hadronisation binds them into colour-neutral hadrons. These hadrons emerge in a narrow cone around the original parton direction, forming a visible spray—a jet.

The CMS detector is one of the two general-purpose detectors at the LHC designed to record these jets with high efficiency and precision, enabling a broad physics program from Higgs boson studies to searches for new phenomena. It is built in a layered, cylindrical geometry around the collision point (see figure 2.3) and, despite the name "Compact", enormous in absolute terms: it is about 15 m in diameter, 21 m in length, and weighs about 14,000 tonnes. At the heart of CMS is a superconducting solenoid coil that generates a magnetic field of 3.8 T within a 6 m inner diameter. This bends the trajectories of charged particles, allowing their momenta to be measured; the steel structure that contains the magnetic flux (the return yoke) also serves as the absorber for muon detection and accounts for the bulk of CMS's mass. Inside the magnet coil, CMS is packed with high-precision tracking and calorimetry systems, and outside the coil are large muon detector chambers, as shown in figure 2.3 [14].

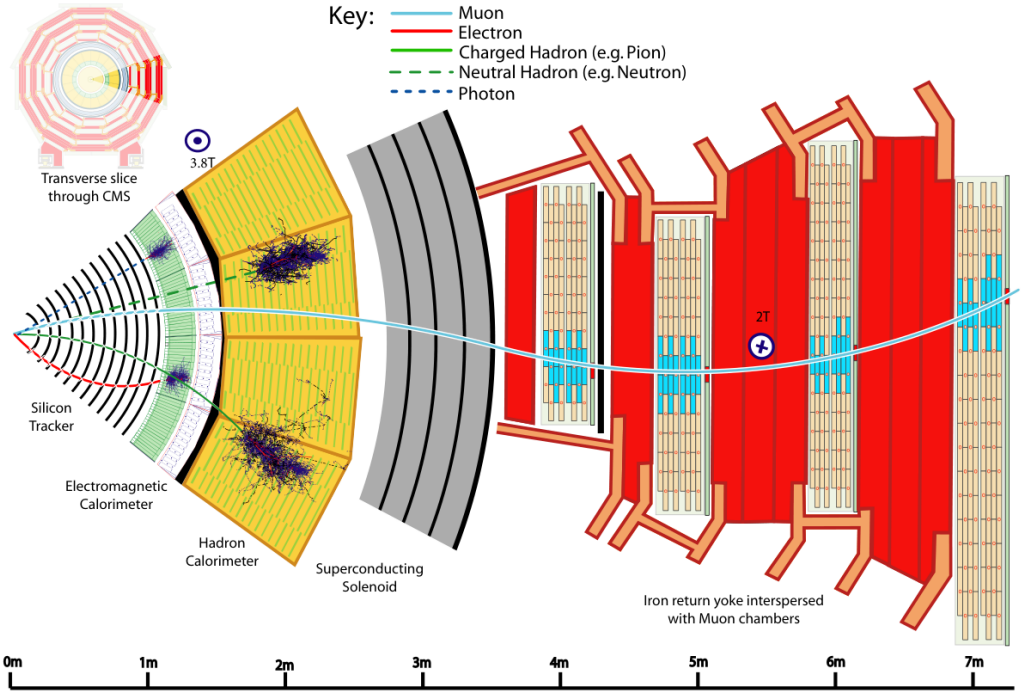


Figure 5: Transverse slice of CMS Experiment [25]. Typical detector interactions are indicated for muons, electrons, hadrons and photons. The dimensions of the different components can be assessed with the length scale on the bottom.

Figure 2.3: Slice of the CMS Detector adapted from [15].

Moving outward from the beam line, the first subsystem is the silicon tracker, a high-granularity detector made of about 75 million individual silicon strips and pixels arranged in concentric layers [14]. When a charged particle from a collision passes through the tracker, it leaves hits in these silicon sensors. By reconstructing sequences of hits, CMS can trace out the tracks of charged particles with fine spatial resolution. The strong magnetic field bends these tracks; the particle's momentum can be determined from the curvature. The tracker is designed to be extremely precise, allowing the reconstruction of secondary vertices from b-hadron decays a few centimetres from the collision point.

Surrounding the tracker is the electromagnetic calorimeter (ECAL), which is made of dense lead tungstate ($PbWO_4$) crystals. The ECAL's task is to fully absorb and measure the energy of electrons and photons [14]. When an electron or high-energy photon enters the ECAL, it initiates an electromagnetic shower in the crystal. The light output from the crystals is proportional to the particle's energy. The CMS ECAL provides excellent energy resolution (on the order of 1% for high-energy electrons/photons) and was pivotal in the Higgs boson discovery via the $H \to \gamma\gamma$ decay mode [8].

Outside the ECAL lies the hadron calorimeter (HCAL). It is sampling calorimeter using alternating layers of absorber (brass or steel) and plastic scintillator. The HCAL is designed to stop and measure the energy of hadrons [14]. Hadrons penetrate the ECAL but are largely absorbed in the thicker material of the HCAL, producing cascades of secondary particles. By collecting the scintillation light from these showers, the HCAL provides a measurement of the hadronic energy. Although the HCAL's resolution is coarser than that of the ECAL, combining its information with the ECAL and tracker allows reconstruction of the energy and direction of jets as well as the estimation of missing transverse energy.

The outermost layers of CMS are the dedicated muon detectors, which give the experiment its name. Muons are charged leptons similar to electrons but about 200 times heavier, and they are penetrating: unlike most particles, muons can traverse substantial amounts of matter. In CMS, after passing through the calorimeters, muons still have enough energy to reach the muon chambers interleaved in the steel return yoke [14]. CMS employs several types of muon detectors (drift tubes, cathode strip chambers, and resistive plate chambers) to track muons independently of the inner tracker. By matching muon tracks in the muon system with those in the inner tracker, CMS achieves a very accurate muon momentum measurement.

Trigger System

As the LHC collision rate is enormous, CMS cannot record data from every collision. Instead, the experiment uses a trigger system to filter events in real time. CMS employs a two-level trigger system: a Level-1 (L1) trigger implemented in custom electronics (fast hardware logic) and a High-Level Trigger (HLT) implemented in software running on a computing farm [16]. The L1 trigger, which is using information from the calorimeters and muon system, reduces the 40 MHz collision rate to around 100 kHz by selecting events with interesting signatures (such

as high-energy objects). Then the HLT takes those L1-selected events and runs a streamlined version of the full event reconstruction to apply more refined selection criteria, outputting a final rate of around 1 kHz to permanent storage [16]. This multi-tiered trigger is crucial for ensuring that the most interesting collisions – those potentially containing rare new physics or useful signals – are recorded for offline analysis, while discarding the rest.

Coordinates

CMS uses a right-handed coordinate system with the origin at the centre of the detector. The x-axis points radially inward toward the centre of the LHC ring, the y-axis points vertically upward, and the z-axis is aligned with the beam direction [17]. Instead of simple Cartesian coordinates (x, y, z), it is often convenient to use cylindrical and spherical coordinates (r, ϕ, θ) (or equivalently (r, ϕ, η)) to describe angles and distances. Here, r denotes the distance from the beam line in the transverse plane (x–y plane), ϕ is the azimuthal angle in that plane (with $\phi = 0$ defined along the positive x-axis), and θ is the polar angle measured from the positive z-axis.

In practice, the polar angle θ is expressed via the pseudorapidity η, defined as:

$$\eta = -\ln\left(\tan\frac{\theta}{2}\right) \tag{2.1}$$

For highly relativistic particles (with $E \gg m$), the pseudorapidity η is approximately equal to the particle's rapidity y, which is given by

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

where E is the particle's energy and p_z is the z-component of the momentum; the transverse momentum is $p_T = \sqrt{p_x^2 + p_y^2}$. This coordinate choice conveniently captures the detector's cylindrical symmetry and the boost-invariant nature of motion along the beam axis [17].

MACHINE LEARNING IN HIGH ENERGY PHYSICS

Machine-learning (ML) methods play an increasingly central role in HEP analyses, from low-level detector reconstruction to high-level statistical inference [18]. Their usefulness derives from the ability to learn complex, non-linear correlations in high-dimensional feature spaces that are difficult to express in analytic form. Historically, particle physicists pioneered the use of multivariate algorithms in the 1990s and 2000s for analysis tasks, relying on techniques like artificial neural networks and boosted decision trees (BDTs) [19]. The emergence of deep learning around 2012 enabled training of very large neural networks that outperformed previous state of the art models [19]. This caused a rapid expansion of HEP applications spanning across particle/event identification, reconstruction and even real-time data filtering [20].

3.1 Machine Learning using Deep Neural Networks

Neural networks (NNs) are computational models designed to approximate a mapping from input variables $\mathbf{x}$ to a desired output $\mathbf{y}$. They consist of interconnected nodes organized into layers, inspired by the structure of the human brain [21]. The input layer contains one node for each dimension of the training data. The output layer corresponds to the desired output dimensions, reflecting the number of distinct mappings the network needs to learn. Between input and output layer, a varying number of hidden layers number enhance the network's ability to adapt and model complex patterns. Each node is associated with a weight, a bias, and an activation function, enabling non-linear mappings for intricate problems. During training, a process called backpropagation adjusts these weights by computing gradients across the network from input to output, guided by a specified loss function [22].

When the number of hidden layers increases significantly, the network is typically referred to as a deep neural network (DNN), capable of modelling highly complex relationships in data. DNNs can approximate arbitrarily complicated functions, making them well-suited for tasks such as particle identification, energy regression, pile-up mitigation, and anomaly detection [23]. In practice, DNN-based algorithms now permeate the workflow of collider and astroparticle experiments, achieving better performance than many traditional methods. For example, convolutional neural networks (CNNs) that treat detector data as images have outperformed physics-motivated features in classifying jet substructure [24]. Likewise, recurrent neural networks (RNNs [22]) or long short-term memory (LSTM [22]) networks can naturally handle sequential data, as demonstrated by their use in jet-flavour taggers to process tracks and vertex sequences [4].

A typical supervised-learning workflow in HEP involves several common steps. First, labelled datasets are prepared using Monte Carlo (MC) simulations, which provide ground-truth information for particle types, kinematics, event categories, etc. Supervised training on MC truth is widespread because real collisions cannot be labelled event-by-event with the desired

signal/background distinctions [18]. However, ML models can overfit to simulation-specific artifacts, potentially limiting their ability to generalize to real data [18]. To mitigate this, physicists carefully design training procedures and validation tests. The dataset is typically split into training and testing portions; models are trained with a suitable loss function (for example, cross-entropy for classification or mean squared error for regression) which reflects the physics goals. Sometimes custom objectives are used – for instance, optimizing directly for a statistical significance metric or incorporating systematic uncertainty penalties – but this is balanced against practical considerations [25].

3.2 Adversarial Machine Learning

Adversarial machine learning studies how deliberately crafted inputs — or unintentional mismodelling — can cause an ML model to make incorrect or biased predictions [26]. In computer vision, the addition of nearly imperceptible perturbations to an image can fool a deep classifier into misidentifying objects [3]. Resistance to small perturbations is also important in HEP context: Recent studies have raised awareness that supervised HEP models may latch onto simulator-specific quirks — for example, subtle differences in detector response modelling — such that they fail to generalize to real data [18]. These quirks make HEP models susceptible to data-simulation discrepancies: a slight miscalibration in the simulation or a tiny adjustment in the input features might cause disproportionate shifts in the model output. Ensuring robustness against these effects is therefore important, especially as ML algorithms are integrated into critical physics results.

Adversarial Attacks

Most textbook adversarial attacks assume continuous, differentiable input spaces (such as pixel intensities in an image) and rely on gradient information to find an effective perturbation. For instance, the Fast Gradient Sign Method (FGSM) and the Projected Gradient Descent (PGD) (see Chapter 4 for detailed descriptions) are two well-known techniques that use the gradient of the model's loss with respect to the input features to adversarially tweak the input [3]. These methods nudge the input in the direction that most increases the model's error, subject to a constraint on the perturbation size (often measured by an L_p norm). FGSM accomplishes this in one step using the sign of the gradients, while PGD applies multiple small steps iteratively and projects the result to ensure the perturbation is staying within a norm bound [5]. Other attack methods include DeepFool, which finds the smallest perturbation to cross the decision-boundary, and various black-box attacks that do not require gradient access. These techniques have been successfully applied in computer vision, where inputs are continuous pixel arrays and small changes can be carefully crafted to cause misclassifications.

However, applying adversarial attacks in HEP is challenging due to the constrained realm of physical features. Despite those complications, researchers have begun to explore HEP-specific adversarial attacks. Initial studies have shown that HEP classifiers are susceptible to small input

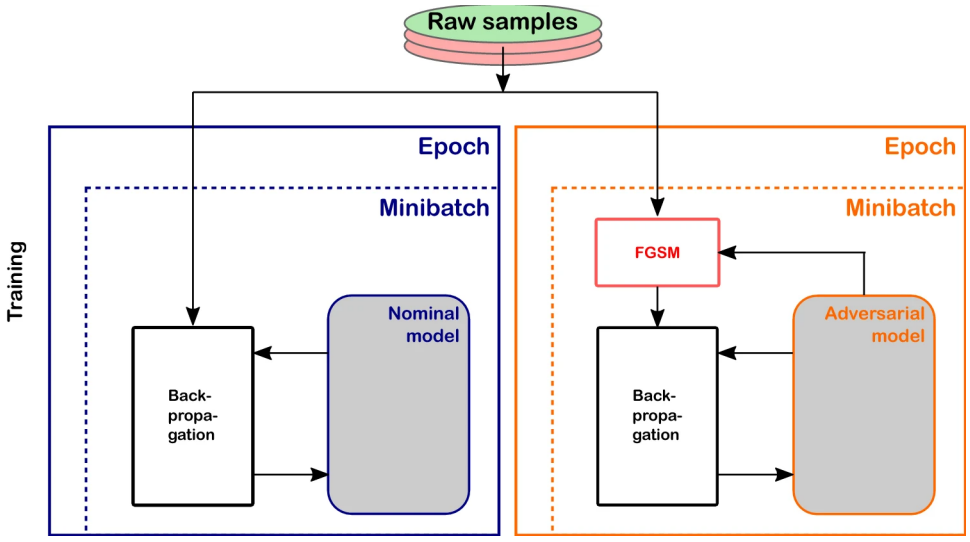


Figure 3.1: Comparison between nominal and adversarial training procedure for a neural network. The network on the right is trained on adversarial samples that are designed to fool the network, generated by the FGSM attack. By training on the adversarial examples, the network is less susceptible to adversarial attacks [1].

distortions on continuous input features; one group systematically applied FGSM perturbations to a jet-tagging network's input and found that the classifier's performance degraded significantly [1]. This is not surprising, given that many networks used in HEP rely on piecewise linear activation functions and high-dimensional feature spaces. Consequently, perturbation of a single high-level feature — analogous to the "one-pixel-attack" in images — could potentially flip a classification if the network is right at a decision boundary [1].

Adversarial training

Adversarial training is a defence technique wherein the model is trained on deliberately perturbed examples in addition to the nominal data. In practice, this means augmenting each training batch with adversarial modified inputs and using them with the correct labels to update model parameters (see figure 3.1). By learning from these fraudulent examples, the model's decision boundaries become smoother and less sensitive to tiny fluctuations in input features [3].

In HEP, adversarial training reduces reliance on simulator-specific quirks and improves resilience to mismodelling. For example, if a tagger leans on a mismodelled feature, training with amplified versions of that mismatch pushes the network to learn more robust discriminants. This strategy was demonstrated by injecting slight perturbations into jet features during training — effectively simulating systematic shifts — and showed that the resulting tagger maintained high nominal performance while becoming significantly less vulnerable to such shifts [1]. The adversarially-trained model coped better with variations in detector response and particle distribution, suggesting improved generalization to real data.

3.3 The DeepJet Tagger

The DeepJet tagger represents a successful application of deep learning to a classic HEP problem by distinguishing heavy-flavour jets from light-quark or gluon jets. These taggers employ hierarchical neural-network architectures to combine information from many low-level inputs and produce a single probability for each flavour category. The key insight of DeepJet is to use as much information as possible about each jet, instead of relying on a hand-selected subset of inputs as previous algorithms did [4]. Earlier-generation taggers like CSV/DeepCSV used a fixed number of high-quality tracks and secondary vertices as input. In contrast, DeepJet forgoes a broad preselection of jet constituents; it ingests an extensive list of per-particle features – including charged and neutral particle-flow candidates, secondary vertex properties, and global jet attributes – and makes the neural network learn which features are important [4].

Candidate selection

The candidate preselection is built on jets reconstructed using the particle-flow (PF) algorithm of CMS. Jets are clustered from PF candidates with the anti-k_T algorithm that is based on the distance between constituents of the jet [27]. The PF event reconstruction identifies and reconstructs individual particles (photons, electrons, muons, charged/neutral hadrons) by optimally combining information from all subdetectors [27]. This provides a detailed list of charged and neutral particle candidates for each jet, along with any reconstructed secondary vertices from decays of long-lived particles.

All PF candidates associated with a jet – both charged and neutral – are used as inputs, supplemented by up to four secondary vertices and a set of global jet features. To maintain a fixed-size input representation, a maximum of 25 charged PF candidates and 25 neutral PF candidates are included per jet. This "no strict preselection" approach retains as much information as possible, avoiding the loss of potentially useful tracks that earlier methods discarded [4].

Architecture

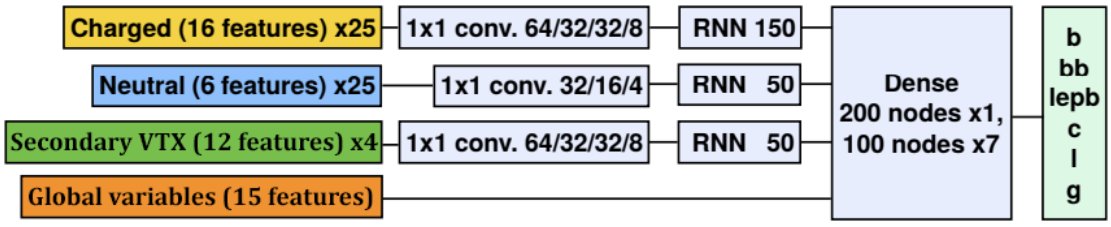


Figure 3.2: Illustration of the DeepJet architecture, adapted from [4] to fit the dataset (see 4.1). Charged and neutral particle-flow candidates, secondary vertices, and global variables are used as inputs to the tagger, which are processed by the hidden layers (white). The model outputs the probabilities for a jet to belong to one of the six jet classes.

The DeepJet Tagger takes 15 global input variables, 16 charged ParticleFlow (CPF) input variables for 25 candidates, 6 neutral ParticleFlow (NPF) input variables for 25 candidates and 12 secondary vertex (SV) input variables for 4 candidates, constituting 613 inputs per jet (see figure 3.2). In the first stage, separate input streams are established for each type of low-level object: one for charged particles (tracks), one for neutral particles, and one for secondary vertices. Each stream passes through a stack of 1×1 convolutional layers with bias parameters. These layers compute per-particle features without mixing information across particles, so each particle receives the same operation regardless of order [4]. Essentially, this track learns a representation for each track, each neutral, and each vertex input. Next, the outputs of these convolutional streams are fed into recurrent layers (LSTM). The charged candidate LSTM uses 150 nodes, the neutral LSTM and secondary vertex LSTM each use 50 nodes. The LSTM outputs then are fed into a dense layer of 200 nodes followed by seven layers of 100 nodes each, using ReLU activation throughout. The final output layer is based on a softmax activation function and consists of six nodes, which are the model outputs for the six jet categories.

METHODOLOGY

In HEP, applying adversarial attacks is challenging due to the nature of the data. Collider events or detector readouts are not simple fixed-size vectors or continuous values — they often involve structured or discrete inputs. This means that naive perturbations which violate those symmetries or feature constraints might be non-physical and thus easily recognized as invalid. In short, the allowed adversarial perturbation in HEP are limited by physical realism in a way typical image perturbations are not.

This chapter outlines the methodological framework of this thesis, designed to systematically investigate adversarial attacks for discrete input features on the DeepJet model. The approach begins with a detailed description of the dataset and feature sets used for training and testing. Following, the evaluation metrics and the adversarial threat model get considered. Subsequently, various attack strategies are presented: Starting with conventional gradient-based methods applied to continuous inputs, progressing to novel techniques that account for the discrete nature of certain features. The chapter concludes by exploring a combined approach that integrates these strategies to enhance the robustness analysis of the model.

4.1 Dataset

As DeepJet's input feature set is extremely rich, comprising hundreds of variables that describe each jet's constituents and related vertices [4], training for the scope of this thesis was carried on a smaller subset of the data containing roughly 10 million jets (see Table 4.1). For each jet, up to 25 charged particle-flow (PF) candidates, 25 neutral PF candidates, and up to 4 secondary vertices are considered, in addition to a set of global jet-level features [4]. To keep the data representation uniform, if a jet has fewer constituents than the maximum, placeholder values (zeros) are used for the missing entries. All inputs are standardized to physically meaningful ranges or default values. Below is a summary for each feature group used in DeepJet. An overview of all features and their respective descriptions is attached in Tables A.2 to A.5.

- **Global features.** Jet-level observables and context summarising the jet and its environment. It includes continuous features such as basic kinematics (`jet_pt`, `jet_eta`) and other variables from the earlier CSV algorithm (e.g. `trackSumJetEtRatio`, `trackSumJetDeltaR`). It has discrete values for counts of reconstructed constituents (e.g. `nsv`, `npv`).
- **Charged PF candidates (25 per jet).** Consists of reconstructed charged particles (mostly charged hadrons and leptons) linked to the primary vertex by tracks. Contains mostly, continuous features with per-track kinematics, geometry relative to the jet/vertices, and impact parameters (e.g. `Cpfcan_ptrel`, `Cpfcan_deltaR`) that encode displacement

from the PV. Integer-valued features are included as well through reconstruction/quality indicators (e.g. `Cpfcan_VTX_ass`, `Cpfcan_quality`).

- **Neutral PF candidates (25 per jet).** Reconstructed neutral particles within the jet (photons and neutral hadrons). Features describe neutral energy shares and positions relative to the jet (e.g. `Npfcan_ptrel`, `Npfcan_deltaR`). Includes integer based features with per-candidate flags (e.g. `Npfcan_HadFrac`, `Npfcan_isGamma`).
- **Secondary vertices (4 per jet).** Grouped as displaced vertices formed from subsets of charged tracks, typically from heavy-flavour hadron decays. Features include track count (`sv_ntracks`) and SV kinematics (e.g. `sv_pt`, `sv_mass`).

Table 4.1: Number of total jets and number of each jet flavour in the sample used for training and testing.

flavour	Training	Testing
b	1397932 (16.5%)	124115 (16.9%)
bb	40100 (0.5%)	3488 (0.5%)
b_{lep}	440395 (5.2%)	39437 (5.4%)
c	896245 (10.6%)	77612 (10.6%)
uds	2960180 (35.0%)	254785 (34.7%)
g	2726240 (32.2%)	234435 (31.9%)
Total	8461092 (100%) [92.0%]	733872 (100%) [8.0%]
	9194964 [100%]	

4.2 Evaluation Metrics and Threat Model

To evaluate adversarial attacks based on their capability it is important to formalize how the attack impact is quantified and how the assumed capabilities of the adversary (the threat model) are clarified. The success of an adversarial attack is measured by its ability to reduce the model's classification accuracy while ensuring perturbations are minimally perceptible with respect to the original data distribution. Several metrics are employed to capture these aspects and how they are used in the physical context of particle physics. Table A.1 provides a summary of the primary evaluation metrics used and their physical interpretation, while these metrics are described individually in the following sections first.

Labelling strategy

Evaluating performance, particularly for heavy-flavour tagging, it is useful to break down the classifier's outputs into physically meaningful categories. In this thesis, jets are categorized as containing bottom (b) quarks (including those coming from *B* hadron decays, denoted "bb", and from b hadrons that produce leptons, "b_{lep}"), containing charm (c) quarks, or containing light-flavour (L) content (which includes up, down, strange quarks labelled "uds" and gluons "g"). The DeepJet network is trained to produce a set of output probabilities listed in table 4.2.

Table 4.2: Jet categories of DeepJet.

jet category	description
b	jets containing one b hadron decaying hadronically
bb	jets containing two b hadrons
b_{lep}	jets containing one b hadron decaying leptonically
c	jets containing at least one c hadron and no b hadrons
uds	jets containing no b or c hadrons (initiated by u, d or s quark)
g	jets containing no b or c hadrons (initiated by gluon)

For the application in physics analyses, the outputs are often used to compute discriminators, which help to distinguish between two of the broader categories B, C and L. The definitions of the BvsL, CvsB and CvsL discriminators are given below in (4.1), (4.2), (4.3). During this thesis the BvsL discriminator is used as the primary metric for evaluation of the DeepJet classifier's performance against adversarial attacks. BvsL distinguishes bottom quark jets (b, bb, b_{lep}) from light-flavour jets (uds, g) and is critical in many high-energy physics analyses, such as those involving Higgs boson or top quark decays, where bottom jets are key signatures. By targeting BvsL, one can assess the classifier's robustness in a scenario that is both highly relevant to physics applications and sensitive to adversarial manipulations, as misclassifying bottom jets as light-flavour jets could significantly impact analysis outcomes. Focusing on BvsL, furthermore, allows for a streamlined evaluation of adversarial efficacy, providing clear insights into the classifier's vulnerabilities while aligning with the needs of precision physics measurements.

$$\text{BvsL} = \frac{P(b) + P(bb) + P(b_{lep})}{P(b) + P(bb) + P(b_{lep}) + P(uds) + P(g)} \tag{4.1}$$

$$\text{CvsB} = \frac{P(c)}{P(c) + P(b) + P(bb) + P(b_{lep})} \tag{4.2}$$

$$\text{CvsL} = \frac{P(c)}{P(c) + P(uds) + P(g)} \tag{4.3}$$

Performance

To effectively evaluate a classifier's performance, especially in the presence of class imbalances, a confusion matrix is used. This matrix summarizes the model's predictions across different categories, providing a detailed breakdown of correct and incorrect classifications for a binary classification problem. The matrix consists of four key components, defined as follows:

- **True Positive (TP):** The number of instances correctly predicted as belonging to the positive class.

