ETP-KA/2025-14

Karlsruher Institut für Technologie

Efficient Integration of GPUs in Particle Physics and Transformer-Based Tau Lepton Identification

Zur Erlangung des akademischen Grades eines

DOKTORS DER NATURWISSENSCHAFTEN
(Dr. rer. nat.)

von der KIT-Fakultät für Physik des
Karlsruher Instituts für Technologie (KIT)
genehmigte

DISSERTATION

von

M.Sc. Tim Aike Voigtländer
aus Baden-Baden

Tag der mündlichen Prüfung: 12.12.2025

Referent: Prof. Dr. Günter Quast
Korreferent: Prof. Dr. Achim Streit

Disclaimer

Research in High-Energy Physics (HEP), such as the work presented in this thesis, is a collaborative effort. The Large Hadron Collider (LHC), which provided the proton-proton collisions used in this analysis, was built and is operated by Organisation européenne pour la recherche nucléaire (CERN), with contributions from the LHC Operations Group and the worldwide LHC experiments, including A Toroidal LHC Apperatus (ATLAS), Compact Muon Solenoid (CMS), A Large Ion Collider Experiment (ALICE), and Large Hadron Collider Beauty (LHCb). The CMS detector was constructed, and is operated, maintained, and continuously improved by the CMS Collaboration. This includes the development and maintenance of the extensive computing infrastructure used for data storage and processing of centrally produced simulated samples, as well as the common software frameworks employed in analyses. I have been a member of the CMS collaboration since 2020 and have personally performed all the studies presented in this thesis.

Artificial Intelligence (AI) tools were employed during the making of this thesis to support the grammar and style of both text and code. Grammarly[1] was used throughout the thesis for spelling and grammar assistance, as well as for paraphrasing select sentences to achieve a clearer, more concise flow. I manually surveyed all proposed changes before approval.

Furthermore, ChatGPT[2], Gemini[3], and Sonnet[4] were utilized in Python code creation, specifically to aid in plotting aggregated results. I supervised and tested proposed changes to ensure factual correctness. The same models were queried to assist in research and provide general stylistic feedback. Generative AI was explicitly not utilized to derive or evaluate any of the results presented in this thesis.

[1] AI-powered writing and grammar assistant, https://gemini.google.com/ (Access Date: 2025-10-28)

[2] OpenAI's conversational AI model, https://app.grammarly.com/ (Access Date: 2025-10-28)

[3] Google's multimodal AI model, https://gemini.google.com/ (Access Date: 2025-10-28)

[4] Anthropic's coding AI model, https://www.anthropic.com/claude/sonnet (Access Date: 2025-10-28)

Contents

CHAPTER 1

Introduction

The field of High-Energy Physics (HEP) distinguishes itself from other natural sciences by the sheer scale of its experiments and the breadth of international cooperation involved. At the center of this global research endeavor stands Organisation européenne pour la recherche nucléaire (CERN), a unique institution that enables scientific, engineering, and computational innovation. The Large Hadron Collider (LHC) [1], located at this melting pot of ideas in the center of Europe, is the flagship accelerator of CERN, and has contributed invaluably to establishing the Standard Model (SM) as the most successful theory of fundamental particles and their interactions. With the discovery of the Higgs boson in 2012 [2, 3], the SM was completed after nearly five decades of experimental pursuit.

Since then, the focus of LHC physics has shifted toward exploring physics beyond the SM: supersymmetries, dark-matter candidates, and the deeper connection between gravity and other fundamental forces in a unified framework [4]. To this end, the LHC will enter its high-luminosity phase in 2030 [5], offering the opportunity for novel research but also the challenges posed by a data rate five times the original specifications.

The LHC is often referred to as *humanity's largest experiment*, though even that description can understate the true scale of its operation. While the 27-kilometer superconducting accelerator ring and the tens of thousands of tonnes of intricate detectors, such as the Compact Muon Solenoid (CMS) detector [6], are undoubtedly impressive, they represent only one side of the collaboration. The other side of CERN becomes apparent when one considers what it takes to process the immense stream of detector data into physically meaningful results.

Right after the raw data of a collision is collected by one of the detectors, high-throughput trigger systems filter through the data to bring the initial 40 MHz data rate down to 1 kHz [7]. Furthermore, simulated event data is required to test the various theoretical models against the measured event data. Afterward, hundreds of reconstruction algorithms, including heavyweight machine learning methods, compute thousands of higher-level features for each real and simulated collision. Even these higher-level features are just the starting point of most studies, and typical HEP analyses require further, often computationally intensive, steps to aggregate the reconstructed event data into physically meaningful results.

Each step on the path of a HEP analysis is computationally intensive, precisely because the number of involved collisions, necessary for acceptable statistical precision, scales into the trillions. With the High-Luminosity Large Hadron Collider (HL-LHC) on the horizon, the total number of recorded collision events will rise by another order of magnitude. At the same time, the reconstruction effort is continuously expanded to integrate additional novel approaches. As a result, HEP faces a fundamental scaling problem. Computing budgets cannot rise at the same rate as the required computing capacity, making improvements in computational efficiency not just a cost-saving measure but a necessity.

One of the primary avenues for improved efficiency is the introduction of heterogeneous hardware, specifically GPUs, which can deliver more computational capacity at the same