- **True Negative (TN):** The number of instances correctly predicted as belonging to the negative class.
- **False Positive (FP):** The number of instances incorrectly predicted as belonging to the positive class when they actually belong to the negative class.
- **False Negative (FN):** The number of instances incorrectly predicted as belonging to the negative class when they actually belong to the positive class.

To assess the classifier's performance for each class individually, two key metrics are derived from the confusion matrix: the True Positive Rate (TPR) and the False Positive Rate (FPR). These are defined as below:

$$TPR = \frac{TP}{TP + FN}, \tag{4.4}$$

$$FPR = \frac{FP}{FP + TN}. \tag{4.5}$$

The TPR measures the fraction of positive instances correctly identified by the classifier. For instance, a high TPR in b-tagging indicates that most b-jets are correctly tagged. The FPR, on the other hand, quantifies the fraction of negative instances incorrectly classified as positive. A low FPR is desirable, as it indicates fewer non-b-jets are mistakenly tagged as b-jets, reducing false alarms.

These metrics form the basis of the Receiver Operating Characteristic (ROC) curve, a graphical tool used to evaluate the trade-off between the classifier's sensitivity (TPR) and its false positive rate (FPR) [28]. The ROC curve plots TPR on the y-axis against FPR on the x-axis, with the FPR often logarithmically scaled to emphasize small values. Each point on the ROC curve corresponds to a different classification threshold, allowing visualization of how the model balances correctly identifying positive instances (e.g., b-jets) against incorrectly tagging negative instances (e.g., non-b-jets).

A key metric derived from the ROC curve is the Area Under the Curve (AUC) score, computed by integrating the ROC curve [28]. The AUC quantifies the overall performance of the classifier across all possible thresholds. An AUC score of 1.0 indicates a perfect classifier achieving maximum TPR with no false positives, while an AUC of 0.5 corresponds to a random classifier (equivalent to a coin toss) with no discriminative power. In practice, the AUC is widely used as a single, comprehensive metric to compare different models or assess improvements in a given model, particularly in tasks where one class (e.g., b-jets) is significantly rarer than the other.

Input similarity

In assessing adversarial attacks of HEP inputs, it is also useful to quantify how severe a perturbation is in terms of a norm, as well as for the statistical difference between distributions. The Jensen-Shannon Distance (JSD) is applied to measure the similarity between the nominal input distribution and the adversarial perturbed input distribution, excluding the default and zero padded values:

$$JSD = \sqrt{\frac{1}{2}(D_{KL}\,(P||M) + D_{KL}(Q||M))} \tag{4.6}$$

The JSD is derived as the square root of the Jensen-Shannon Divergence, which itself is a symmetrized and bounded version of the Kullback-Leibler (KL) divergence,

$$D_{KL} = \sum_{x \in \chi} P(x) \log \frac{P(x)}{Q(x)}. \tag{4.7}$$

A low JSD (close to zero) indicates that the perturbed inputs are almost indistinguishable from nominal inputs in those one-dimensional projections, even though they may be arranged so as to fool the ML model[1] [29]. This is useful to assess how *stealthy* a given attack is in a statistical sense. For instance, a study found that for certain attacks the average JSD for key kinematic distributions is on the order of $\mathcal{O}(10^{-3})$, indicating very minor differences [29].

Equations (4.6) and (4.7) formalize the definitions of JSD and KL divergence. The bins for the JSD calculations were chosen separately between continuous and discrete input features, where continuous bins are based on the Freedman-Diaconis rule [30] and the binning for discrete values correspond to a linear range between feature minima/maxima with a step size of one. In essence, JSD provides a single summary number to quantify input similarity, complementing other measures used such as differences in correlation matrices, visual inspection of feature histograms, etc. Going forward, the JSD is presented for various features under different attack scenarios, to confirm that the proposed adversarial attack strategies meet the design criterion of stealthiness.

Threat Model

In this thesis, a white-box threat model is assumed. This means the adversary is assumed to have full knowledge of the model architecture and parameters (the trained DeepJet model) and is able to compute gradients with respect to the input. The attacker also has access to the same input features that are used in the model – in this case, the simulated truth labels for jets. The

[1] This can happen through intentional mismodelling where bars get switched while retaining their local scale and is not in the scope of this thesis.

adversarial perturbations are constrained to be small in magnitude – consistent with potential detector effects or plausible data manipulation. This is tied to real scenarios by noting that small changes in input features could arise from, e.g. slight miscalibrations of the detector or uncertainties in reconstruction algorithms. Thus, the attacks discussed can be thought of as deliberately structured mismodelling that an attacker might introduce.

The attacker is not assumed to be able to change the model on the fly. All attacks are either done on the trained model in a post-training evaluation setting or as an adversarial input set during the training procedure. Also only feature-level attacks are considered, not physically shifting detector readouts – the attacker can adjust the input values fed to the algorithm (within realistic bounds) but it does not impose that unaltered features remain consistent. For example, an attacker cannot create a jet out of thin air; they can only tweak existing feature values of a jet. Consequently, all attacks are subject to a prior selection of valid input features, based on a masking described in the next chapter.

4.3 Experimental Setup:

All methods in this section use the standard DeepJet neural network architecture for jet-flavour identification (see 3.3) as the victim model. The training/testing procedures were carried out over the course of 30 epochs with a batch size of 1024. The backpropagation was based on the Adam optimizer (a variant of stochastic gradient descent) using an initial learning rate on the order of 0.0001. Masks $\mathcal{M}_{\text{float}}$, $\mathcal{M}_{\text{int}}$, and $\mathcal{M}^*_{\text{int}}$ were used to distinguish between continuous (floating-point valued), discretely mapped, and true discrete input features. $\mathcal{M}_{\text{float}}$ contains kinematic quantities like track momentum and impact parameter, $\mathcal{M}_{\text{int}}$ includes features that nominally should not be targeted by traditional adversarial attacks in DeepJet, such as hit multiplicities or track counts or derived features from binned distributions (e.g. PUPPI weights). As $\mathcal{M}_{\text{int}}$ is primarily used to avoid the manipulation of undesired input features independent of the actual data type, a sub-set $\mathcal{M}^*_{\text{int}} \subset \mathcal{M}_{\text{int}}$ of true integer-based values was used for the scope of this thesis.

It is worth noting, that not all integer features are used in $\mathcal{M}^*_{\text{int}}$, as some features, such as `sv_ntracks`, would require generating new track data. This is not included into the category of slight miscalibrations. An overview for the masking of the input features and how they are typed inside of the DeepJet model is provided in Tables A.2 - A.5.

Furthermore a clamping is applied to features $x_i \in \mathcal{M}^*_{\text{int}}$ to guarantee physical plausibility. For integers clamping is especially important as these features often span only a small portion of the natural numbers, representing boolean flags (such as `IsGamma`, `HadFrac`), or categories (e.g. `VTX_ass`). This clamping is based on bounding features to their respective minimum and maximum, reverting changes that go out of bounds back to their nominal state.

4.4 Gradient-Based Reference Attacks

Having defined the evaluation metrics and having also discussed how the dataset is structured, it is now possible to assess the models robustness for adversarial attacks. Starting with the textbook formulation of FGSM — a one-step adversarial perturbation in the direction of the input gradient's sign [3] – and evaluating its efficacy under two contrasting input-space assumptions: (1) using only continuous features versus (2) a naive treatment of integer-valued observables as continuous. This provides a baseline for understanding how discrete inputs complicate adversarial attacks.

FGSM on Continuous Inputs

First, the *standard way* of working with simple adversarial attacks in `b-hive` to date is discussed — perturbing only the continuous-valued inputs. Integer-valued features are fixed (effectively ignored by the attack) via the mask $\mathcal{M}_{\text{float}}$. The update rule for each feature x_i is given by Equation (4.8):

$$x_i^{\text{adv,float}} = \begin{cases} x_i + \epsilon \cdot \varepsilon_{ind} \cdot \text{sign}\big(\nabla_{x_i} J(\Theta, x, y)\big), & x_i \in \mathcal{M}_{\text{float}} \\ x_i, & \text{otherwise,} \end{cases} \tag{4.8}$$

where $J(\Theta, x, y)$ is the loss function of the model with parameters Θ on input x with true label y. Here, ε_{ind}, is not a fixed constant but rather a tensor attributing for the relative scaling of individual features, which is globally scaled by a magnitude ϵ. In words, small steps of a relative size ϵ are applied in the direction of the gradient's sign for each continuous feature, while leaving discrete features unchanged. This formulation is the same used in previous studies of adversarial attacks on DeepJet models [31], and implements a per-feature perturbation of maximum allowed size ϵ in the direction that most increases the loss.

FGSM with naive Integer Handling

Next, the discrete features in the FGSM attack are included without special treatment – essentially pretending that all inputs are continuous. In this naive approach, the mask is dropped and FGSM is applied "as-is" across the entire feature vector, including integer-valued observables. The update rule simplifies to Equation (4.9) for every feature:

$$x_i^{\text{adv,int}} = x_i + \epsilon \cdot \varepsilon_{ind} \cdot \text{sign}\big(\nabla_{x_i} J(\Theta, x, y)\big). \tag{4.9}$$

This direct application allows the gradient to perturb integer features by a fractional amount. For example, if x_i represents a count in a simplified case where $\varepsilon_{ind} \equiv 1$, and the gradient $\nabla_{x_i} J$ is positive, then Equation (4.9) would increment the count by ϵ, potentially resulting in a non-integer value. Such values are non-physical, since one cannot have fractional tracks; similarly, a negative perturbation could result in a count going below zero (e.g. $0 - 0.2 = -0.2$

tracks). These invalid states illustrate the main drawback of the naive method. Consequently, the naive approach fails dramatically at generalizing for the perturbed input set, which can be seen in figure A.1.

Projected Gradient Descent

While FGSM provides a quick, one–shot probe of model vulnerability, it often underestimates the true worst-case loss increase because it is restricted to a single gradient step. PGD extends FGSM into a multi-step procedure that iteratively walks the input towards regions of higher loss while re-projecting the perturbed sample back into an ℓ_∞–ball of radius $\varepsilon := \epsilon \cdot \varepsilon_{ind}$ around the original input. This makes PGD the de facto "gold standard" white-box adversary for continuous domains [5].

For each iteration $t = 0, \ldots, k-1$ only the floating-point coordinates $x_i \in \mathcal{M}_{\text{float}}$ are updated,

$$\begin{aligned} \tilde{x}_i^{(t+1)} &= x_i^{(t)} + \alpha \ \operatorname{sign}\big(\nabla_{x_i} J(\Theta, x^{(t)}, y)\big) \\ x_i^{(t+1)} &= \operatorname{clip}\big(\tilde{x}_i^{(t+1)},\ x_i^{(0)} - \varepsilon,\ x_i^{(0)} + \varepsilon\big), \end{aligned} \tag{4.10}$$

where $x^{(0)}$ is the clean input, α is the per-step step-size (often chosen as $\alpha = \varepsilon/k$), and `clip` performs the projection back into the ℓ_∞ constraint set.

4.5 Probabilistic Integer Perturbation

Having identified the shortcomings of standard FGSM on discrete features, a novel attack method – Probabilistic Integer Perturbation (PIP) – that respects the discreteness of certain inputs is discussed here. Instead of adding a fraction ϵ to an integer-valued feature, PIP uses the gradient information to decide whether or not to step an integer value to a neighbouring value. Crucially, this decision is made stochastically, via a Bernoulli trial whose probability is derived from the gradient magnitude. In this way, PIP can be thought of as a stochastic FGSM tailored to discrete variables, ensuring that outputs remain in the integer domain.

The scheme is defined in Equation (4.11)

$$\begin{aligned} p_i &= \left\lceil \min\!\left(1,\ \frac{|\nabla_{x_i}\mathcal{L}|}{\max|\nabla_x \mathcal{L}| + c}\right) \right\rceil^s, \quad u_i \sim \text{Bernoulli}(p_i), \\ x_i^{\text{adv}} &= \begin{cases} x_i + u_i \cdot \operatorname{sign}(\nabla_{x_i}\mathcal{L}), & x_i \in \mathcal{M}_{\text{int}}^* \\ x_i, & \text{otherwise,} \end{cases} \end{aligned} \tag{4.11}$$

where $\mathcal{L} = J(\Theta, x, y)$ is the loss and $\max \nabla_{x_i}\mathcal{L}$ denotes the maximum absolute gradient among all features (or specifically among integer features – the formulation can be restricted to the $\mathcal{M}_{\text{int}}^*$ subset). A small positive constant $c = 10^{-8}$ is added to the denominator to avoid division

by zero issues. In practice this has a negligible effect though. The term p_i is essentially a normalized and rescaled version of the gradient magnitude for feature i. After dividing by the maximum gradient of each batch, it is capped to 1, and then raised to a power s (the *sharpness* hyperparameter)—thus $p_i \in [0, 1]$. A Bernoulli random variable $u_i \in \{0, 1\}$ is then drawn with the success probability p_i. If $u_i = 1$, the feature i is then perturbed by adding or subtracting one unit in the direction of the gradient's sign. If $u_i = 0$, the feature is left unchanged. By construction, this produces an integer adversarial example for integer features, since only whole units get added or subtracted.

To build intuition, consider $s = 1$ initially. Then p_i is directly proportional to $\nabla_{x_i}\mathcal{L}$. In this case, each integer feature is flipped with probability proportional to how sensitive the model's loss is to that feature. If a feature's gradient is very small relative to others, its flip probability will be near zero, meaning it is very unlikely to waste perturbation budget on a feature that doesn't affect the outcome. Contrary, if a feature has the largest gradient, it gets $p \approx 1$ and will almost certainly be flipped. Introducing the sharpness parameter $s > 0$ allows to adjust this distribution. If $s > 1$, then probabilities are raised to a higher power, which sharpens the distribution: features with $p_i < 1$ will have their probability pushed lower and any feature that was at the maximum (with raw probability 1) remains at 1. Thus, higher s makes PIP more aggressively focus on the top-gradient features (many low-gradient features will effectively get probability nearly 0). Conversely, if $0 < s < 1$, the distribution flattens: even features with smaller gradients get a relatively higher chance to flip. Tuning s thus controls how many integer features on average will be perturbed in one attack instance. A larger s yields sparser attacks (fewer features changed, targeting only the most "salient" ones), while a smaller s yields more widespread changes. In this study, PIP gets applied across a range of sharpness values to examine the sensitivity.

Another important implementation detail is the randomness in PIP. Because u_i is drawn from a Bernoulli distribution, each attack realization can be different, even for the same input and same model. This stochasticity matches the discrete nature of the problem—integers change in whole steps—but it requires reproducibility for fair evaluation and reproducible training. This is ensured through seeding in b-hive and allows to debug and analyse the attack behaviour reliably.

Comparison to Existing Methods: The PIP method parallels several probabilistic strategies for adversarial attacks on discrete data. In particular, it echoes the Gumbel-Softmax reparameterisation trick [32], which enables approximate gradients for categorical variables via stochastic sampling. Related methods, such as the Gumbel-Softmax (Jang et al., 2017), Gumbel Attack [33], and the Probabilistic Categorical Adversarial Attack (PCAA) (He et al., 2023), transform discrete selection into a continuous probability space to permit gradient-based optimization. Unlike these iterative, optimized approaches, PIP is a single-step heuristic: the instantaneous gradient from one backward pass is used to assign flip probabilities and randomly perturb integer features. This simplicity reduces computation to one gradient evaluation and random sampling, making the combination with continuous-feature attacks easier and well-suited for HEP applications.

Similar to FGSM, PIP allows an iterative application offering a more nuanced attack that is able to correct itself after each iteration (e.g. when the gradient sign is flipped after one iteration and $u_i = 1$). This allows for a composite attack of PIP and PGD over multiple iteration.

4.6 Probabilistic Integer-Perturbed Projected Gradient Descent

Probabilistic Integer-Perturbed Projected Gradient Descent (PIP-PGD) is a composite attack of PGD and PIP. It perturbs floating-point and integer features independently, acting on a broader spectrum of the input domain. This attack incorporates PGD's efficacy on continuous spaces with PIP's discrete probabilistic nature. In this context "PGD(1)" will refer to a single-step application (i.e. FGSM), while higher numbered PGD(k) would mean k iterative steps on the continuous features. Likewise, these steps get also applied to PIP, where the same approach is used, i.e. PIP(k) corresponds to k individual PIP attacks, where after each iteration, the updated gradient gets reevaluated. The joint attack can be written as:

$$x_i^{\text{adv}} = \begin{cases} x_i^{\text{adv,float}}, & x_i \in \mathcal{M}_{\text{float}} \\ x_i^{\text{adv,int}}, & x_i \in \mathcal{M}_{\text{int}}^* \\ x_i, & \text{otherwise} \end{cases} \tag{4.12}$$

where $x_i^{\text{adv,float}}$ is produced by a PGD attack (4.8) on continuous features and $x_i^{\text{adv,int}}$ is produced by a PIP attack (4.11) on the integer features. In words, a float-space attack and an int-space attack is run in parallel, each acting on different components of the input. The feature sets are disjoint ($\mathcal{M}_{\text{float}} \wedge \mathcal{M}_{\text{int}} = \emptyset$), therefore the perturbations do not directly conflict or overlap. Accordingly continuous and discrete parts can be treated independently first and then combined to form a full adversarial input.

There are a few choices to clarify in this combined procedure. One question is whether to compute gradients once or twice for the two subsets. In this implementation, the gradient of the loss with respect to the entire input is taken once per iteration and then used for both the PGD update on continuous features and the probability computation in PIP for discrete features. This ensures that the continuous and discrete perturbations are based on the same "view" of the input's vulnerability. It also ensures that both attacks commute, so that the order of application is uncorrelated.

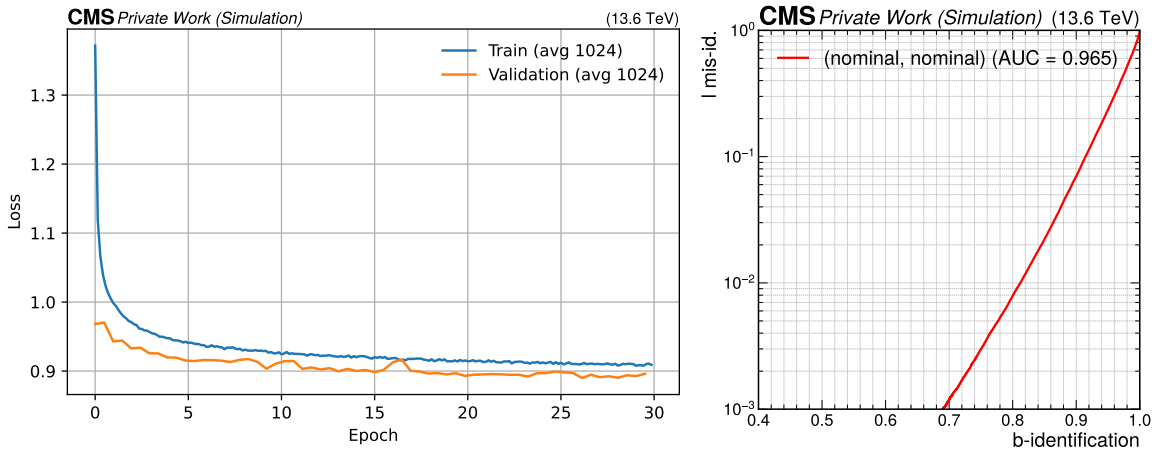
To evaluate the performance of adversarial attacks, the models nominal performance is assessed as a baseline for unperturbed input data. Based on this data, the input similarity (see 4.2) is addressed for multi-iterational attacks of PGD, PIP and PIP-PGD. Furthermore, a PGD attack is applied as a reference for a successful attack. The JSD value is used throughout this entire chapter to quantify the mismatching of the given adversarial input. ROC Curves act as labels for the success of the attacks while being complemented by additional loss/validation curves that offer a broader view of the attacks performance. All attacks are evaluated in terms of attack severity, performance on nominal data, and adversarial-trained performance, followed by an analysis of their transferability and cross-robustness to determine whether PIP — alone or in combination with continuous-domain attacks — offers robustness beyond existing methods.

5.1 Reference study

This section establishes critical benchmarks for evaluating adversarial attacks on DeepJet. It first assesses the model's nominal performance under standard conditions to provide a baseline for unperturbed data. Then, it examines the impact of PGD attacks to set a reference of a successful attack for comparing subsequent adversarial strategies.

Nominal Performance

At nominal training and testing, the training validation yields a stable curve with a convergence at around 0.93. The validation is sitting slightly below at roughly 0.90 offering stable convergence too (see figure 5.1a). Figure 5.1b portrays the identification score for BvsL found to be $AUC = 0.965$.



(a) Training and validation loss for nominal training up to 30 epochs.

(b) ROC Curve for the nominal trained baseline tested against nominal input.

Projected Gradient Descent

To assess the efficacy of the novel approach it is also necessary to look at well established adversarial attacks as an adversarial baseline — in this case PGD. For the sake of simplicity a magnitude of $\epsilon = 0.1$ is applied for all following PGD attacks[1].

Severity: Figure 5.2 provides a comprehensive view of input similarity across the entire input domain for up to three PGD iterations. The figure reveals an important patterns: (1) The JSD values for all features and iterations generally fall within the range of $O(10^{-2})$ to $O(10^{-3})$, in agreement to the threshold typically considered for stealthy attacks. (2) The perturbation magnitude generally stays constant over multiple iteration. However, this is not uniform across all features; some features show minimal change even after multiple iterations, while others exhibit significant perturbation from the first iteration.

This selective perturbation behaviour stems from the gradient reassessment process inherent in PGD. Unlike single-step methods like FGSM that always assume the worst-case perturbation direction, PGD's iterative approach allows for more nuanced optimization. Features with strong gradients in the initial iterations may see their gradients diminish or even reverse direction in subsequent iterations, causing them to "project back" toward their original values. The overall perturbation pattern suggests that PGD attacks are highly targeted. They focus computational effort on features that provide the most leverage for compromising the classifier's performance.

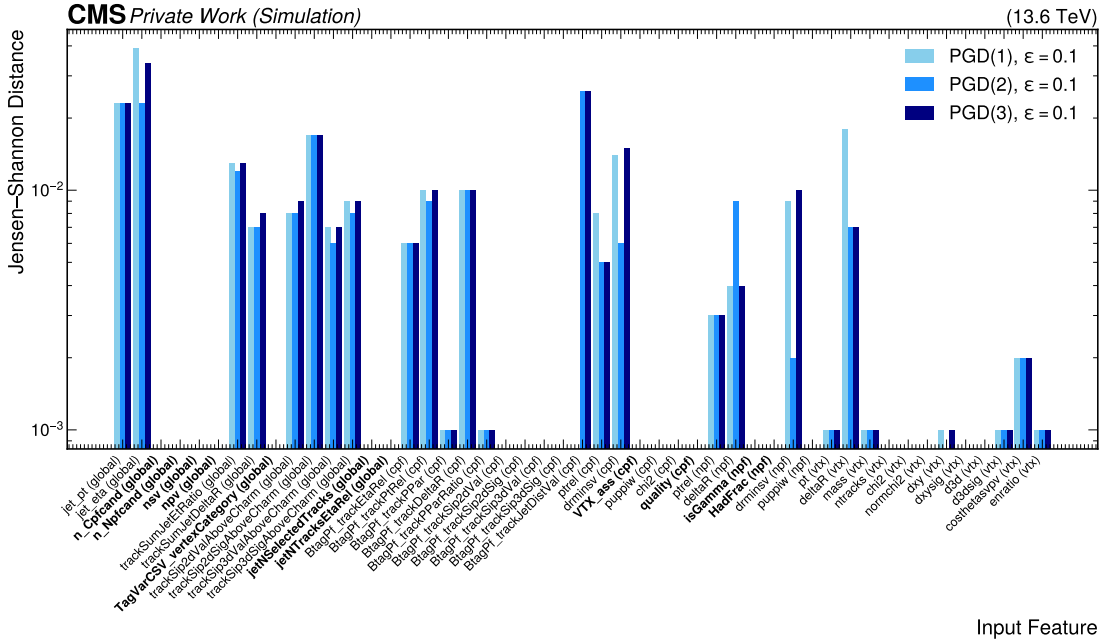


Figure 5.2: JSD input similarity development for up to three iterations of the PGD attack with $\epsilon = 0.1$ tested against a nominal trained model.

[1] As briefly discussed in section 4.4 an additional scaling is applied proportional to the relative feature scale tensor ε_{ind}. The behaviour for varying PGD magnitudes does not fall into the scope of this thesis.

Attack: Figure 5.3 illustrates the impact of multiple PGD iterations on the BvsL discrimination task. The baseline nominal performance (AUC = 0.965) serves as the reference point against which attack efficacy is measured. The single-iteration PGD(1) attack achieves an AUC of approximately 0.937, representing a clear reduction in performance. This initial attack demonstrates that even minimal adversarial perturbation can compromise the classifier's discriminative power. The two-iteration PGD(2) attack yields the same AUC score as PGD(1), while the three-iteration variant reduces the AUC slightly to approximately 0.936.

This progressive degradation highlights two important aspects of PGD attacks. The iterative nature allows for more sophisticated perturbation strategies that can exploit the model's decision boundaries more effectively than single-step methods. Moreover, the diminishing returns observed between iterations 2 and 3 suggest that the attack approaches a performance floor, beyond which additional iterations provide minimal benefit.

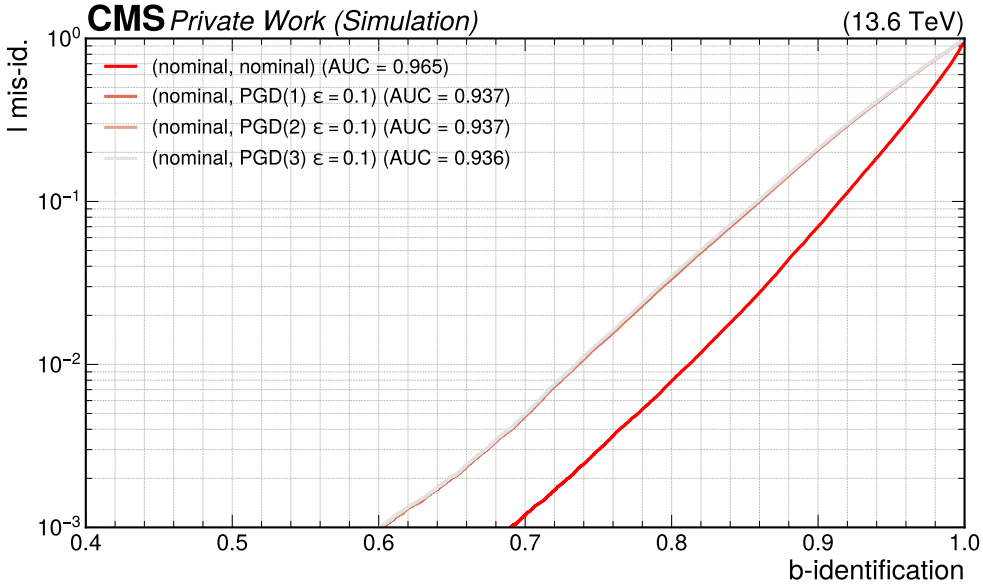


Figure 5.3: ROC curves for BvsL misidentification. Nominal trained model tested against one, two, and three iterations of PGD attacked inputs with $\epsilon = 0.1$.

Adversarial Training: Adversarial training with PGD perturbations introduces unique challenges to the learning process. The training and loss curve (figure 5.4) exhibit a higher spread between the convergence in loss (training sitting at approximately 1.0, validation at roughly 0.95), reflecting the inherent difficulty of learning robust representations under continuous adversarial perturbation. This is caused due to the necessity to simultaneously optimize for performance on clean data while developing defences against gradient-based attacks.

Notably, the training process achieves stable convergence despite the adversarial component. This means that PGD-based adversarial training is a viable strategy for improving model robustness.

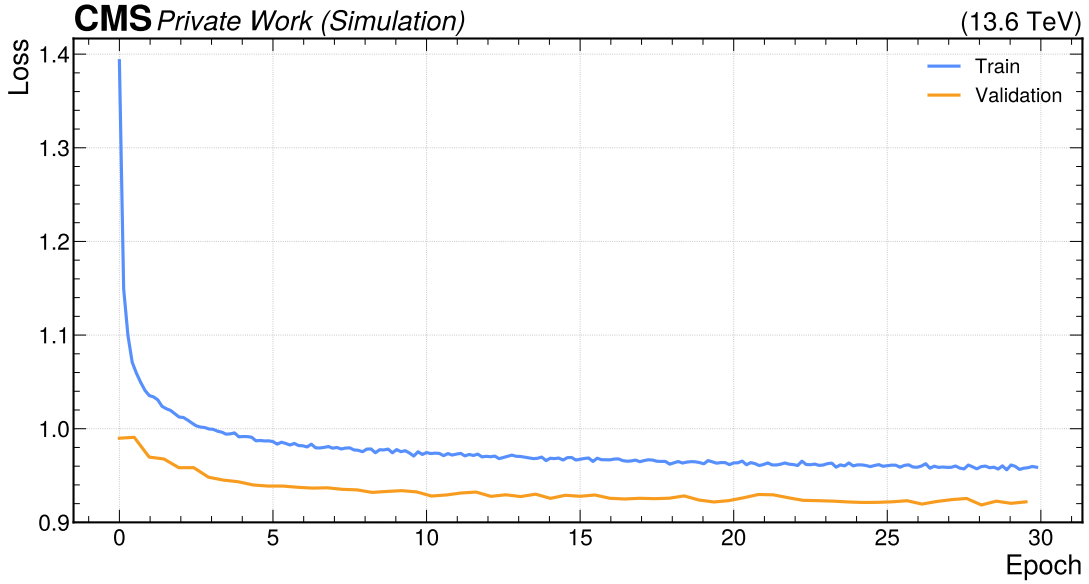


Figure 5.4: Training and validation loss for a PGD(1) trained model with a magnitude of $\epsilon = 0.1$ for up to 30 epochs.

A PGD trained model offers robustness across PGD inferred data ($AUC = 0.955$), while remaining a high AUC score for nominal data ($AUC = 0.960$). Compared to the nominal trained model against PGD ($AUC = 0.937$), it offers effective defence while staying true nominal performance of ($AUC = 0.955$). The corresponding ROC curves are depicted in figure 5.5.

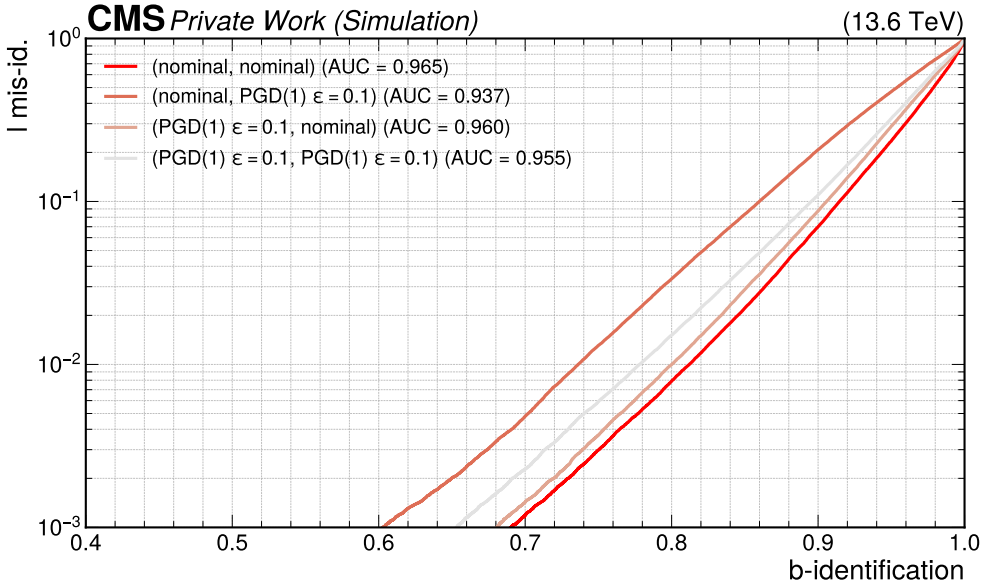


Figure 5.5: ROC curves for BvsL misidentification for a PGD(1) and nominal trained model tested against nominal or PGD(1) perturbed inputs with a magnitude of $\epsilon = 0.1$.

5.2 Probabilistic Integer Perturbation

If not declared specifically a sharpness of $s = 1$ is assumed throughout this section. This value was chosen as it represents a compromise between nominal performance and a full-blown attack (see section 5.4 for more details).

Severity: The stealth characteristics of PIP attacks are fundamentally different from continuous perturbation methods due to their discrete nature. Figures 5.6 illustrates the perturbation severity over the targeted input domain. A comprehensive list of histograms for the input similarity of all features is provided in the Appendix A.2.

The JSD analysis reveals that PIP perturbations maintain higher characteristics compared to PGD, with divergence values typically in the range of $O(10^{-2})$ and some values going as far as $O(10^{-1})$. These values occur as PIP is inherently discrete and its impact is generally dependent on the value range of each feature (e.g. for features such as `Npfcan_isGamma` — a boolean flag — flipping a bit corresponds to a 100% change in the respective input domain).

The comparison of single and double-iteration for PIP attacks — as seen in figures 5.7 and 5.8 — indicates a relatively minimal difference in perturbation severity.

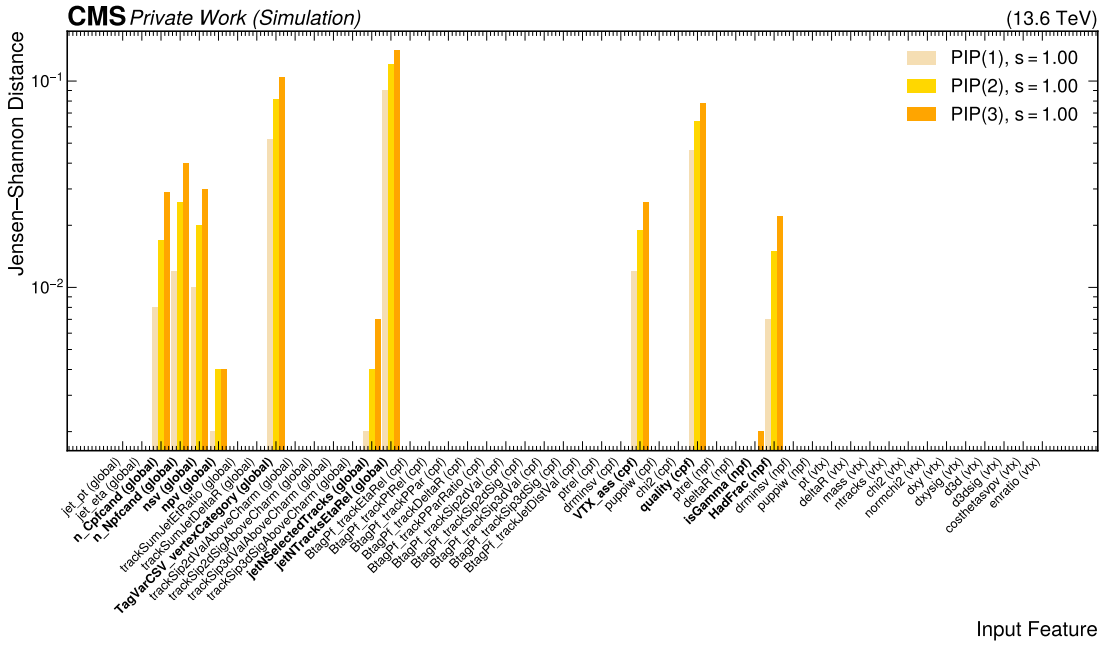
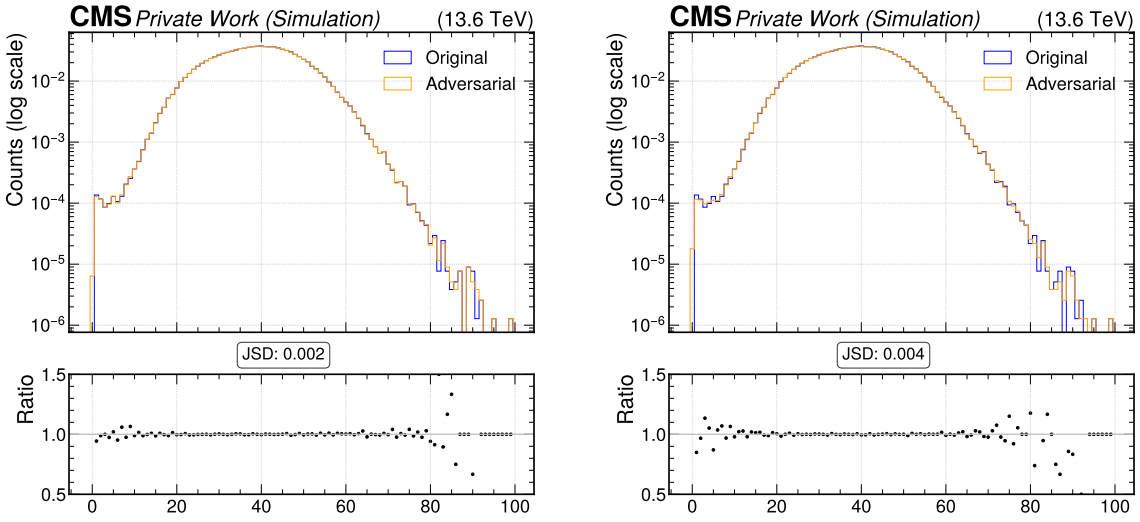


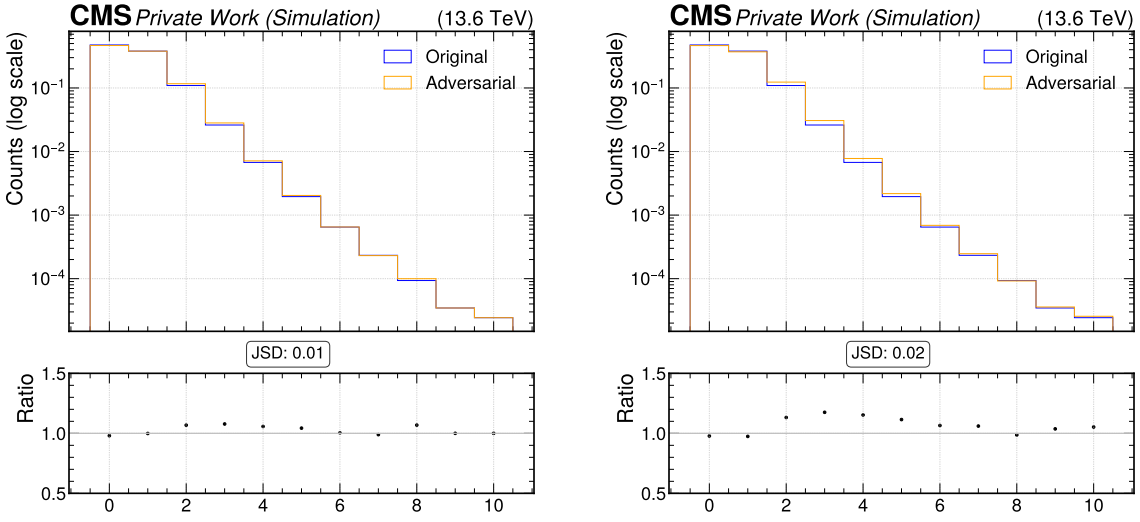
Figure 5.6: JSD input similarity development for up to three iterations of the PIP attack with $s = 1$ compared against a nominal trained model.



(a) Input similarity for PIP(1).

(b) Input similarity for PIP(2).

Figure 5.7: Histogram for global `npv` for one and two iterations (left and right respectively) of PIP with a sharpness of $s = 1$ compared against nominal inputs.



(a) Input similarity for PIP(1).

(b) Input similarity for PIP(2).

Figure 5.8: Histogram for global `nsv` for one and two iterations (left and right respectively) of PIP with a sharpness of $s = 1$ compared against nominal inputs.

Attack: The PIP attack exhibits a distinct characteristic compared to PGD. Its effectiveness depends on the number of iterations and the sharpness parameter in roughly same parts. Figure 5.9 illustrates this relationship, showing how varying sharpness values affect the attack's ability to compromise the classifier:

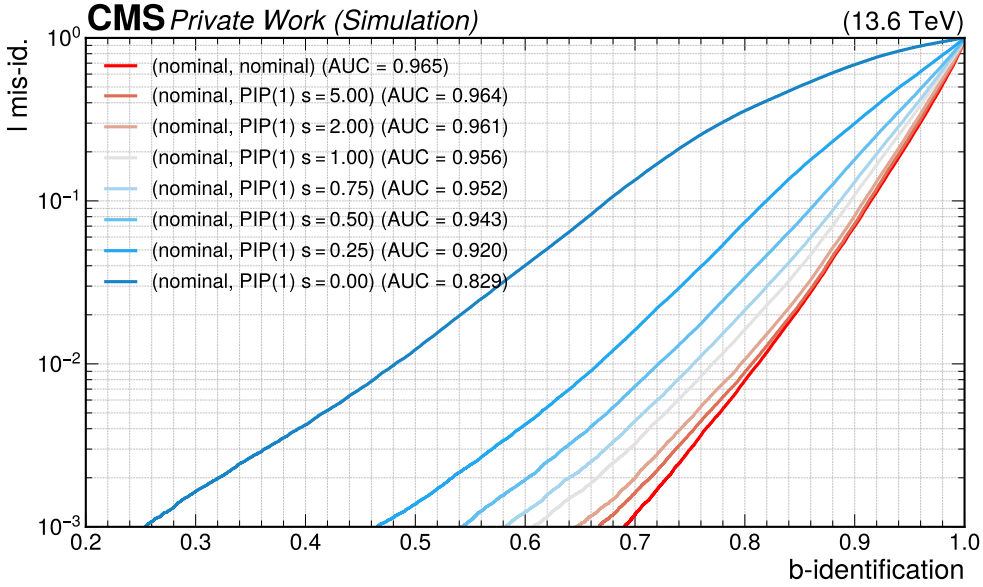


Figure 5.9: ROC curves for BvsL misidentification of the nominal trained model tested against different sharpness values for the PIP attack at one iteration.

- At **high sharpness** ($s = 5$), the attack exhibits minimal impact, with AUC values remaining close to the nominal baseline. This reflects the probabilistic nature of the PIP method—when the sharpness is high, the probability of flipping integer features remains small, resulting in perturbations that are too subtle to significantly affect the classifier's decision boundaries.
- At **moderate sharpness** around $s = 1.0$, a more pronounced degradation becomes apparent, with the AUC dropping to approximately 0.956. This represents the sweet spot where the attack achieves meaningful perturbation while maintaining reasonable stealth characteristics.
- A **low sharpness** of $s = 0$ corresponds to a completely deterministic attack, where all features with a gradient unequal to zero get eventually increased/decreased. It produces the most aggressive attack, reaching an AUC of around 0.829.

Likewise, figure 5.10 highlights PIP's iterative behaviour. Unlike PGD, where additional iterations yield progressively smaller changes, PIP exhibits steady AUC degradation, with subsequent iterations reducing the AUC from 0.956 at one iteration to 0.916 at five iterations.

This behaviour arises from PIP's stochastic nature, where features unaltered in prior iterations gain further opportunities for modification in later ones. This increases their cumulative perturbation probability across iterations.

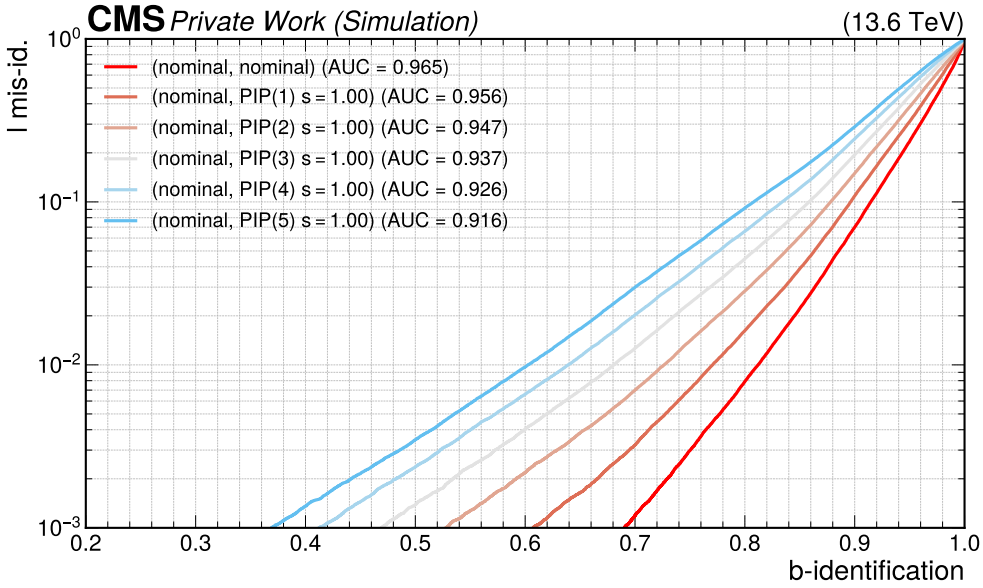


Figure 5.10: ROC curve for BvsL misidentification of the nominal trained model tested against up to five iteration of the PIP attack at $s = 1$.

Adversarial Training: PIP's adversarial training presents a distinct learning scenario compared to continuous perturbation methods. Figure 5.11 shows the training dynamics for PIP with sharpness $s = 1$ over 30 epochs, revealing several interesting characteristics of discrete adversarial training.

The training loss curve demonstrates visible fluctuations, particularly in comparison to PGD-based training. This instability stems from the discrete nature of PIP perturbations — unlike continuous methods that can produce arbitrarily small perturbations, PIP operates on a finite set of possible integer modifications. This creates less but more pronounced perturbations offering a greater challenge in the training procedure. This instability reflects in the validation loss too, indicating problems to generalize for unseen PIP perturbations. Compared to nominal training or PGD, the validation loss slightly exceeds the training loss, suggesting mild adaptation to the lower-severity PIP perturbations (5.12).

Figure 5.12 shows that the PIP-trained model achieves robust performance against PIP-perturbed data, with an AUC of 0.967, slightly surpassing the nominal-trained baseline ($AUC = 0.965$) by approximately 0.2%. When evaluated on nominal data, the model maintains a strong AUC of 0.962, indicating minimal performance degradation compared to the undisturbed model.

Figure 5.11: Training and validation loss for a PIP trained model over 30 epochs with a sharpness of 1.

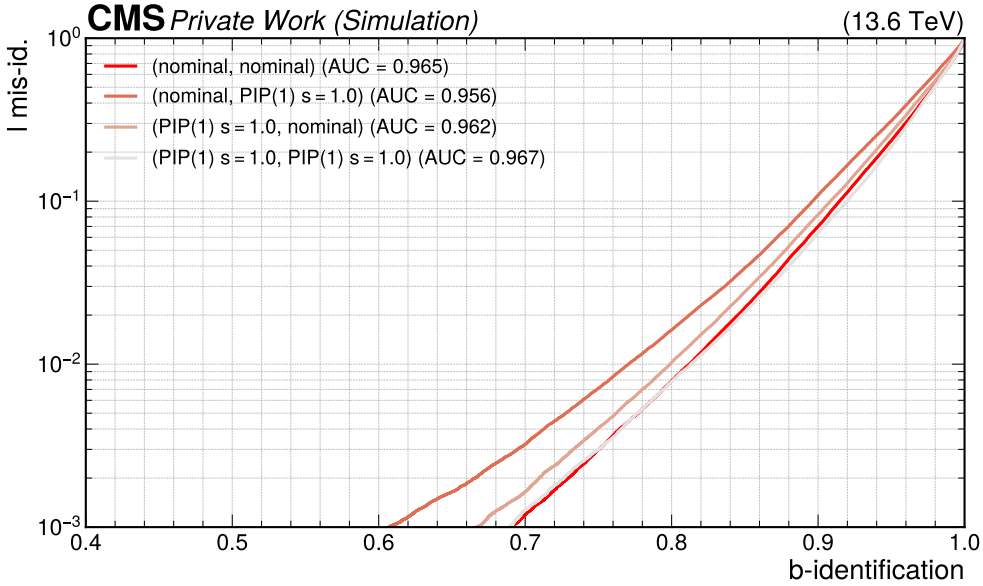


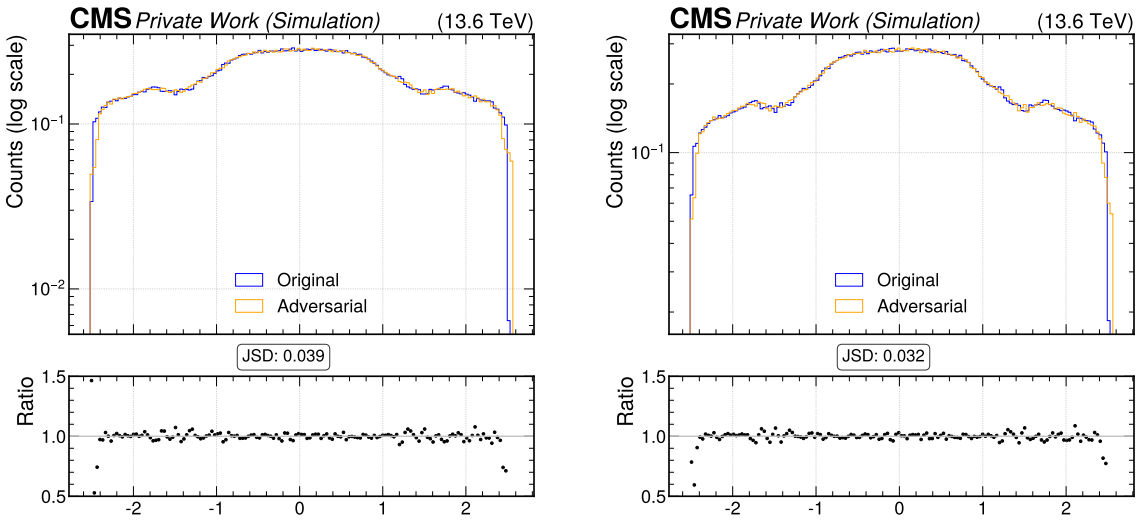
Figure 5.12: ROC curves for BvsL misidentification for a PIP(1) and nominal trained model tested against nominal or PIP(1) perturbed inputs with a sharpness of $s = 1.0$.

5.3 Probabilistic Integer-Perturbed Projected Gradient Descent

As described in Sec. 4.6, the PIP-PGD attack perturbs floating-point and integer features independently in a single pass. The following analysis examines its effect on input similarity, model performance, and training behaviour.

Severity: The JSD analysis reveals that combined attacks retain stealth characteristics comparable to standalone PIP or PGD attacks, with JSD values ranging from $\mathcal{O}(10^{-3})$ to $\mathcal{O}(10^{-1})$ across the entire attack domain. A detailed list of JSD values for individual features is provided in Appendix A.3.

The histograms between one and two iterations for `jet_eta` (float) and `nsv` (integer) is depicted in figures 5.13 and 5.14 respectively. These features were specifically chosen to show the perturbation across discrete and continuous inputs. Their respective JSDs lie approximately within the same order of magnitude.

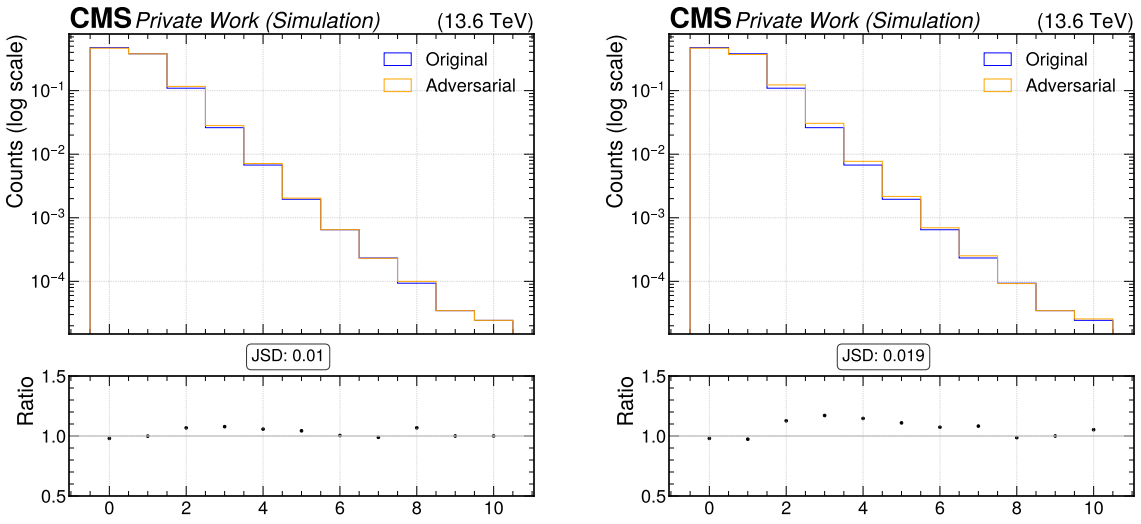


(a) Input similarity for PIP-PGD(1).

(b) Input similarity for PIP-PGD(2).

Figure 5.13: Histogram of input perturbation for continuously-valued global `jet_eta` for one (left) and two (right) iterations of PIP-PGD compared against nominal inputs.

Figure 5.15 provides an overview of input similarity across all features for the combined PIP-PGD attack, highlighting how different feature types respond to the hybrid perturbation strategy. While certain integer features — which are denoted orange in the figure — such as `TagVarCSV_jetNTracksEtaRel`, `TagVarCSV_vertexCategory` or `Cpfcan_quality` exhibit larger perturbations than others, their overall similarity remains comparable.



(a) Input similarity for PIP-PGD(1).

(b) Input similarity for PIP-PGD(2).

Figure 5.14: Histogram of input perturbation for discrete-valued global `nsv` for one (left) and two (right) iterations of PIP-PGD compared against nominal inputs.

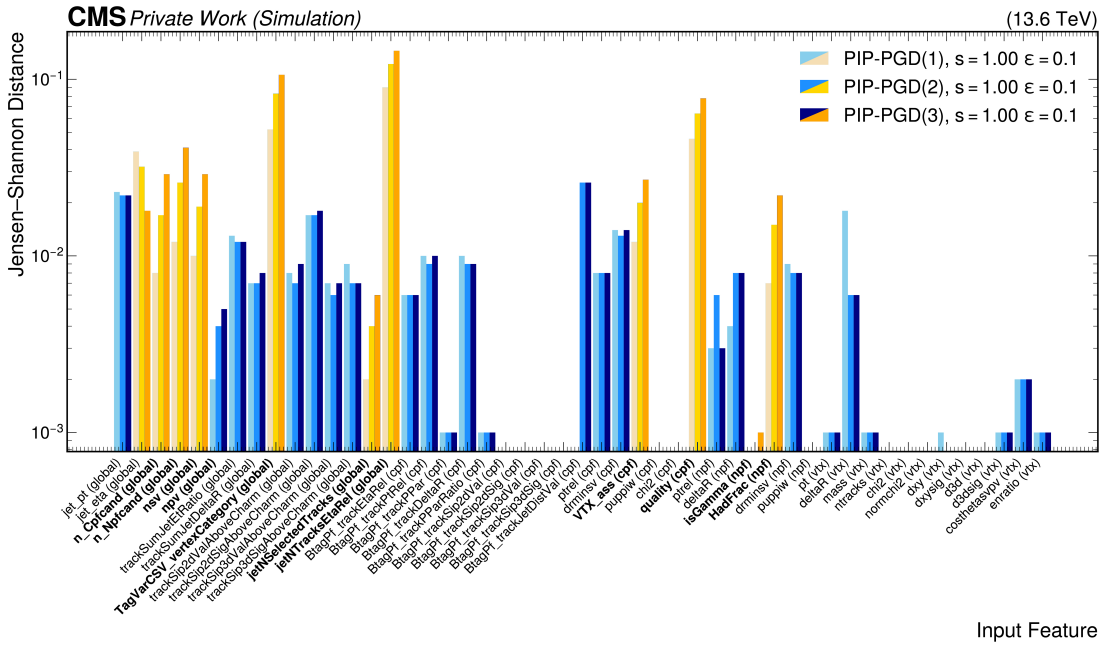


Figure 5.15: JSD input similarity development for different iterations of the PIP-PGD attack for $s = 1$ and $\epsilon = 0.1$ while attributing for individual float features. Orange bars denote integer based features, blue bars correspond to floating-point features. Values below 10^{-1} are not included due to rounding.

The pattern indicates that the combined attack balances stealth and efficacy effectively: integer features introduce targeted modifications atop continuous feature perturbations, enabling a more aggressive perturbation strategy.

Attack: This increased aggression is apparent for variations in the applied sharpness ranging from a slight degradation at $s = 5$ ($AUC = 0.936$) towards a moderate impact at $s = 1$ ($AUC = 0.928$) and a severe degradation at $s = 0.5$ with an AUC of 0.916 as seen in figure 5.16. The gap between nominal performance ($AUC = 0.965$) and PIP at $s = 5$ is predominately caused by PGD(1) ($AUC = 0.937$, see figure 5.5).

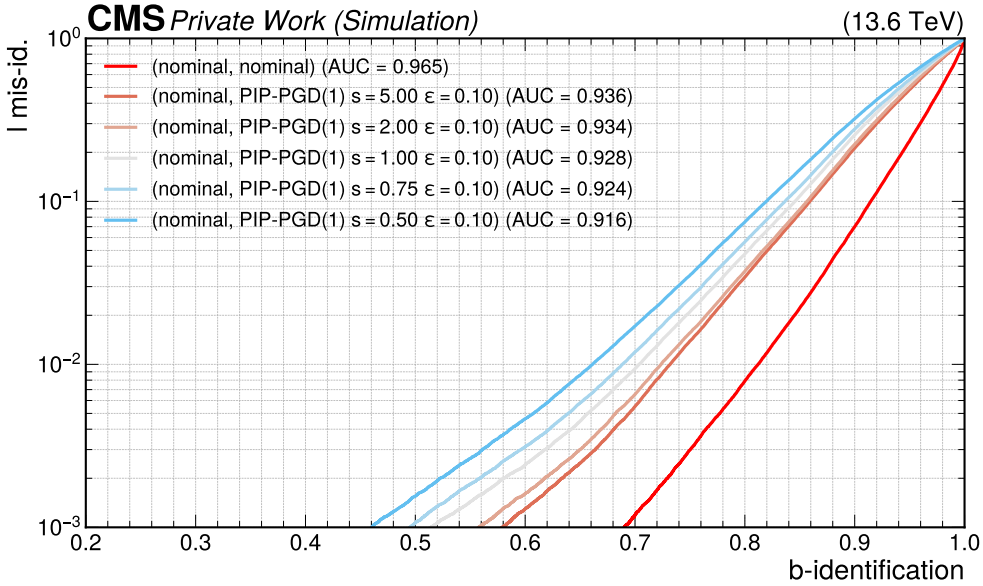


Figure 5.16: AUC score of BvsL misidentification for PIP-PGD with sharpness between $s = 0.5$ and $s = 5$ at one iteration with a magnitude of $\epsilon = 0.10$ tested against the nominal trained model.

Likewise, the AUC score degrades with increasing iterations for the combined PIP-PGD attack from $AUC = 0.928$ at one iteration towards stark inference of $AUC = 0.885$ at five (see figure 5.17). The continued AUC decline for iterations > 1 is largely due to PIP stochastic nature, as discussed in section 5.2.

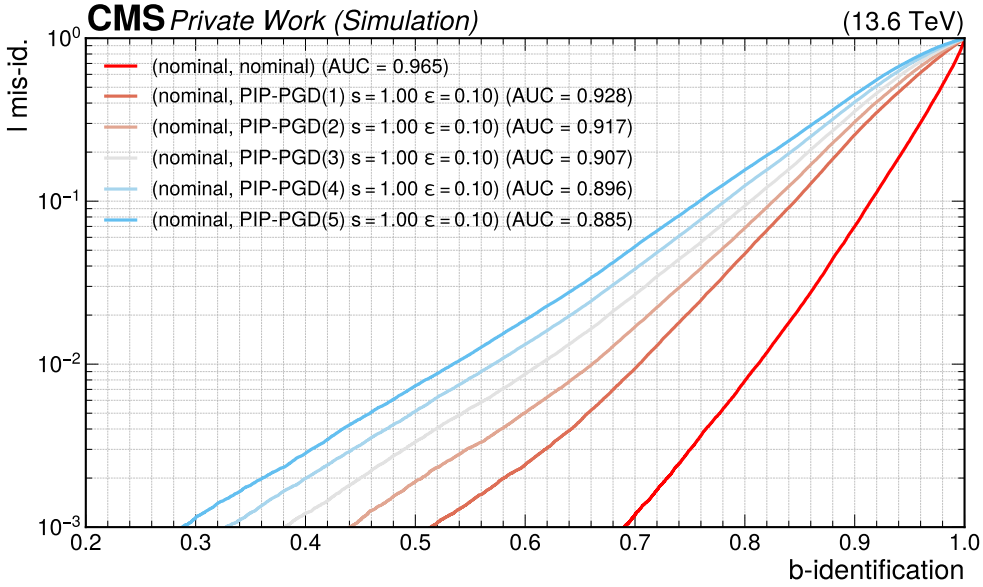


Figure 5.17: AUC score of PIP-PGD for BvsL misidentification for up to 5 iterations with a fixed magnitude of $\epsilon = 0.1$ and a sharpness of $s = 1$ tested against the nominal trained model.

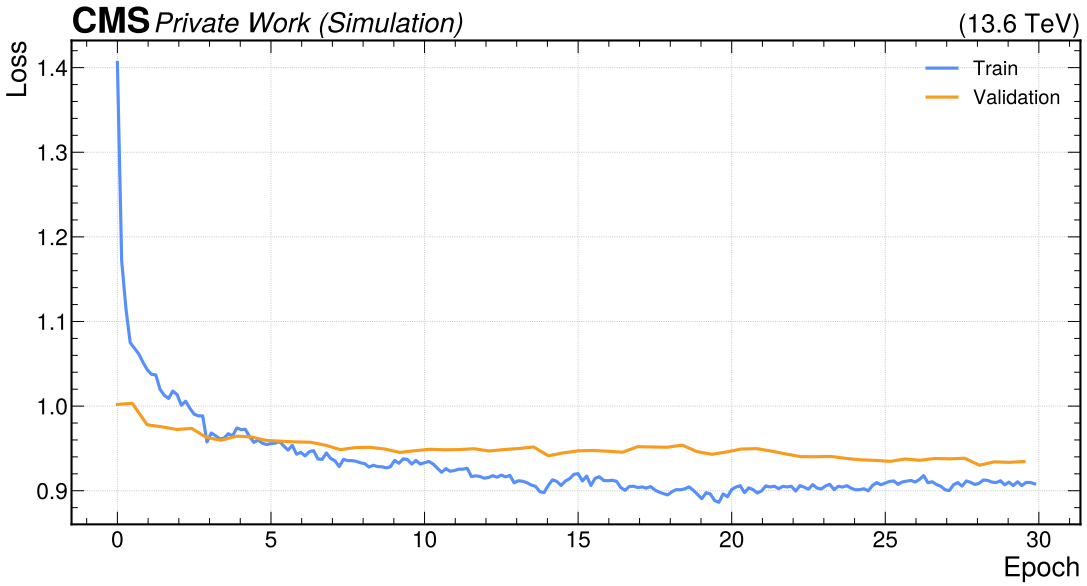


Figure 5.18: Training and validation loss for PIP-PGD(1) with a sharpness of $s = 1$ and a magnitude of $\epsilon = 0.1$, while scaling individual attack features (floats) based on an epsilon tensor.

Adversarial Training: The training loss curve (see figure 5.18) for PIP-PGD exhibits moderate volatility, falling between the stability of pure PGD training and the higher volatility of PIP training . This intermediate behaviour is attributable to the model's need to develop defences that are effective against continuous and probabilistic discrete modifications.

The validation loss demonstrates robust generalization, suggesting that the combined training approach successfully teaches the model to recognize and resist both perturbation types. The convergence pattern indicates that the model learns to balance the competing objectives of maintaining nominal performance while developing comprehensive adversarial defences.

Notably, the final validation loss values are competitive with single-method adversarial training and even surpass pure PIP training. This is an indication that the combined approach does not sacrifice overall performance for broader robustness. It furthermore suggests that the two perturbation types target complementary aspects of the model's decision boundaries, allowing for more comprehensive defence development.

The training efficiency appears reasonable, with convergence achieved within a similar time frame to single-method approaches. This efficiency is particularly important given the increased computational complexity of generating both types of adversarial examples during training.

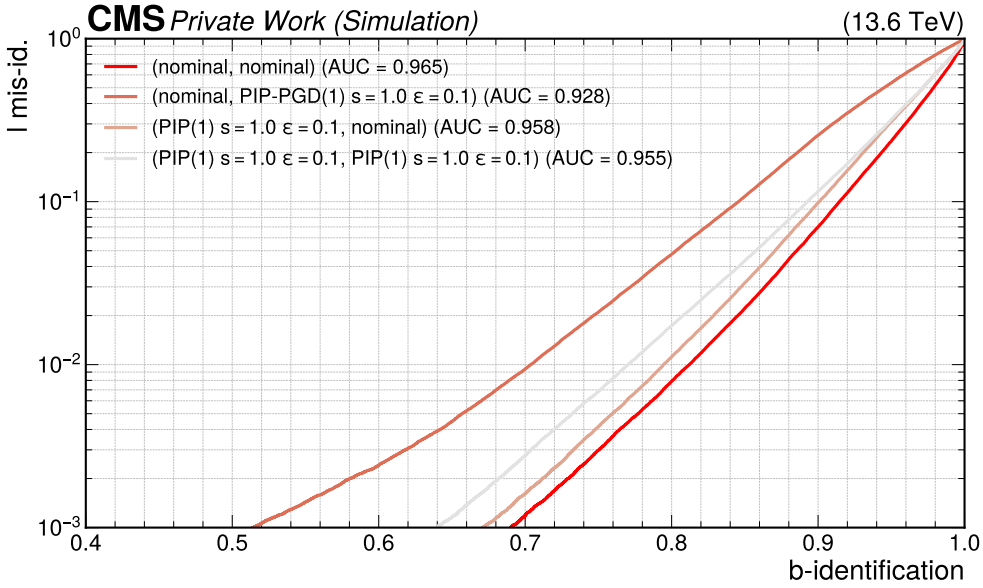


Figure 5.19: ROC curves for BvsL misidentification for a PIP-PGD(1) and nominal trained model tested against nominal or PIP-PGD(1) perturbed inputs with $s = 1.0$ and $\epsilon = 0.1$.

Figure 5.19 shows that the adversarially trained PIP-PGD model, with an AUC of 0.958 against nominal inputs, closely matches the baseline AUC for nominal inputs ($AUC = 0.965$). It also demonstrates robustness across PGD ($AUC = 0.952$), PIP ($AUC = 0.960$), and itself ($AUC = 0.955$). Compared to the nominal-trained model on PIP-PGD perturbed data ($AUC = 0.928$), the PIP-PGD model achieves consistently strong performance across all tested scenarios.

5.4 Transferability and Cross-Robustness

This section concludes the evaluation of PIP by assessing the transferability and cross-robustness of adversarial training strategies, with a focus on the combined PIP-PGD attack. By analysing the interplay of *iteration count* and *sharpness* in PIP's variability and assessing robustness across diverse attack scenarios, it is demonstrated how the hybrid approach balances efficacy and generalization.

Variability of PIP

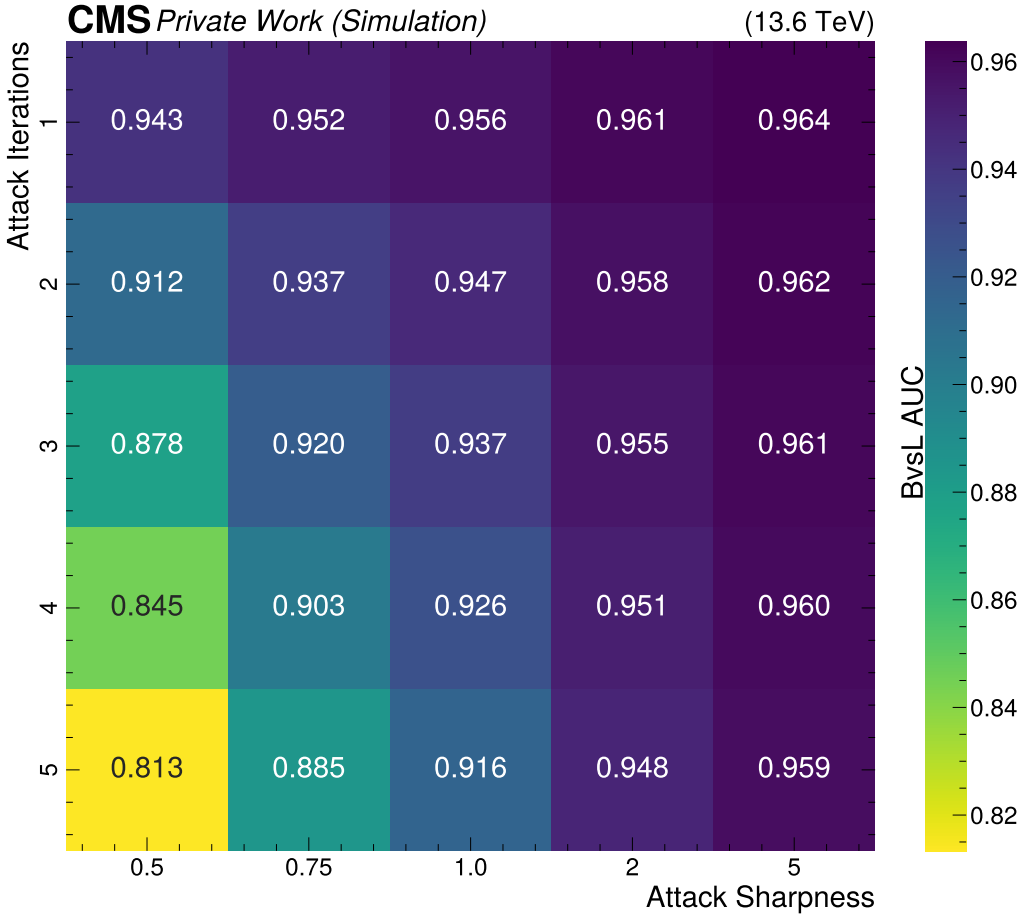


Figure 5.20: AUC score matrix of BvsL misidentification for different iteration and sharpness values in the PIP-PGD attack tested against the nominal trained model.

Figure 5.20 visualises the impact of jointly varying the *iteration count* k (rows) and the *sharpness* parameter s (columns) on the attack efficacy for PIP. For each (k, s) point the resulting BvsL AUC is reported after the attack has been applied to the nominally trained model. Notably, the heatmap is inverted to emphasize stronger degradation from the nominal baseline.

Key trends.

1. **Iteration depth controls raw attack power.** Increasing from $k = 1$ (single-step FGSM/PIP) to $k = 5$ the AUC degrades monotonically, with the most pronounced drop occurring for low-s columns. Concretely, the AUC falls from 0.943 to 0.813 at (s=0.5) — a relative reduction of roughly 13% for only four extra gradient evaluations.

2. **Sharpness s trades off focus against coverage.** As anticipated in section 4.5, large s produces a more sparse integer perturbation pattern, meaning the attack relies mainly on the continuous PGD component. This yields noticeably smaller AUC drops (right-most column remains $\gtrsim$0.959 even at k=5). Conversely, $s \leq 1$ flips a broader set of integer features; the attack strength then scales almost linearly with k.

3. **Region for practical settings.** The contour where AUC $\approx$ 0.937 — the same degradation as for PGD(1) — occurs at (k=3, s=1). Any application of a combined attack should therefore lie below this threshold (so at higher sharpness or less iterations) to not overshadow the application of PGD(1).

The heatmap highlights the flexibility of PIP: by adjusting iteration count k and sharpness s, one can interpolate between subtle, nearly imperceptible perturbations and aggressive attacks that reduce the tagger's performance by $\gtrsim$ 15%, targeting only integer features. The (k, s) plane is non-degenerate, with no single optimal setting; the ideal configuration depends on prioritizing stealth (larger s, smaller k) or maximum performance degradation (smaller s, larger k). Thus, PIP can be tuned to remain subtle when combined with other attacks while enhancing robustness against integer-based corruption, extending existing adversarial algorithms.

Cross-Robustness of PIP-PGD

The last section of this thesis focuses on a cross-robustness study between trained and tested models across all previous mentioned adversarial attacks. For the sake of simplicity, a sharpness of $s = 1$, PGD magnitude of $\epsilon = 0.1$, and only one iteration is discussed here.

Figure 5.21 illustrates a key property of adversarial training with integer-based attacks. The model trained on PIP achieves excellent performance against the same attack ($AUC = 0.967$), slightly outperforming the nominal model on unperturbed data ($AUC = 0.965$). This suggests strong adaptation to the characteristic perturbations of probabilistic integer modifications. However, this enhanced robustness is highly specific to the trained attack pattern. When evaluated against other attacks, such as PGD or PIP-PGD, performance degrades noticeably, indicating a form of adversarial overfitting where the model resists a narrow class of perturbations but struggles to generalize.

In contrast, adversarial training with the PIP-PGD strategy yields more balanced robustness across all tested attack scenarios. Although its AUC on the PIP attack is slightly lower than that of models trained solely on PIP or, respectively, PGD on PGD, the combined model

exhibits better transferability, maintaining robust performance under both PIP ($AUC = 0.960$) and PGD ($AUC = 0.952$) attacks. This suggests that combined adversarial training mitigates attack-specific overfitting, fostering a more generalizable defence.

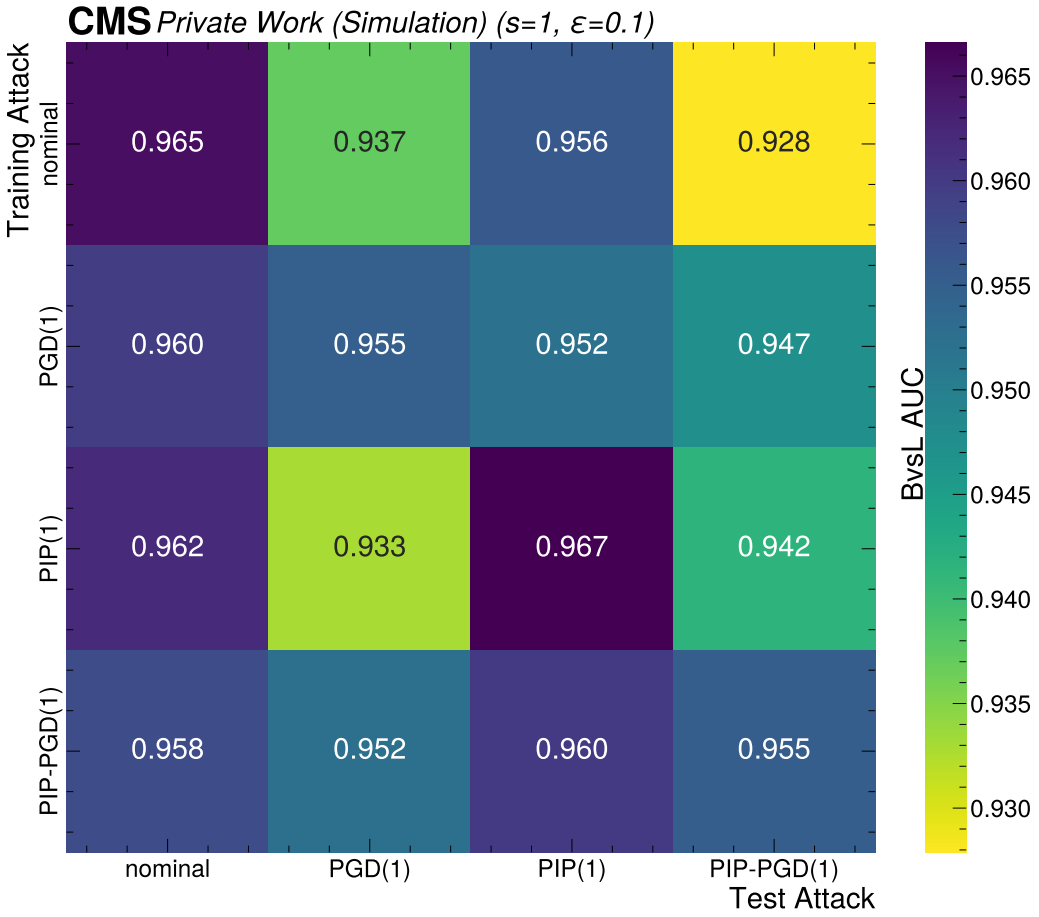


Figure 5.21: AUC score for BvsL misidentification for all combinations of adversarial training and adversarial testing between nominal, PGD(1), PIP(1), and joint PIP-PGD(1) attacks with a sharpness of $s = 1.0$ and $\epsilon = 0.1$ while attributing for individual features.

Models trained solely on PGD struggle to achieve robustness against integer-based attacks, with an AUC of ≤ 0.952, though they remain stable against PGD perturbations.

Overall, the PIP-PGD training strategy offers the best cross-attack robustness while closely preserving nominal performance, with degradation of approximately 1.3% for PGD-perturbed data, 0.5% for PIP-perturbed data, and 1% for PIP-PGD-perturbed data.

CONCLUSION AND OUTLOOK

This thesis has introduced PIP, specifically designed for discrete features in particle physics applications. Applied to the DeepJet flavour tagging algorithm used in CMS, this work demonstrates how to develop a robustness against vulnerabilities in state-of-the-art machine learning models that were previously disregarded.

The key innovation of PIP comes from its probabilistic approach to discrete perturbations without continuous relaxation. Unlike continuous attack methods that generate non-physical fractional values when being applied to integer features, PIP uses gradient information to probabilistically determine whether to increment or decrement discrete features by integer units. This ensures all perturbations remain physically meaningful while effectively probing model robustness. A tunable sharpness parameter controls the attack's aggressiveness, allowing researchers to balance effectiveness with stealth considerations.

The investigation of PIP-PGD attacks reveals synergistic effects exceeding the impact of either method alone. Attacking both, continuous and discrete, features simultaneously exploits complementary vulnerabilities, leading to more severe performance degradation than expected from individual attack impacts.

Adversarial training experiments show that models trained against only one attack type exhibit limited transferability to other attack types, indicating attack-specific overfitting. However, adversarial training with PIP-PGD attacks provides balanced robustness across all tested scenarios while maintaining nominal performance within 1-2% of baseline levels. This combined training approach offers the most comprehensive protection against both continuous and discrete adversarial perturbations.

Beyond the specific application to DeepJet, this work contributes to the broader field of adversarial machine learning by suggesting a general and computational cheap framework for attacking discrete features on top of continuous values that can be adapted to other scientific domains. The successful combination of continuous and discrete attacks demonstrates the importance of considering multiple perturbation modalities in robustness evaluation, particularly relevant for scientific applications involving mixed data types.

While PIP offers a robust method that can be applied in conjunction with existing continuous attack methods, several improvements could enhance its effectiveness. The current implementation does not account for relative feature scaling in the same way that PGD uses feature-specific scaling factors. Incorporating relative scaling based on feature importance or sensitivity has the potential to enhance the efficacy while maintaining physical realism.

The synergy between PGD and PIP could may be attributable to both methods acting on simple gradient information that are generally aligned. However, it remains uncertain whether PIP works equally well with other more sophisticated attack methods. Future research could explore integration with advanced continuous attacks to reveal new synergistic opportunities.

Beyond methodological improvements, PIP could be extended to other scientific machine learning applications where discrete features play important roles. Research areas such as astronomy, materials science, and bioinformatics often involve mixed data types that could benefit from similar robustness analysis approaches. Additionally, incorporating domain-specific knowledge about particle physics has the potential to create more realistic and targeted adversarial perturbations, improving both attack effectiveness and physical interpretability.

BIBLIOGRAPHY

[1] Annika Stein et al. "Improving Robustness of Jet Tagging Algorithms with Adversarial Training". In: *Computing and Software for Big Science* 6.1 (2022), p. 15. ISSN: 2510-2044. DOI: 10.1007/s41781-022-00087-1. URL: https://doi.org/10.1007/s41781-022-00087-1.

[2] The CMS Collaboration. "Observation of Higgs Boson Decay to Bottom Quarks". In: *Phys. Rev. Lett.* 121 (12 Sept. 2018), p. 121801. DOI: 10.1103/PhysRevLett.121.121801. URL: https://link.aps.org/doi/10.1103/PhysRevLett.121.121801.

[3] Ian J. Goodfellow, Jonathon Shlens, and Christian Szegedy. *Explaining and Harnessing Adversarial Examples*. 2015. arXiv: 1412.6572 [stat.ML]. URL: https://arxiv.org/abs/1412.6572.

[4] E. Bols et al. "Jet flavour classification using DeepJet". In: *Journal of Instrumentation* 15.12 (Dec. 2020), P12012. DOI: 10.1088/1748-0221/15/12/P12012. URL: https://dx.doi.org/10.1088/1748-0221/15/12/P12012.

[5] Aleksander Madry et al. *Towards Deep Learning Models Resistant to Adversarial Attacks*. 2019. arXiv: 1706.06083 [stat.ML]. URL: https://arxiv.org/abs/1706.06083.

[6] Antonio Pich. *The Standard Model of Electroweak Interactions*. 2012. arXiv: 1201.0537 [hep-ph]. URL: https://arxiv.org/abs/1201.0537.

[7] Cush. *Standard Model of Elementary Particles*. Public domain image, Wikimedia Commons. 2018. URL: https://commons.wikimedia.org/wiki/File:Standard_Model_of_Elementary_Particles_Anti.svg (visited on 07/13/2025).

[8] The CMS Collaboration. "Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC". In: *Physics Letters B* 716.1 (Sept. 2012), pp. 30–61. ISSN: 0370-2693. DOI: 10.1016/j.physletb.2012.08.021. URL: http://dx.doi.org/10.1016/j.physletb.2012.08.021.

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

[10] The CMS Collaboration. "Performance of the CMS high-level trigger during LHC Run 2". In: *Journal of Instrumentation* 19.11 (Nov. 2024), P11021. ISSN: 1748-0221. DOI: 10.1088/1748-0221/19/11/p11021. URL: http://dx.doi.org/10.1088/1748-0221/19/11/P11021.

[11] Thiago R. F. P. Tomei. *CMS Upgrades for the High-Luminosity LHC Era*. 2025. arXiv: 2501.03412 [hep-ex]. URL: https://arxiv.org/abs/2501.03412.

[12] Esma Mobs. "The CERN accelerator complex - August 2018". In: (2018). General Photo. URL: https://cds.cern.ch/record/2636343.

[13] Jory Sonneveld. *SUSY Highlights: Current Results and Future Prospects*. 2025. arXiv: 2507.16400 [hep-ex]. URL: https://arxiv.org/abs/2507.16400.

[14] The CMS Collaboration. "Development of the CMS detector for the CERN LHC Run 3". In: *Journal of Instrumentation* 19.05 (May 2024), P05064. ISSN: 1748-0221. DOI: 10.1088/1748-0221/19/05/p05064. URL: http://dx.doi.org/10.1088/1748-0221/19/05/P05064.

[15] The CMS Collaboration. "Particle-flow reconstruction and global event description with the CMS detector". In: *Journal of Instrumentation* 12.10 (Oct. 2017), P10003–P10003. ISSN: 1748-0221. DOI: 10.1088/1748-0221/12/10/p10003. URL: http://dx.doi.org/10.1088/1748-0221/12/10/P10003.

[16] The CMS Collaboration. "The CMS trigger system". In: *Journal of Instrumentation* 12.01 (Jan. 2017), P01020–P01020. ISSN: 1748-0221. DOI: 10.1088/1748-0221/12/01/p01020. URL: http://dx.doi.org/10.1088/1748-0221/12/01/P01020.

[17] The CMS Collaboration. "The CMS experiment at the CERN LHC". In: *Journal of Instrumentation* 3.08 (Aug. 2008), S08004. DOI: 10.1088/1748-0221/3/08/S08004. URL: https://dx.doi.org/10.1088/1748-0221/3/08/S08004.

[18] Franck Rothen et al. "Enhancing generalization in high-energy physics using white-box adversarial attacks". In: *Phys. Rev. D* 112 (1 July 2025), p. 016004. DOI: 10.1103/PhysRevD.112.016004. URL: https://link.aps.org/doi/10.1103/PhysRevD.112.016004.

[19] Dan Guest, Kyle Cranmer, and Daniel Whiteson. "Deep Learning and Its Application to LHC Physics". In: *Annual Review of Nuclear and Particle Science* 68.1 (Oct. 2018), pp. 161–181. ISSN: 1545-4134. DOI: 10.1146/annurev-nucl-101917-021019. URL: http://dx.doi.org/10.1146/annurev-nucl-101917-021019.

[20] Kim Albertsson et al. *Machine Learning in High Energy Physics Community White Paper*. 2019. arXiv: 1807.02876 [physics.comp-ph]. URL: https://arxiv.org/abs/1807.02876.

[21] M. M. Hammad. *Artificial Neural Network and Deep Learning: Fundamentals and Theory*. 2024. arXiv: 2408.16002 [cs.LG]. URL: https://arxiv.org/abs/2408.16002.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. Book in preparation for MIT Press. MIT Press, 2016. URL: http://www.deeplearningbook.org.

[23] Omar M. Khalaf and Ahmed M. Hamed. "Particle Identification with Deep Neural Networks Across Collision Energies in Simulated Proton-Proton Collisions". In: (May 2025). arXiv: 2505.06732 [hep-ex].

[24] Luke de Oliveira et al. "Jet-images — deep learning edition". In: *Journal of High Energy Physics* 2016.7 (July 2016). ISSN: 1029-8479. DOI: 10.1007/jhep07(2016)069. URL: http://dx.doi.org/10.1007/JHEP07(2016)069.

[25] Jai Bardhan et al. "Loss function to optimise signal significance in particle physics". In: *38th conference on Neural Information Processing Systems*. Dec. 2024. arXiv: 2412.09500 [hep-ph].

[26] Martin Erdmann et al. *Deep Learning for Physics Research*. WORLD SCIENTIFIC, 2021. DOI: 10.1142/12294. eprint: https://www.worldscientific.com/doi/pdf/10.1142/12294. URL: https://www.worldscientific.com/doi/abs/10.1142/12294.

[27] CMS Collaboration. "A new calibration method for charm jet identification validated with proton-proton collision events at $\sqrt{s} = 13TeV$". In: *Journal of Instrumentation* 17.03 (Mar. 2022), P03014. ISSN: 1748-0221. DOI: 10.1088/1748-0221/17/03/p03014. URL: http://dx.doi.org/10.1088/1748-0221/17/03/P03014.

[28] A. Hoecker et al. *TMVA - Toolkit for Multivariate Data Analysis*. 2009. arXiv: physics/0703039 [physics.data-an]. URL: https://arxiv.org/abs/physics/0703039.

[29] Timo Saala et al. *Enforcing Fundamental Relations via Adversarial Attacks on Input Parameter Correlations*. 2025. arXiv: 2501.05588 [cs.LG]. URL: https://arxiv.org/abs/2501.05588.

[30] David Freedman and Persi Diaconis. "On the histogram as a density estimator: L2 theory". In: *Zeitschrift für Wahrscheinlichkeitstheorie und Verwandte Gebiete* 57.4 (Dec. 1981), pp. 453–476. ISSN: 1432-2064. DOI: 10.1007/BF01025868. URL: https://doi.org/10.1007/BF01025868.

[31] "b-hive: a modular training framework for state-of-the-art object-tagging within the Python ecosystem at the CMS experiment". In: (2024). URL: https://cds.cern.ch/record/2896100.

[32] Han Xu et al. *Probabilistic Categorical Adversarial Attack & Adversarial Training*. 2023. arXiv: 2210.09364 [cs.LG]. URL: https://arxiv.org/abs/2210.09364.

[33] Puyudi Yang et al. *Greedy Attack and Gumbel Attack: Generating Adversarial Examples for Discrete Data*. 2018. arXiv: 1805.12316 [cs.LG]. URL: https://arxiv.org/abs/1805.12316.

APPENDIX

Table A.1: Evaluation metrics used to quantify model performance and perturbation impact.

Metric	Definition / Calculation	Physical Interpretation
ROC / AUC	The Receiver Operating Characteristic (ROC) curve plots the True Positive Rate (TPR, or sensitivity) against the False Positive Rate (FPR) across varying classification thresholds. The Area Under the Curve (AUC) is the integral of the ROC curve, ranging from 0.5 (random classifier) to 1.0 (perfect classifier).	The ROC curve illustrates the trade-off between correctly identifying heavy-flavour jets (TPR) and mistakenly tagging background jets (FPR). A higher AUC indicates better model performance in distinguishing signal from background, critical for robust jet tagging.
ΔAUC (AUC Drop)	Difference in the model's Area Under the ROC Curve (AUC) between nominal (unperturbed) and adversarial datasets. Often reported as a percentage of the nominal AUC.	Measures the loss of discriminative power for jet tagging. A larger drop means the adversarial attack significantly degrades the classifier's ability to distinguish heavy-flavour jets from background (attack success).
Jensen–Shannon Distance (JSD)	The square root of the Jensen–Shannon divergence between the distribution of a given feature for nominal vs. attacked samples (see Equation (4.6)). Computed on each feature's 1D histogram, excluding default/padding values.	Quantifies the *statistical difference* between original and perturbed feature distributions. Low JSD (near 0) indicates minimal change (stealthy), while high JSD reveals visible distribution shifts.

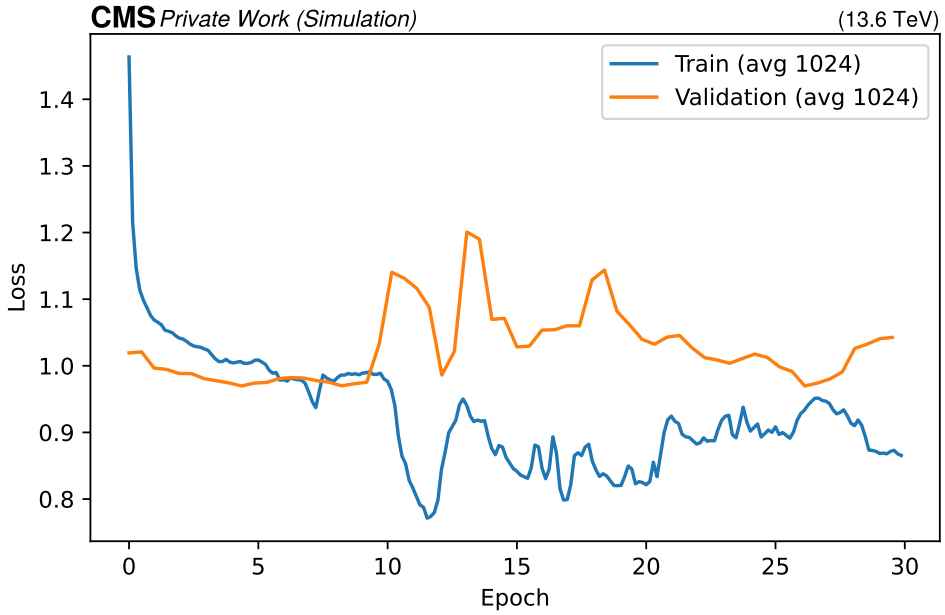


Figure A.1: Training and validation loss for FGSM applied to the entire input spectrum training.

A.1 Input Features

Table A.2: Descriptions of the global input features with applied masks $\mathcal{M}_{\text{float}}$, $\mathcal{M}_{\text{int}}$, $\mathcal{M}^{*}_{\text{int}} \subset \mathcal{M}_{\text{int}}$.

variable name	type	description	mask
jet_pt	float	transverse momentum of the jet in GeV	$\mathcal{M}_{\text{float}}$
jet_eta	float	pseudorapidity of the jet	$\mathcal{M}_{\text{float}}$
n_Cpfcand	int	number of charged PF candidates in the jet	$\mathcal{M}_{\text{int}}$
n_Npfcand	int	number of neutral PF candidates in the jet	$\mathcal{M}_{\text{int}}$
nsv	int	number of secondary vertices in the jet	$\mathcal{M}_{\text{int}}$
npv	int	number of primary vertices in the event	$\mathcal{M}_{\text{int}}$
TagVarCSV_trackSumJetEtRatio	float	transverse energy ratio of tracks and jet	$\mathcal{M}_{\text{float}}$
TagVarCSV_trackSumJetDeltaR	float	spatial distance between the sum of tracks and jet axis	$\mathcal{M}_{\text{float}}$
TagVarCSV_vertexCategory	int	secondary vertex category index	$\mathcal{M}_{\text{int}}$
TagVarCSV_trackSip2dValAboveCharm	float	2D signed impact parameter of first track lifting mass above charm in cm	$\mathcal{M}_{\text{float}}$
TagVarCSV_trackSip2dSigAboveCharm	float	2D signed impact parameter significance of first track lifting mass above charm	$\mathcal{M}_{\text{float}}$
TagVarCSV_trackSip3dValAboveCharm	float	3D signed impact parameter of first track lifting mass above charm in cm	$\mathcal{M}_{\text{float}}$
TagVarCSV_trackSip3dSigAboveCharm	float	3D signed impact parameter significance of first track lifting mass above charm	$\mathcal{M}_{\text{float}}$
TagVarCSV_jetNSelectedTracks	int	number of tracks associated to jet	$\mathcal{M}_{\text{int}}$
TagVarCSV_jetNTracksEtaRel	int	number of tracks associated to jet for which η_{rel} is computed	$\mathcal{M}_{\text{int}}$

Table A.3: Descriptions of the CPF input features with applied masks $\mathcal{M}_{\text{float}}$, $\mathcal{M}_{\text{int}}$, $\mathcal{M}^*_{\text{int}} \subset \mathcal{M}_{\text{int}}$.

variable name	type	description	mask
Cpfcan_BtagPf_trackEtaRel	float	pseudorapidity of charged track relative to jet axis	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackPtRel	float	momentum of charged track transverse to jet axis in GeV	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackPPar	float	momentum of charged track along jet axis in GeV	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackDeltaR	float	spatial distance between track and jet axis	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackPParRatio	float	momentum fraction of charged track along jet axis	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackSip2dVal	float	2D signed impact parameter of charged track in cm	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackSip2dSig	float	2D signed impact parameter significance of charged track	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackSip3dVal	float	3D signed impact parameter of charged track in cm	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackSip3dSig	float	3D signed impact parameter significance of charged track	$\mathcal{M}_{\text{float}}$
Cpfcan_BtagPf_trackJetDistVal	float	minimum distance between track and jet axis	$\mathcal{M}_{\text{float}}$
Cpfcan_ptrel	float	transverse momentum fraction of the track	$\mathcal{M}_{\text{float}}$
Cpfcan_drminsv	float	spatial distance between the track and the closest secondary vertex	$\mathcal{M}_{\text{float}}$
Cpfcan_VTX_ass	int	integer flag: usage of track in primary vertex fit	$\mathcal{M}_{\text{int}}^*$
Cpfcan_puppiw	int	PUPPI weight of the charged PF candidate	$\mathcal{M}_{\text{int}}$
Cpfcan_chi2	int	χ^2 of charged track fit	$\mathcal{M}_{\text{int}}$
Cpfcan_quality	int	integer flag: quality of fitted track	$\mathcal{M}_{\text{int}}^*$

Table A.4: Descriptions of the NPF input features with applied masks $\mathcal{M}_{\text{float}}$, $\mathcal{M}_{\text{int}}$, $\mathcal{M}^*_{\text{int}} \subset \mathcal{M}_{\text{int}}$.

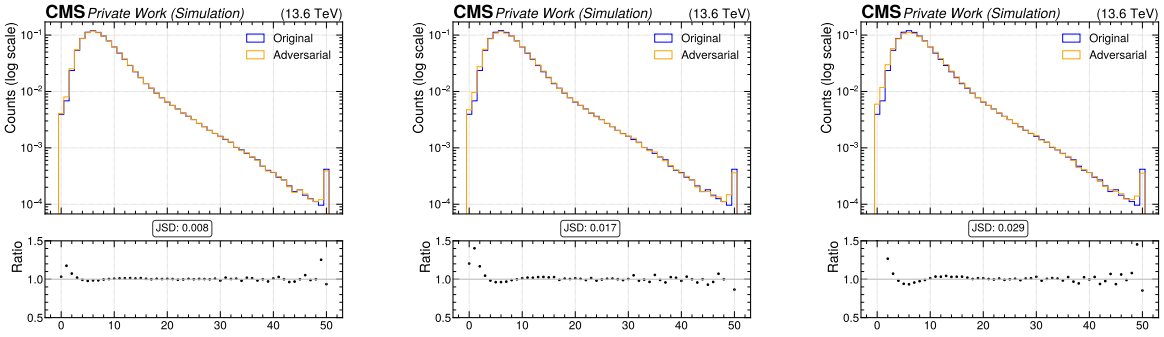
variable name	type	description	mask
Npfcan_ptrel	float	transverse momentum fraction of the track	$\mathcal{M}_{\text{float}}$
Npfcan_deltaR	float	spatial distance between track and jet axis	$\mathcal{M}_{\text{float}}$
Npfcan_isGamma	int	integer flag: particle identity (1 = photon, 0 = no photon)	$\mathcal{M}_{\text{int}}^*$
Npfcan_HadFrac	int	fraction of the energy deposited in the hadronic calorimeter	$\mathcal{M}_{\text{int}}^*$
Npfcan_drminsv	float	spatial distance between the track and the closest secondary vertex	$\mathcal{M}_{\text{float}}$
Npfcan_puppiw	int	PUPPI weight of the neutral PF candidate	$\mathcal{M}_{\text{int}}$

Table A.5: Descriptions of the SV input features with applied masks $\mathcal{M}_{\text{float}}$, $\mathcal{M}_{\text{int}}$, $\mathcal{M}^{*}_{\text{int}} \subset \mathcal{M}_{\text{int}}$.

variable name	data type	description	mask
sv_pt	float	transverse momentum of the SV in GeV	$\mathcal{M}_{\text{float}}$
sv_deltaR	float	spatial distance between SV and jet axis	$\mathcal{M}_{\text{float}}$
sv_mass	float	mass of the SV in GeV	$\mathcal{M}_{\text{float}}$
sv_ntracks	int	number of tracks associated to the SV	$\mathcal{M}_{\text{int}}$
sv_chi2	int	χ^2 of the SV fit	$\mathcal{M}_{\text{int}}$
sv_normchi2	int	χ^2 divided by the degrees of freedom	$\mathcal{M}_{\text{int}}$
sv_dxy	float	2D flight distance of SV in cm	$\mathcal{M}_{\text{float}}$
sv_dxysig	float	2D flight distance significance of SV	$\mathcal{M}_{\text{float}}$
sv_d3d	float	3D flight distance of SV in cm	$\mathcal{M}_{\text{float}}$
sv_d3dsig	float	3D flight distance significance of SV	$\mathcal{M}_{\text{float}}$
sv_costhetasvpv	float	cosine of the angle between SV flight direction and SV momentum	$\mathcal{M}_{\text{float}}$
sv_enratio	float	energy fraction of the SV	$\mathcal{M}_{\text{float}}$

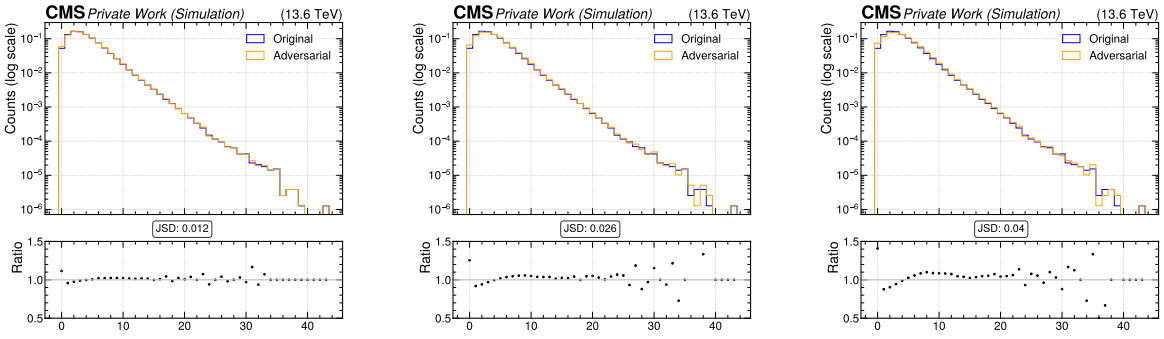
A.2 PIP Input Similarities

Global Features



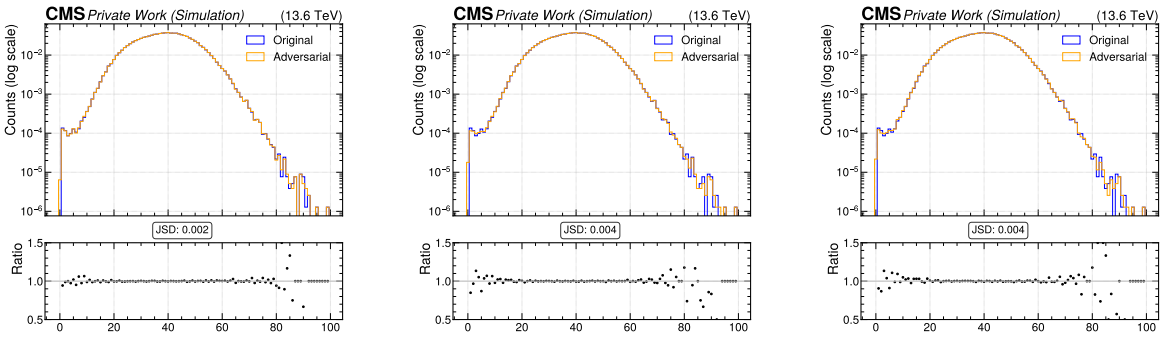
(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

Figure A.2: Histogram of `n_Cpfcand` for multiple iterations of PIP tested against nominal inputs.



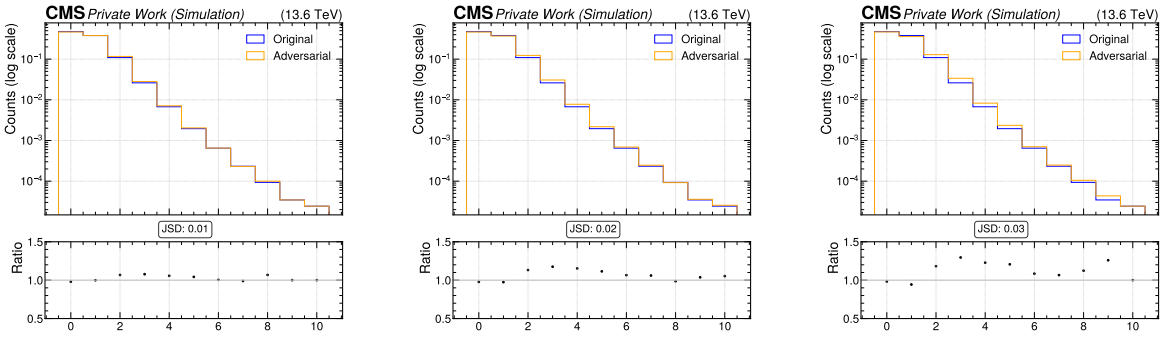
(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

Figure A.3: Histogram of `n_Npfcand` for multiple iterations of PIP tested against nominal inputs.



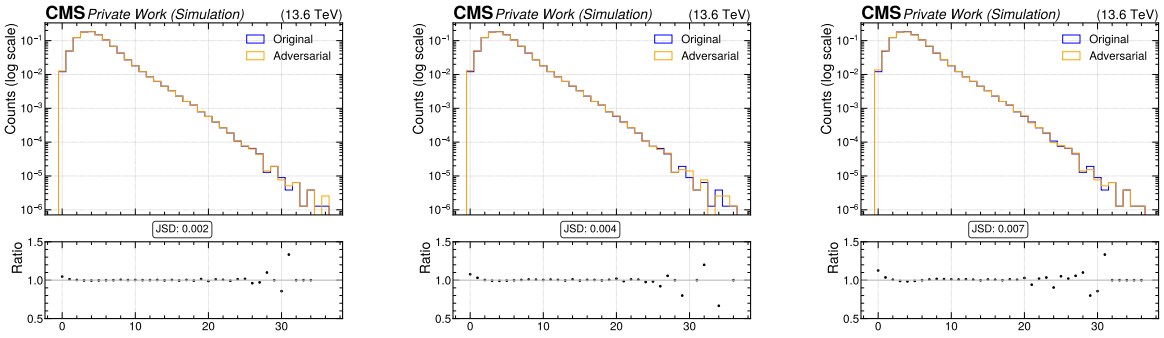
(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

Figure A.4: Histogram of `npv` for multiple iterations of PIP tested against nominal inputs.



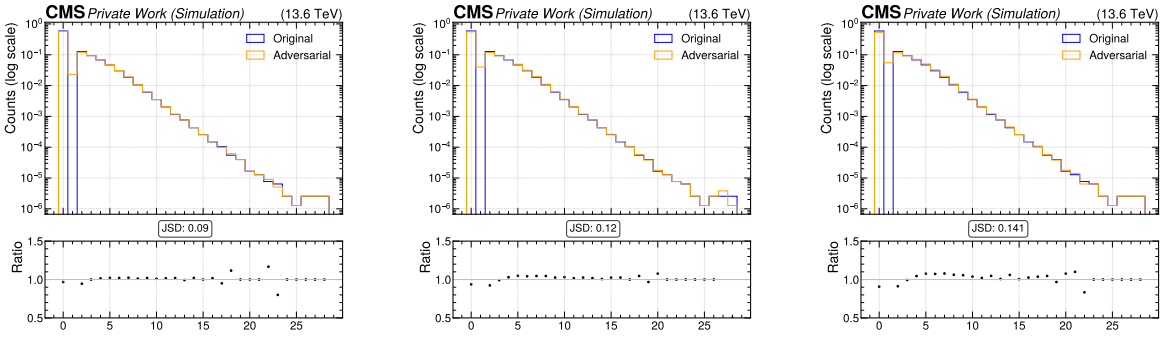
(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

Figure A.5: Histogram of `nsv` for multiple iterations of PIP tested against nominal inputs.



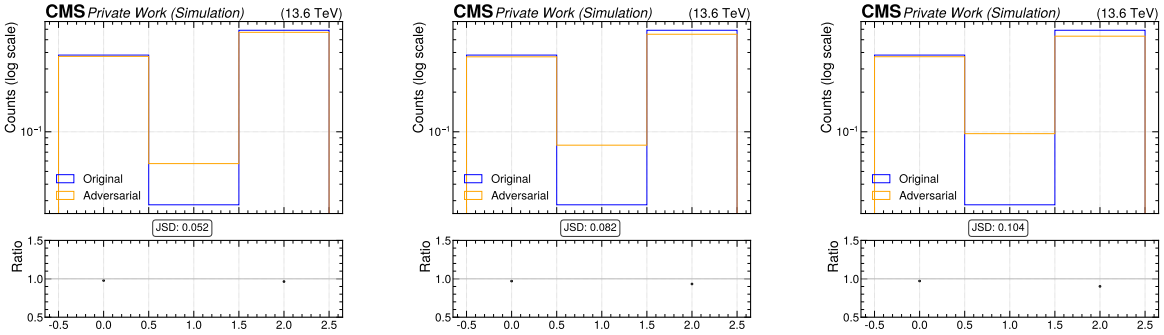
(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

Figure A.6: Histogram of `TagVarCSV_jetNSelectedTracks` for multiple iterations of PIP tested against nominal inputs.



(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

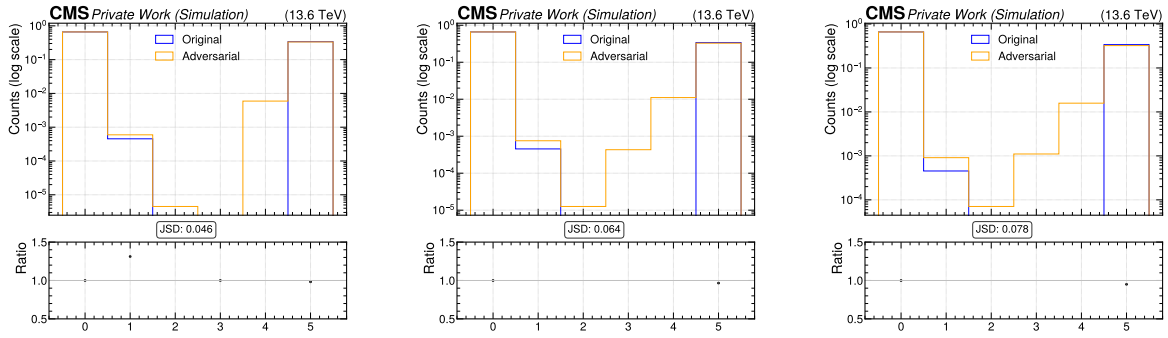
Figure A.7: Histogram of `TagVarCSV_jetNTracksEtaRel` for multiple iterations of PIP tested against nominal inputs.



(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

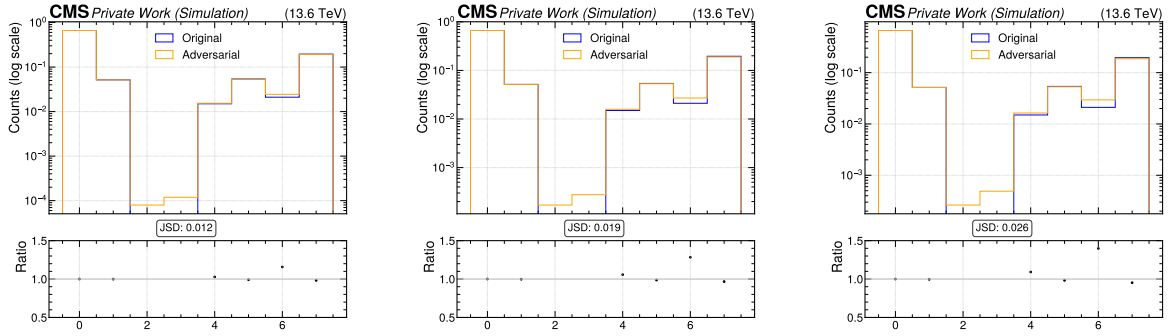
Figure A.8: Histogram of `TagVarCSV_vertexCategory` for multiple iterations of PIP tested against nominal inputs.

CPF Features



(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

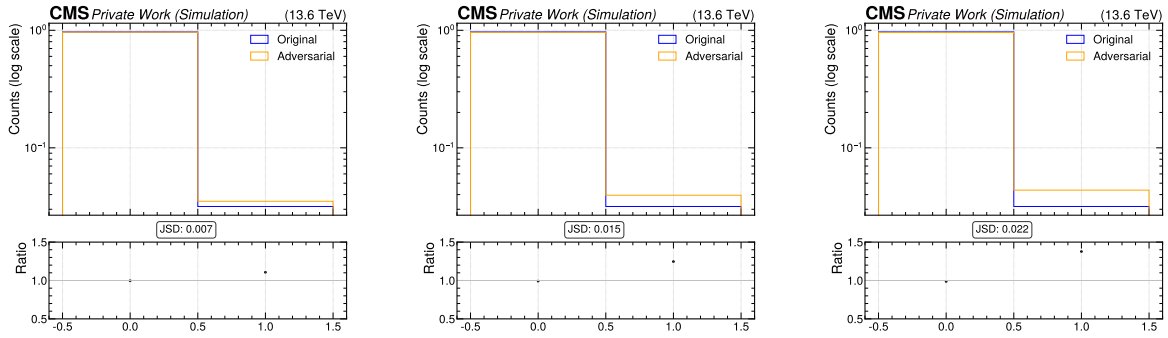
Figure A.9: Histogram of `Cpfcan_quality` for multiple iterations of PIP tested against nominal inputs.



(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

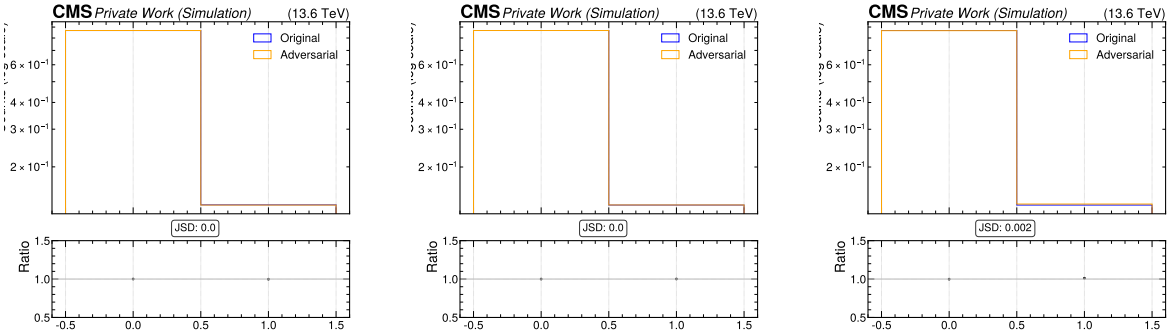
Figure A.10: Histogram of `Cpfcan_VTX_ass` for multiple iterations of PIP tested against nominal inputs.

NPF Features



(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

Figure A.11: Histogram of `Npfcan_HadFrac` for multiple iterations of PIP tested against nominal inputs.

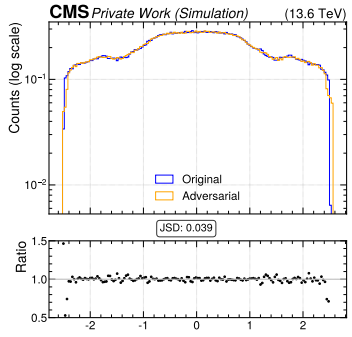


(a) Input similarity for PIP(1). (b) Input similarity for PIP(2). (c) Input similarity for PIP(3).

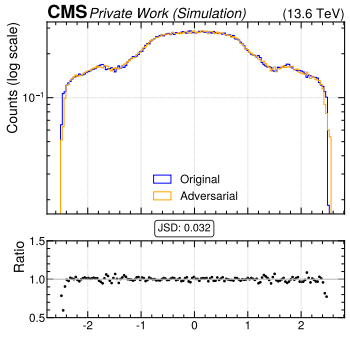
Figure A.12: Histogram of `Npfcan_isGamma` for multiple iterations of PIP tested against nominal inputs.

A.3 PIP-PGD Input Similarities

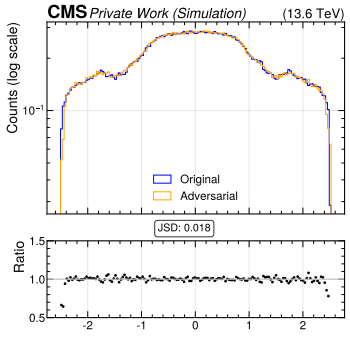
Global Features



Input similarity for PIP-PGD(1).

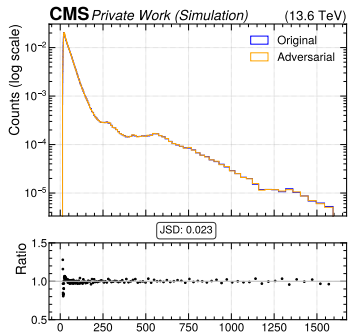


Input similarity for PIP-PGD(2).

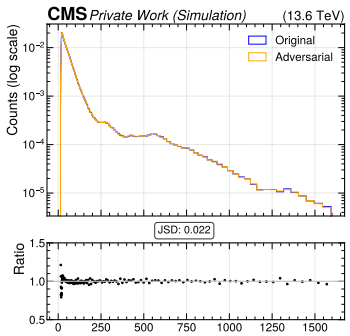


Input similarity for PIP-PGD(3).

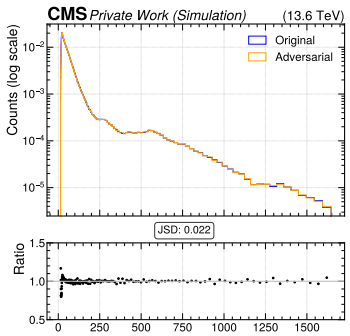
Histogram of `jet_eta` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

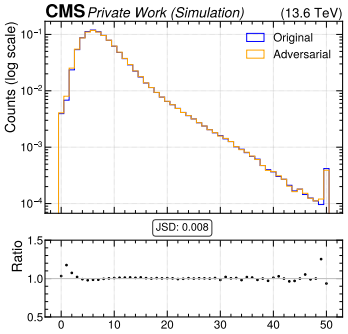


Input similarity for PIP-PGD(2).

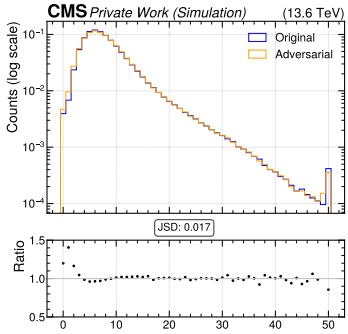


Input similarity for PIP-PGD(3).

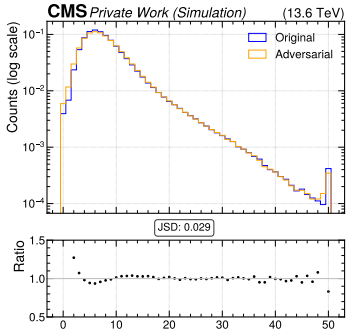
Histogram of `jet_pt` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

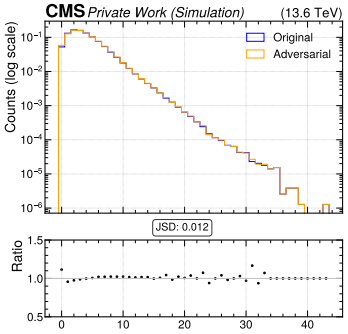


Input similarity for PIP-PGD(2).

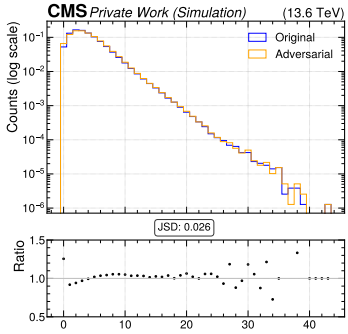


Input similarity for PIP-PGD(3).

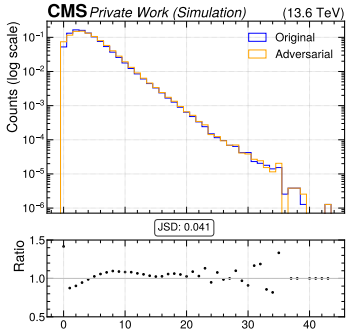
Histogram of `n_Cpfcand` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

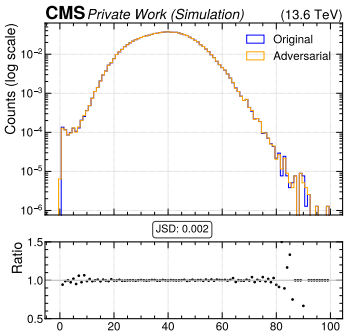


Input similarity for PIP-PGD(2).

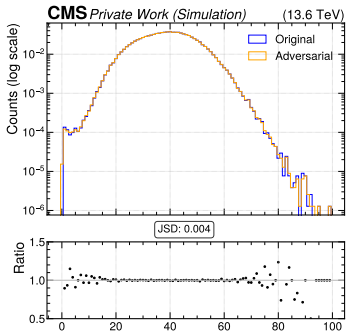


Input similarity for PIP-PGD(3).

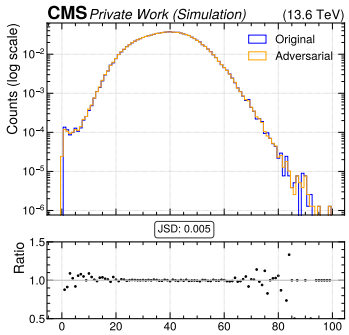
Histogram of `n_Npfcand` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

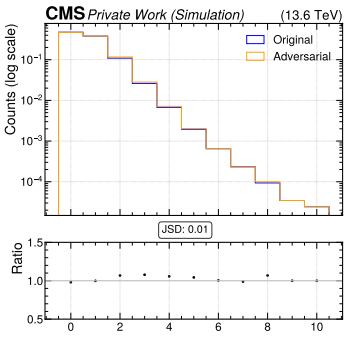


Input similarity for PIP-PGD(2).

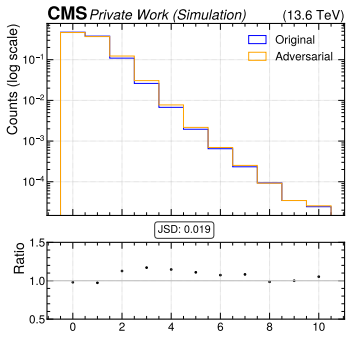


Input similarity for PIP-PGD(3).

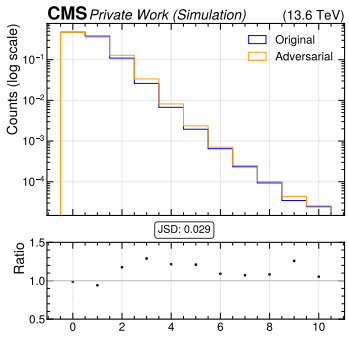
Histogram of `npv` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

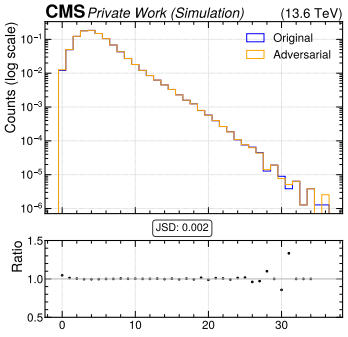


Input similarity for PIP-PGD(2).

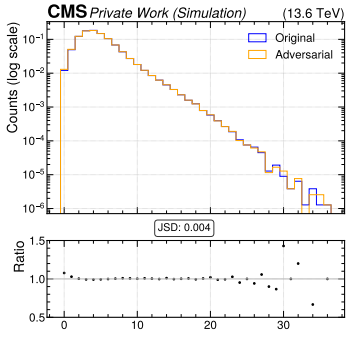


Input similarity for PIP-PGD(3).

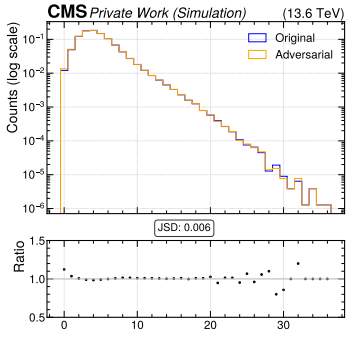
Histogram of `nsv` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

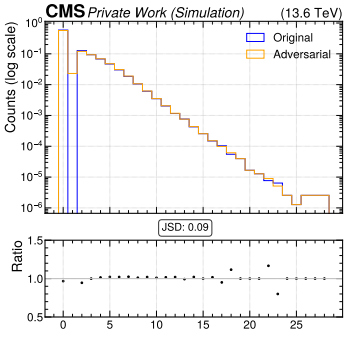


Input similarity for PIP-PGD(2).

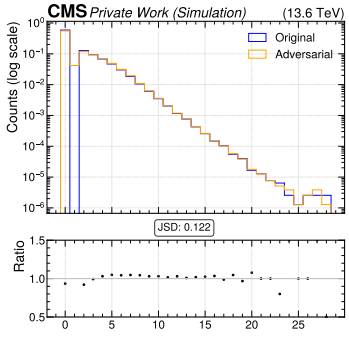


Input similarity for PIP-PGD(3).

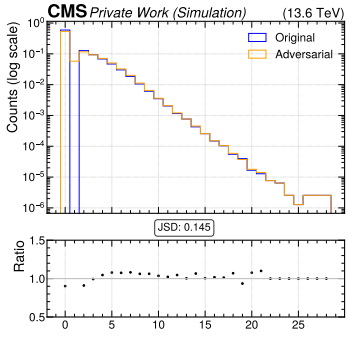
Histogram of `TagVarCSV_jetNSelectedTracks` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

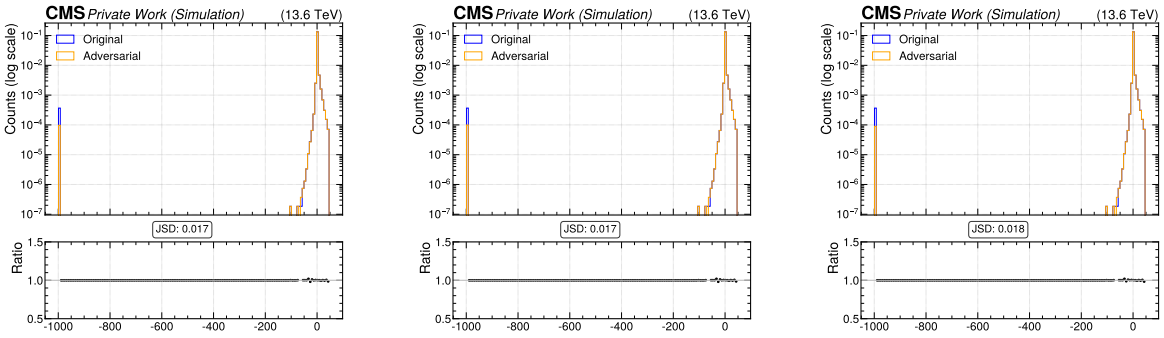


Input similarity for PIP-PGD(2).



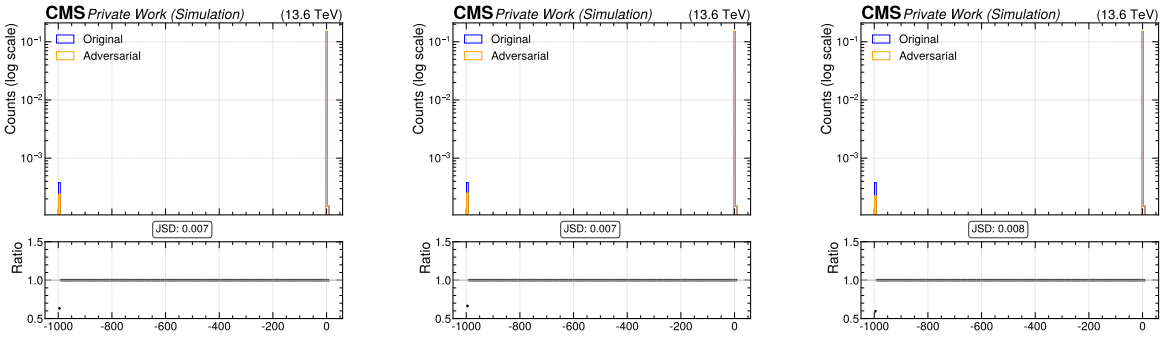
Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_jetNTracksEtaRel` for multiple iterations of PIP-PGD tested against nominal inputs.



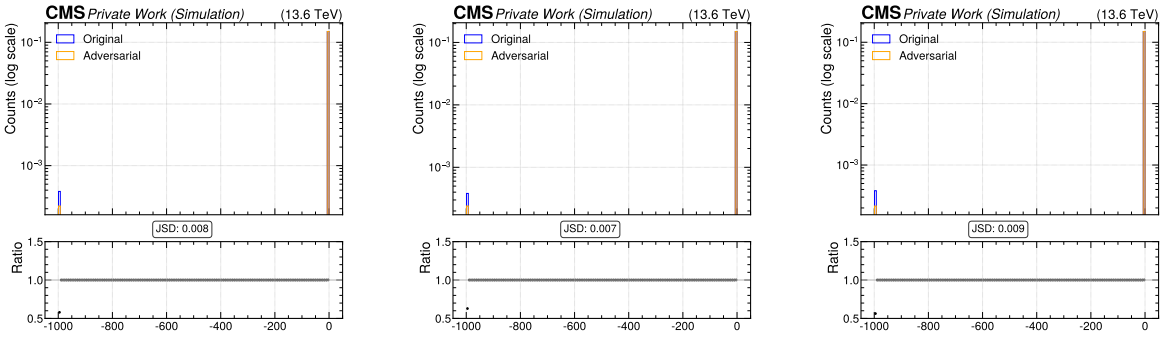
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_trackSip2dSigAboveCharm` for multiple iterations of PIP-PGD tested against nominal inputs.



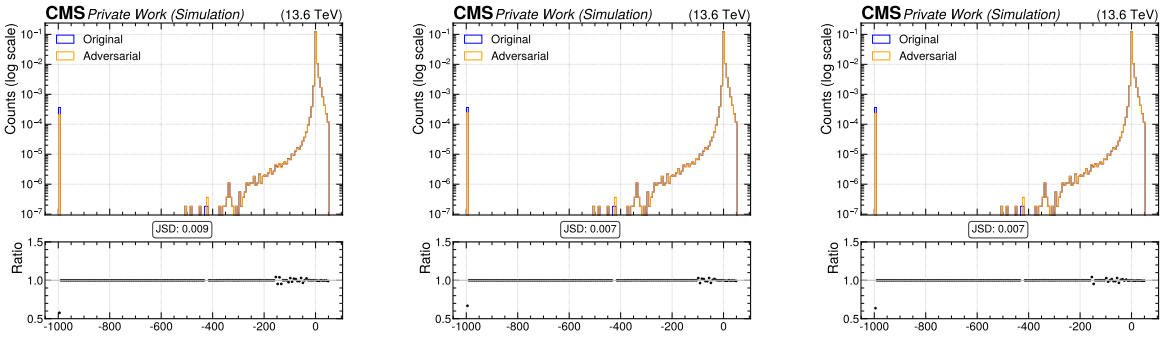
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_trackSumJetDeltaR` for multiple iterations of PIP-PGD tested against nominal inputs.



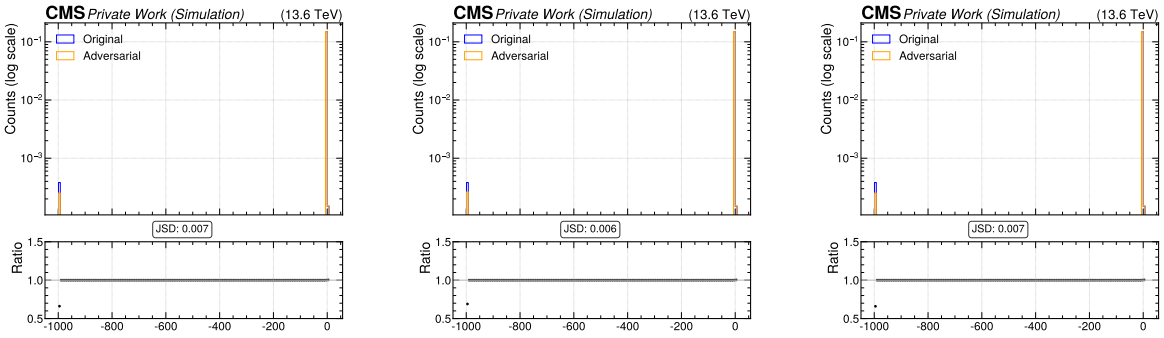
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_trackSip2dValAboveCharm` for multiple iterations of PIP-PGD tested against nominal inputs.



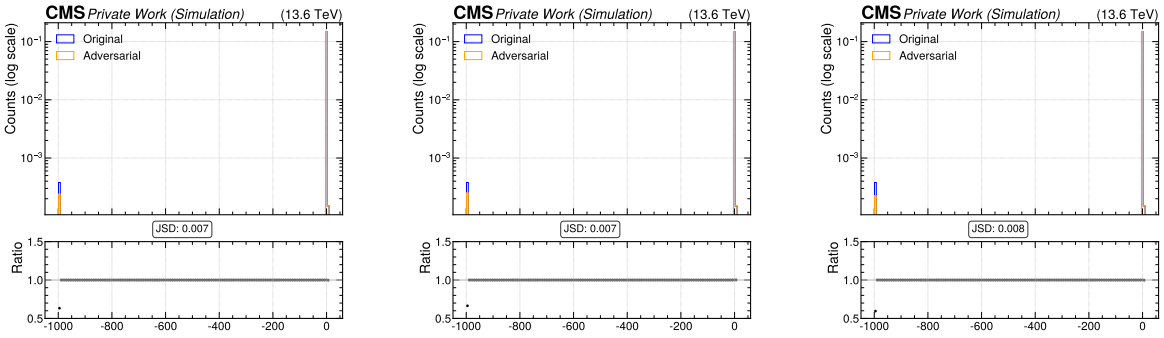
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_trackSip3dSigAboveCharm` for multiple iterations of PIP-PGD tested against nominal inputs.



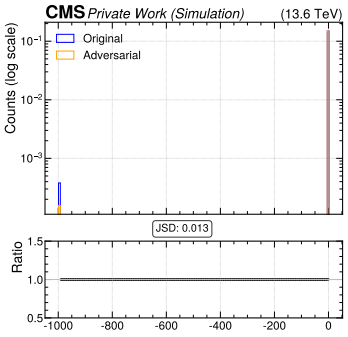
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_trackSip3dValAboveCharm` for multiple iterations of PIP-PGD tested against nominal inputs.

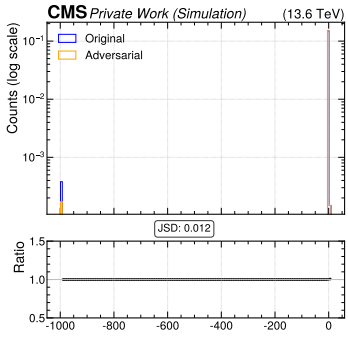


Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

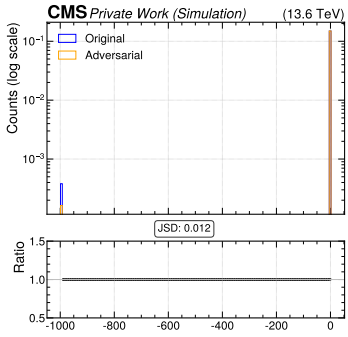
Histogram of `TagVarCSV_trackSumJetDeltaR` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

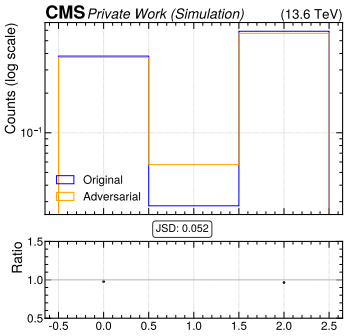


Input similarity for PIP-PGD(2).

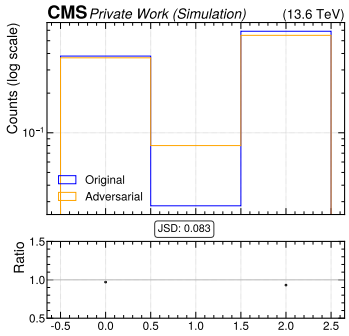


Input similarity for PIP-PGD(3).

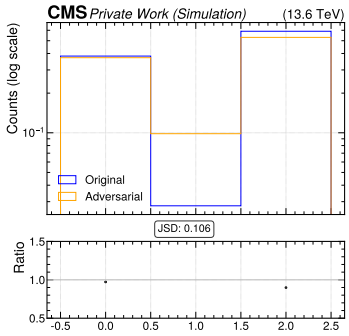
Histogram of `TagVarCSV_trackSumJetEtRatio` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).



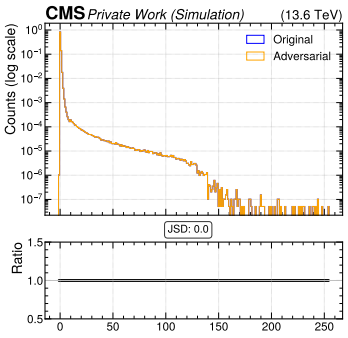
Input similarity for PIP-PGD(2).



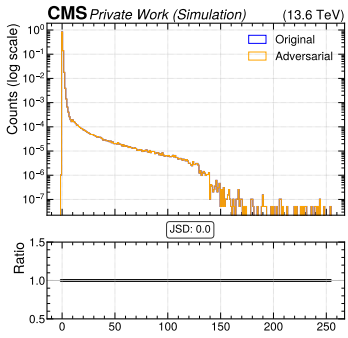
Input similarity for PIP-PGD(3).

Histogram of `TagVarCSV_vertexCategory` for multiple iterations of PIP-PGD tested against nominal inputs.

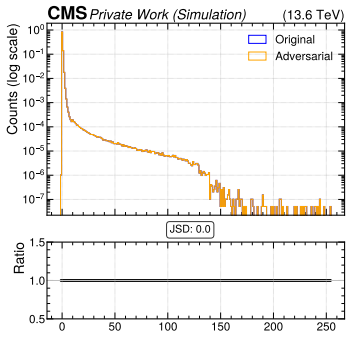
CPF Features



Input similarity for PIP-PGD(1).

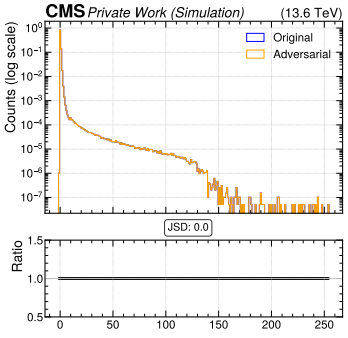


Input similarity for PIP-PGD(2).

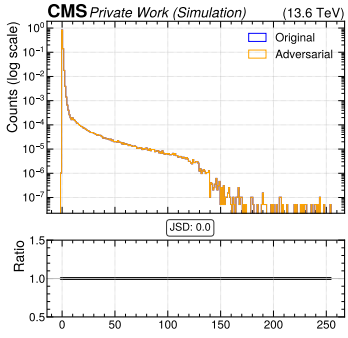


Input similarity for PIP-PGD(3).

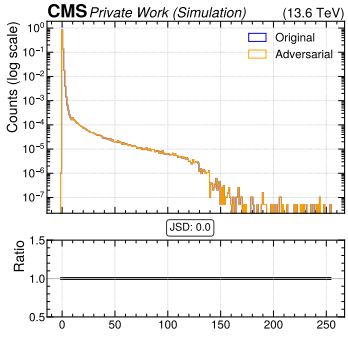
Histogram of `Cpfcan_chi2` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

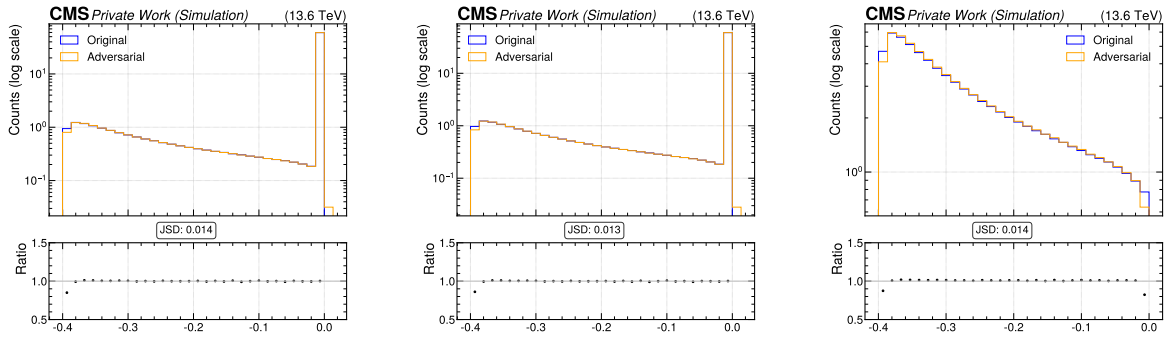


Input similarity for PIP-PGD(2).



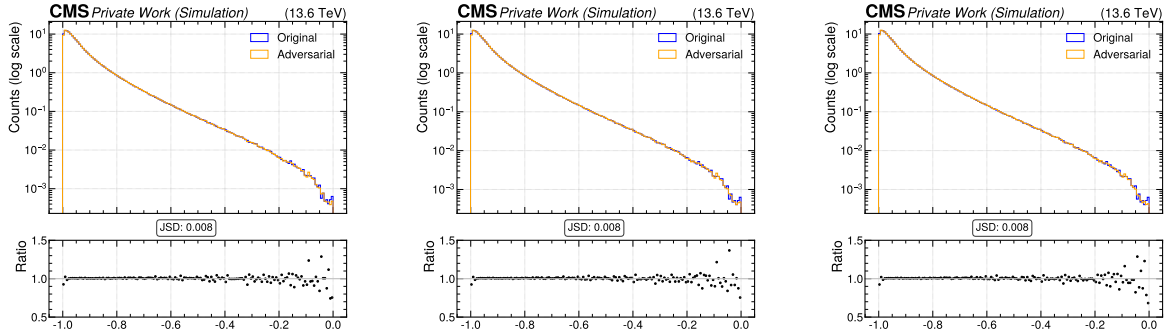
Input similarity for PIP-PGD(3).

Histogram of `Cpfcan_chi2` for multiple iterations of PIP-PGD tested against nominal inputs.



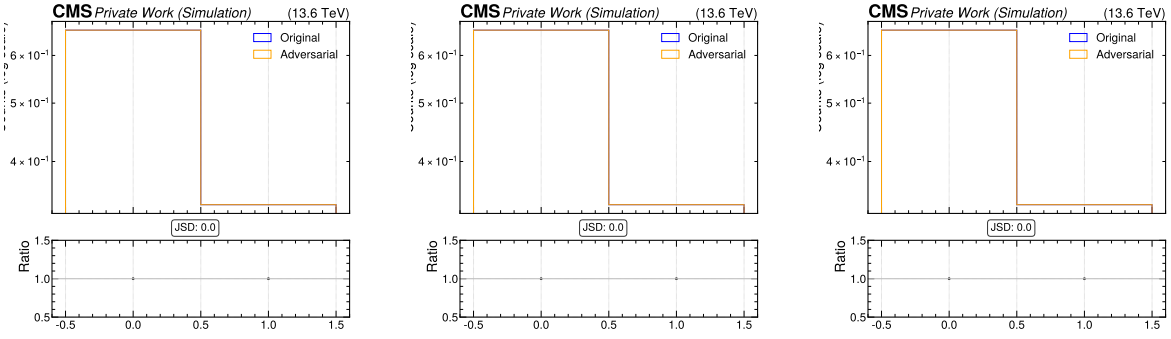
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `Cpfcan_drminsv` for multiple iterations of PIP-PGD tested against nominal inputs.



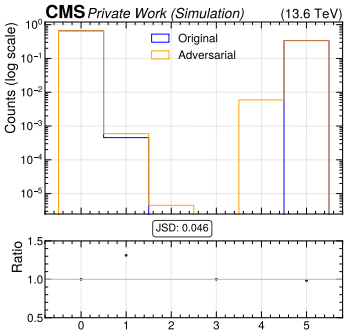
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `Cpfcan_ptrel` for multiple iterations of PIP-PGD tested against nominal inputs.

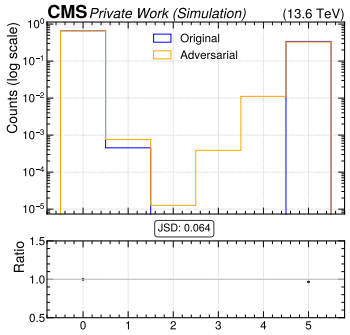


Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

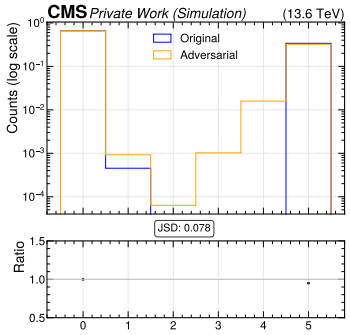
Histogram of `Cpfcan_puppiw` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

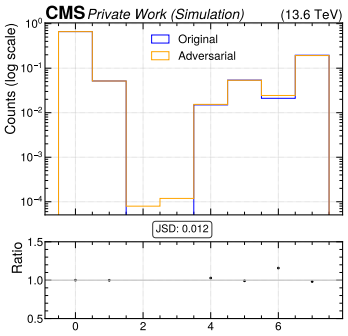


Input similarity for PIP-PGD(2).

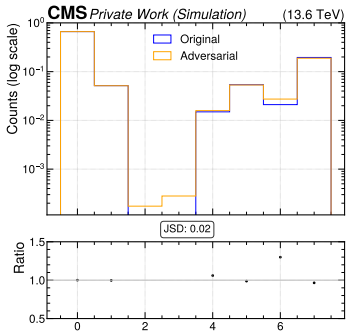


Input similarity for PIP-PGD(3).

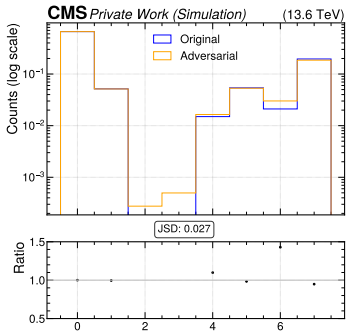
Histogram of `Cpfcan_quality` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

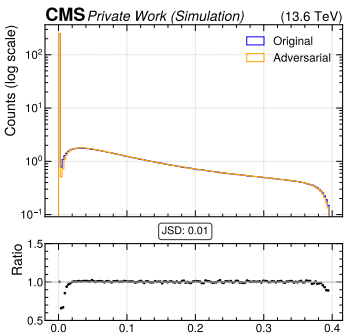


Input similarity for PIP-PGD(2).

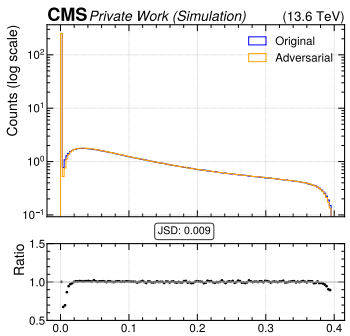


Input similarity for PIP-PGD(3).

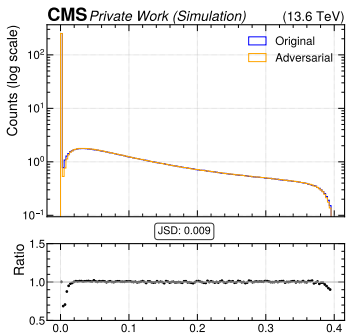
Histogram of `Cpfcan_VTX_ass` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

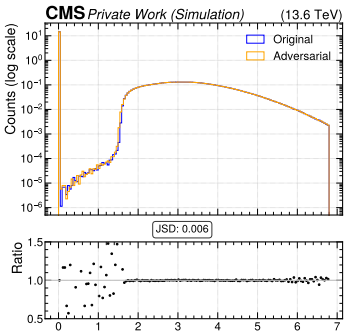


Input similarity for PIP-PGD(2).

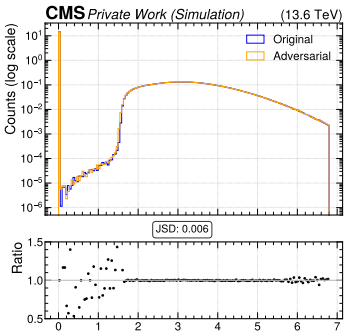


Input similarity for PIP-PGD(3).

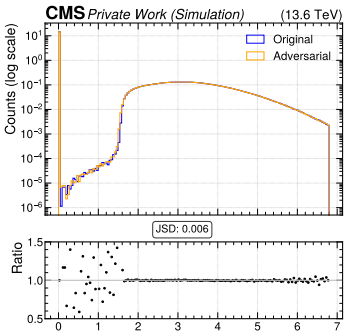
Histogram of `Cpfcan_BtagPf_trackDeltaR` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

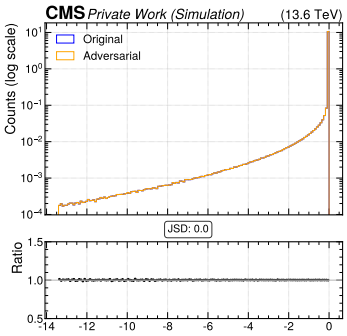


Input similarity for PIP-PGD(2).

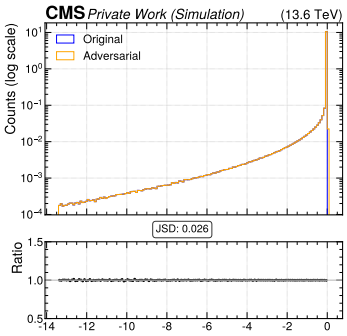


Input similarity for PIP-PGD(3).

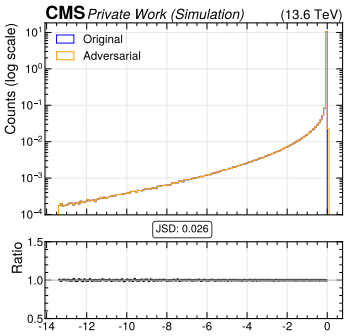
Histogram of `Cpfcan_BtagPf_trackEtaRel` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

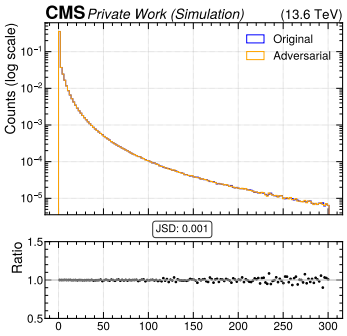


Input similarity for PIP-PGD(2).

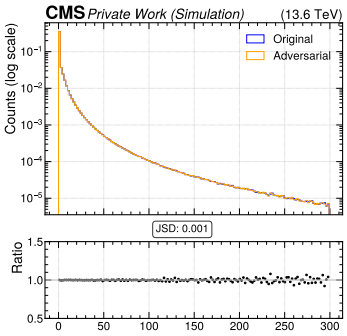


Input similarity for PIP-PGD(3).

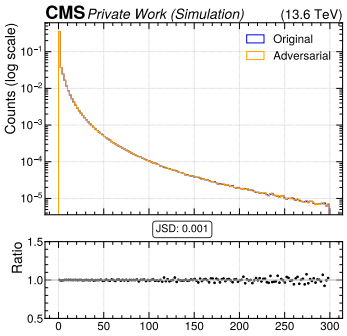
Histogram of `Cpfcan_BtagPf_trackJetDistVal` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

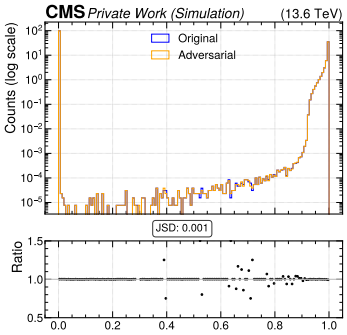


Input similarity for PIP-PGD(2).

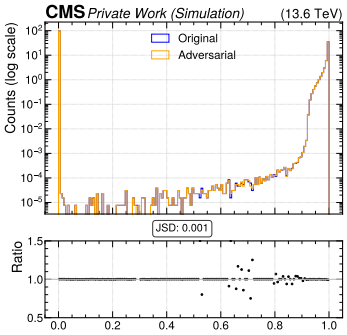


Input similarity for PIP-PGD(3).

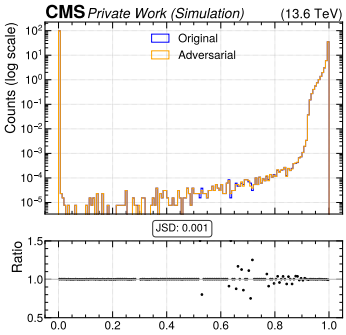
Histogram of `Cpfcan_BtagPf_trackPPar` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

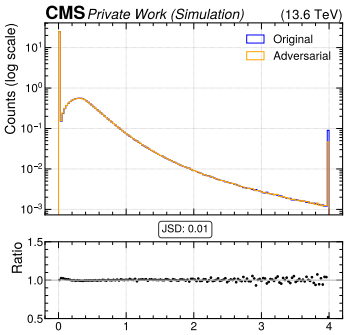


Input similarity for PIP-PGD(2).

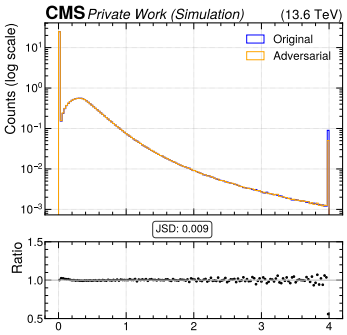


Input similarity for PIP-PGD(3).

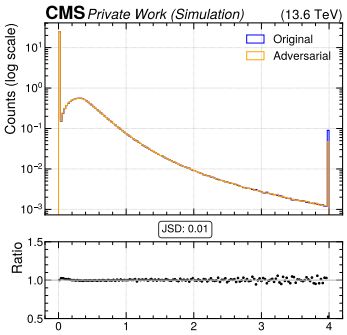
Histogram of `Cpfcan_BtagPf_trackPParRatio` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

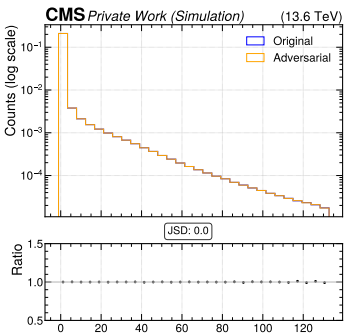


Input similarity for PIP-PGD(2).

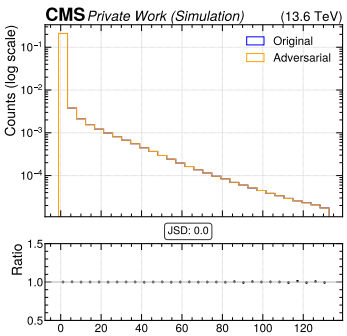


Input similarity for PIP-PGD(3).

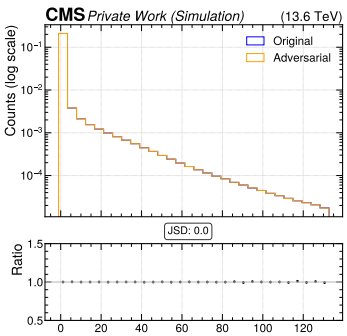
Histogram of `Cpfcan_BtagPf_trackPtRel` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

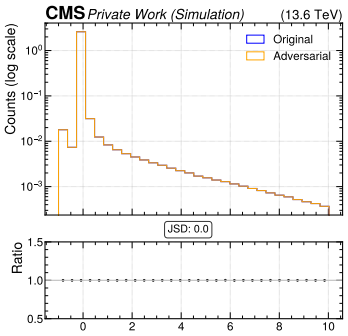


Input similarity for PIP-PGD(2).

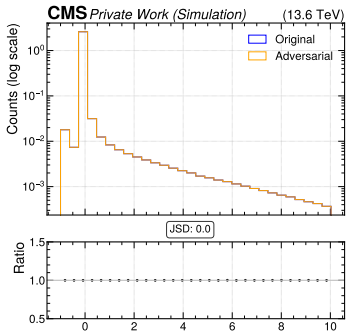


Input similarity for PIP-PGD(3).

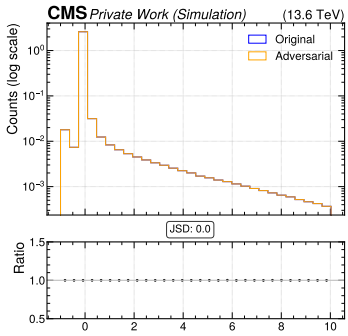
Histogram of `Cpfcan_BtagPf_trackSip2dSig` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

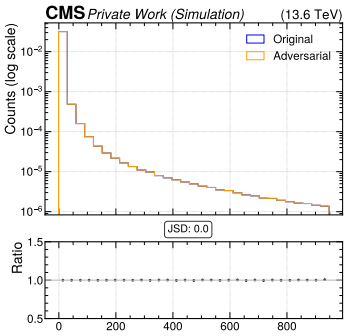


Input similarity for PIP-PGD(2).

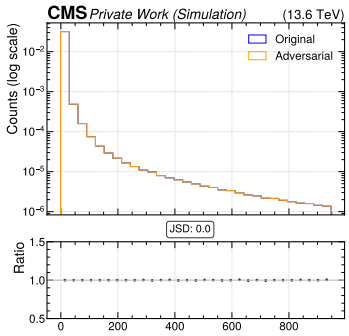


Input similarity for PIP-PGD(3).

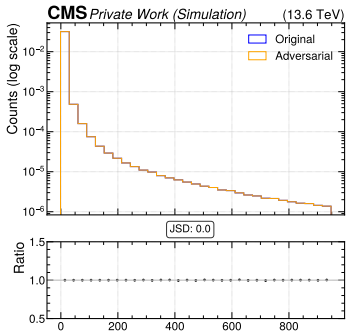
Histogram of `Cpfcan_BtagPf_trackSip2dVal` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

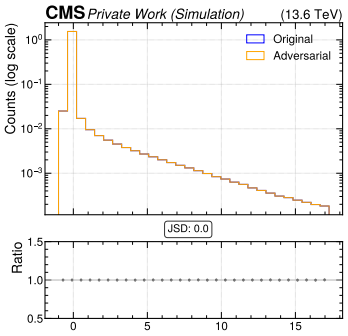


Input similarity for PIP-PGD(2).

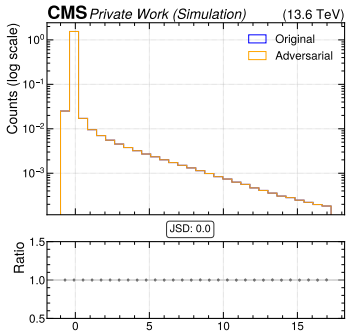


Input similarity for PIP-PGD(3).

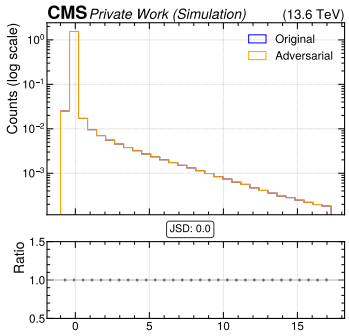
Histogram of `Cpfcan_BtagPf_trackSip3dSig` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).



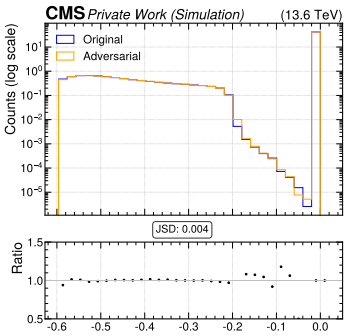
Input similarity for PIP-PGD(2).



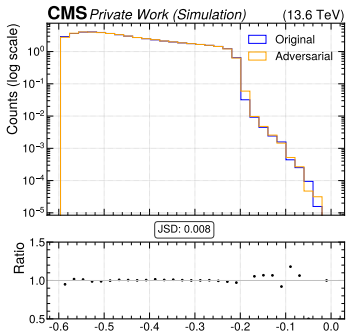
Input similarity for PIP-PGD(3).

Histogram of `Cpfcan_BtagPf_trackSip3dVal` for multiple iterations of PIP-PGD tested against nominal inputs.

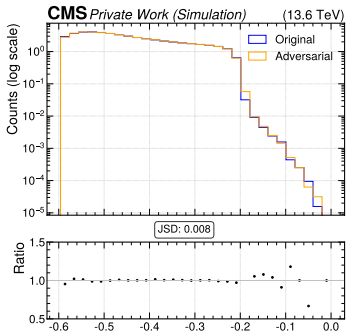
NPF Features



Input similarity for PIP-PGD(1).

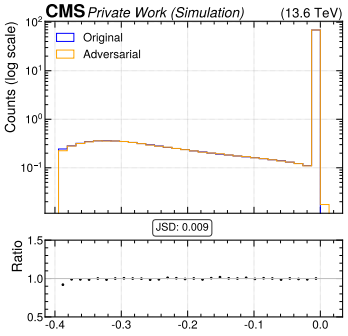


Input similarity for PIP-PGD(2).

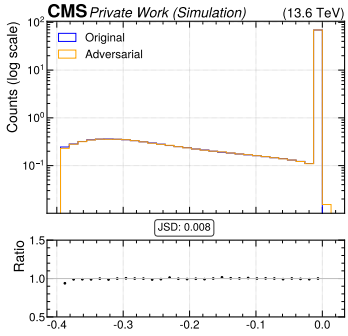


Input similarity for PIP-PGD(3).

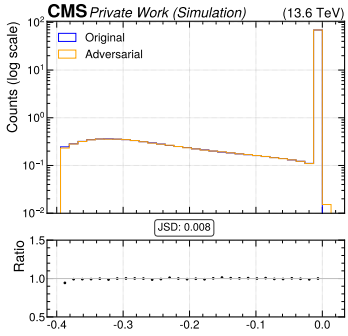
Histogram of `Npfcan_deltaR` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

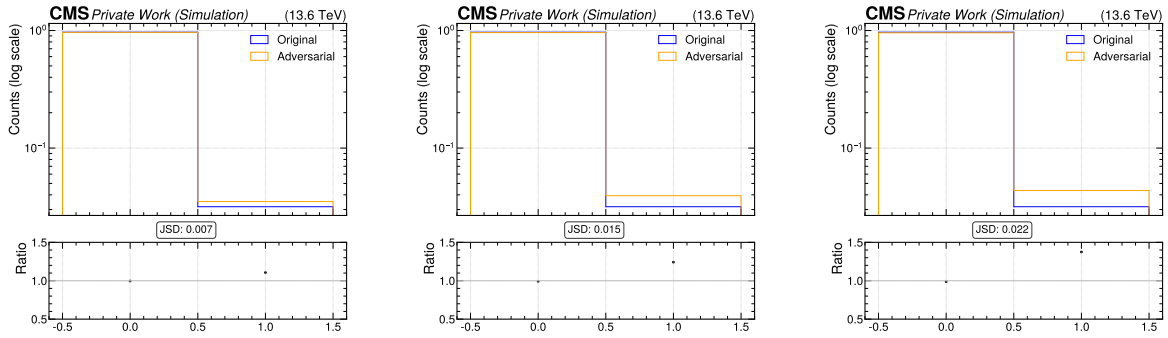


Input similarity for PIP-PGD(2).



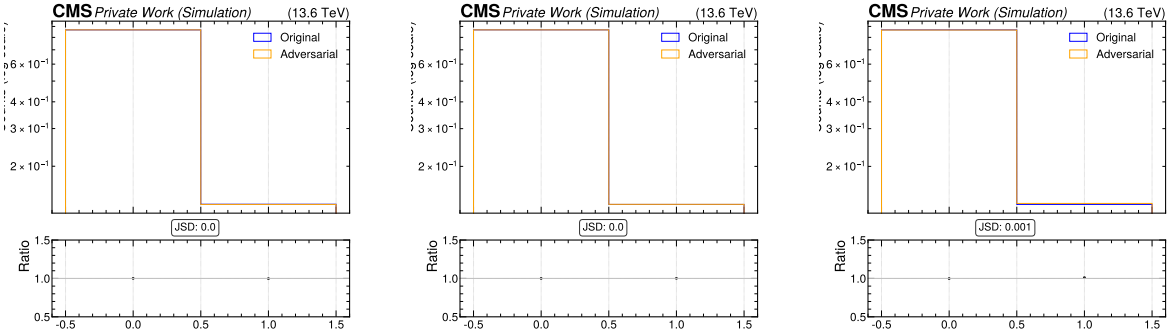
Input similarity for PIP-PGD(3).

Histogram of `Npfcan_drminsv` for multiple iterations of PIP-PGD tested against nominal inputs.



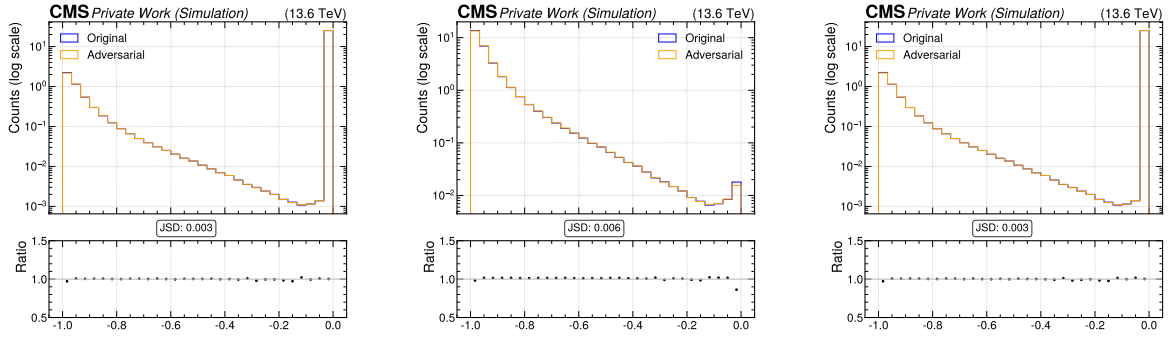
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `Npfcan_HadFrac` for multiple iterations of PIP-PGD tested against nominal inputs.



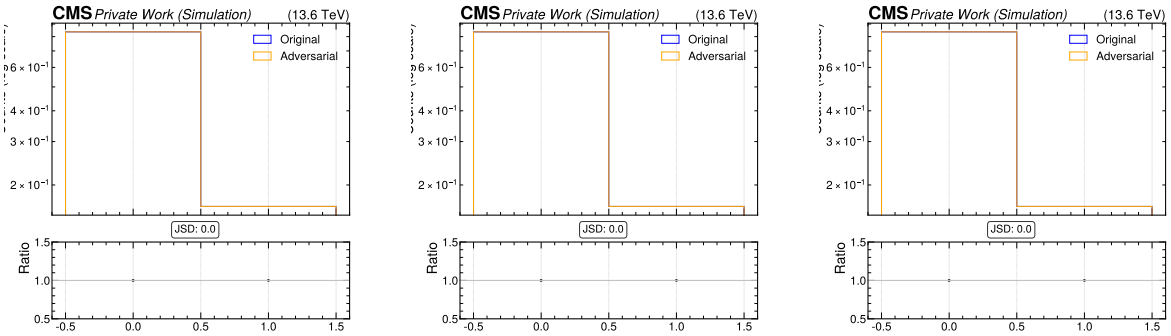
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `Npfcan_isGamma` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

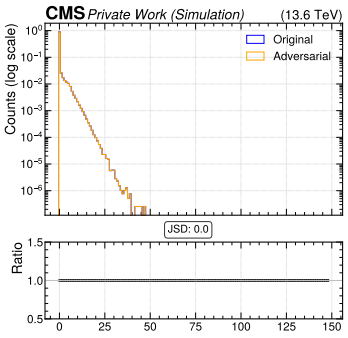
Histogram of `Npfcan_ptrel` for multiple iterations of PIP-PGD tested against nominal inputs.



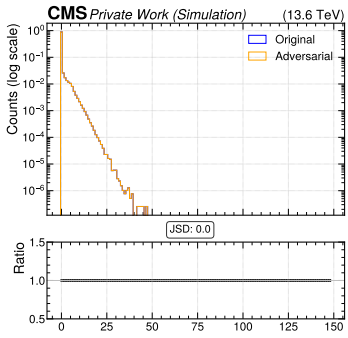
Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `Npfcan_puppiw` for multiple iterations of PIP-PGD tested against nominal inputs.

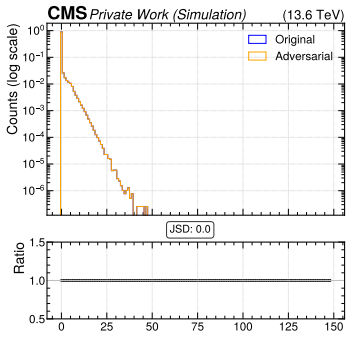
SV Features



Input similarity for PIP-PGD(1).

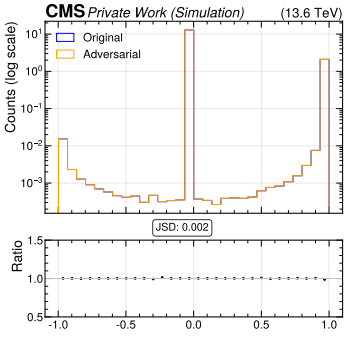


Input similarity for PIP-PGD(2).

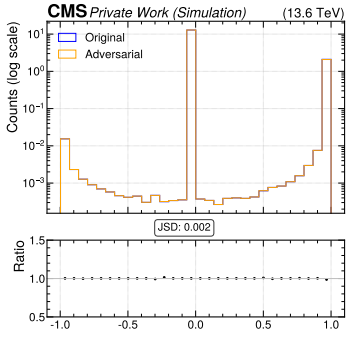


Input similarity for PIP-PGD(3).

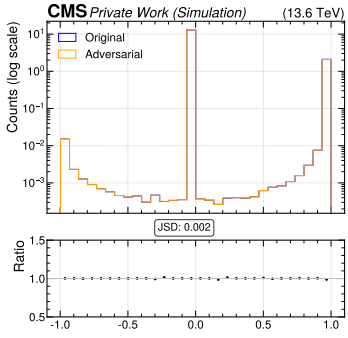
Histogram of `sv_chi2` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

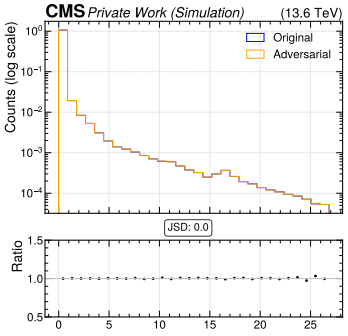


Input similarity for PIP-PGD(2).

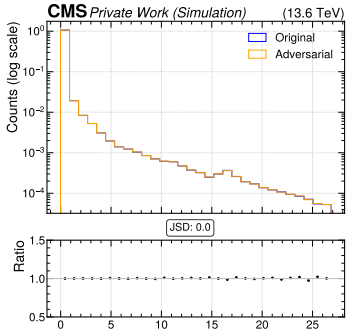


Input similarity for PIP-PGD(3).

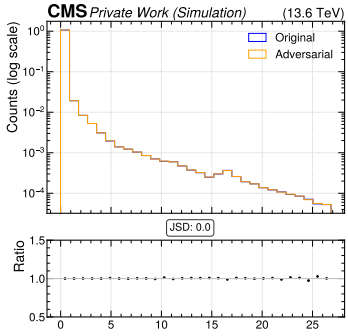
Histogram of `sv_costhetasvpv` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

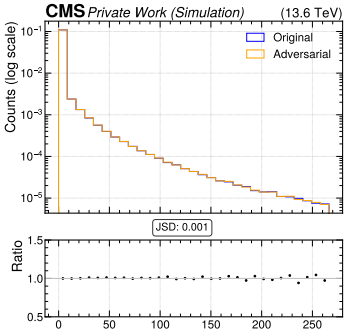


Input similarity for PIP-PGD(2).

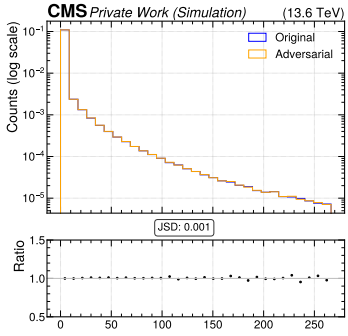


Input similarity for PIP-PGD(3).

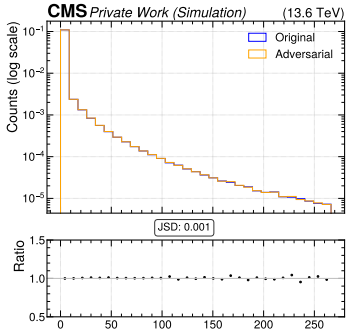
Histogram of `sv_d3d` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

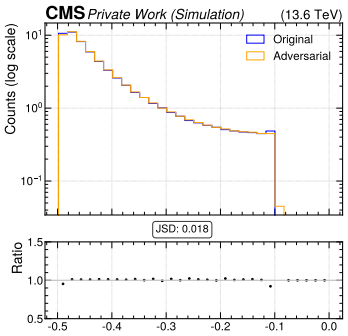


Input similarity for PIP-PGD(2).

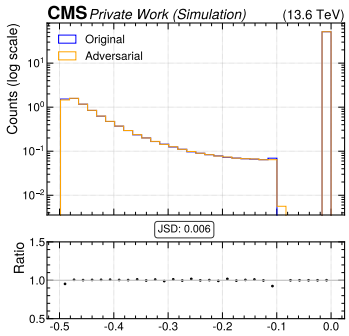


Input similarity for PIP-PGD(3).

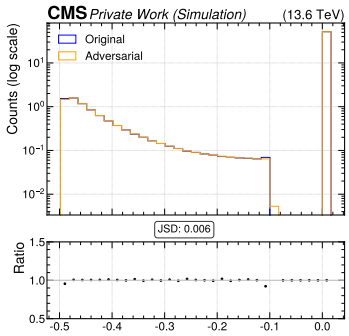
Histogram of `sv_d3dsig` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

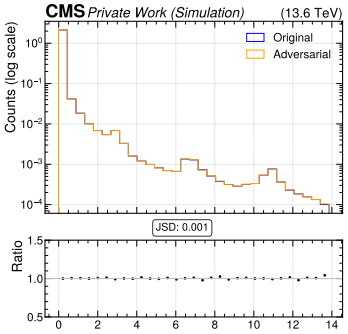


Input similarity for PIP-PGD(2).

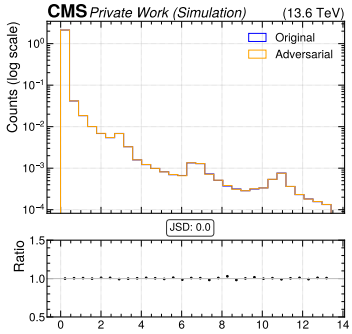


Input similarity for PIP-PGD(3).

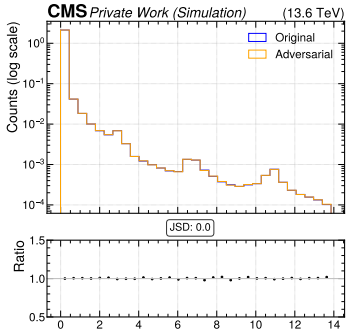
Histogram of `sv_deltaR` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

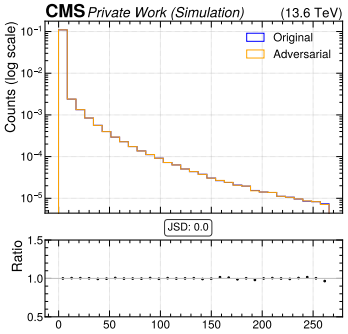


Input similarity for PIP-PGD(2).

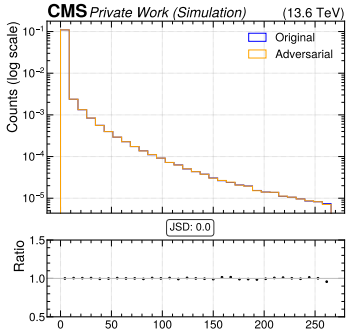


Input similarity for PIP-PGD(3).

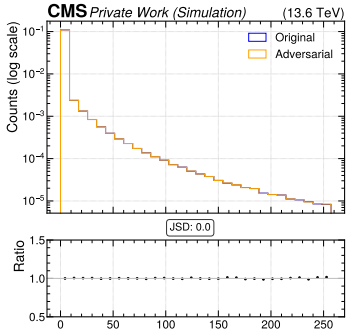
Histogram of `sv_dxy` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

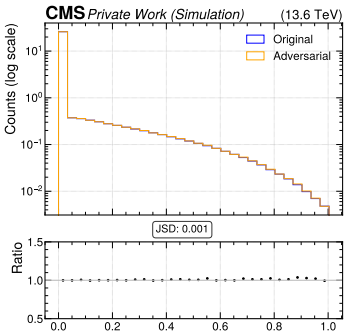


Input similarity for PIP-PGD(2).

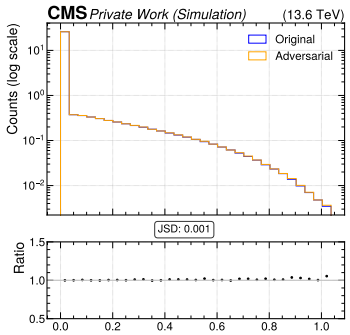


Input similarity for PIP-PGD(3).

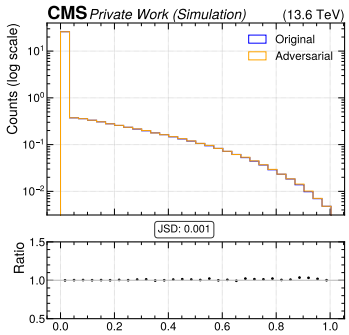
Histogram of `sv_dxysig` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

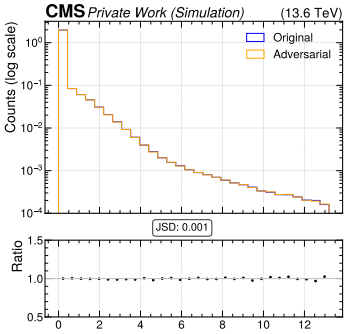


Input similarity for PIP-PGD(2).

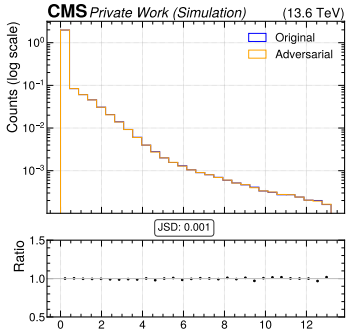


Input similarity for PIP-PGD(3).

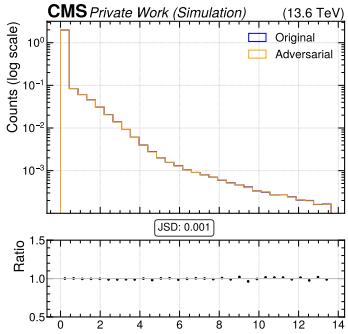
Histogram of `sv_enratio` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

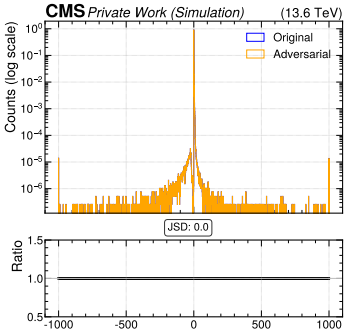


Input similarity for PIP-PGD(2).

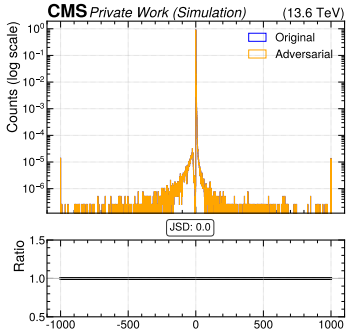


Input similarity for PIP-PGD(3).

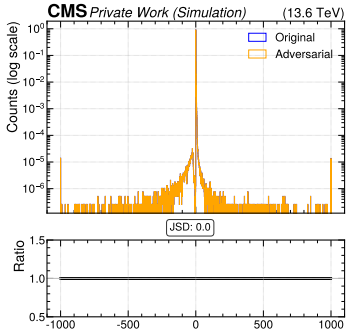
Histogram of `sv_mass` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

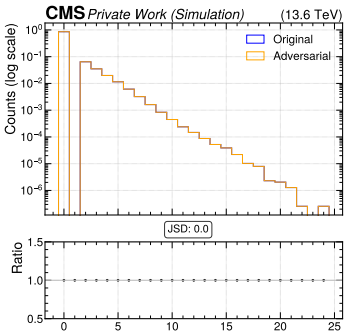


Input similarity for PIP-PGD(2).

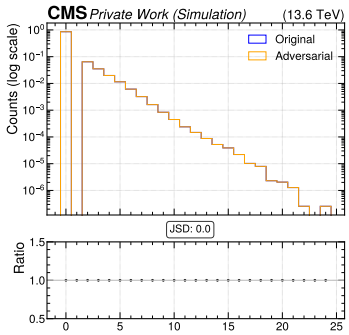


Input similarity for PIP-PGD(3).

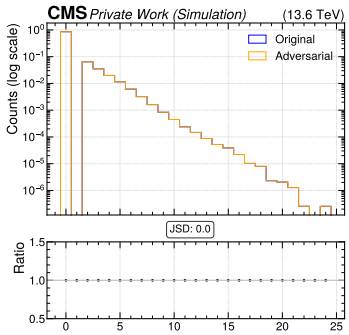
Histogram of `sv_normchi2` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1).

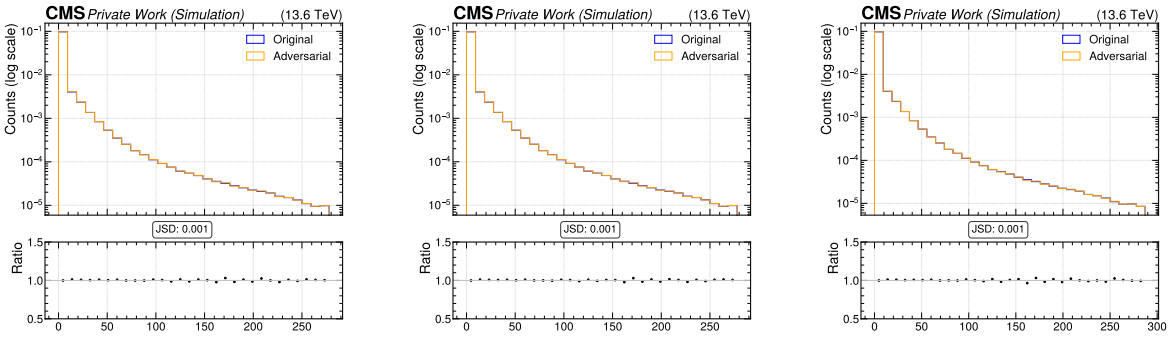


Input similarity for PIP-PGD(2).



Input similarity for PIP-PGD(3).

Histogram of `sv_ntracks` for multiple iterations of PIP-PGD tested against nominal inputs.



Input similarity for PIP-PGD(1). Input similarity for PIP-PGD(2). Input similarity for PIP-PGD(3).

Histogram of `sv_pt` for multiple iterations of PIP-PGD tested against nominal inputs.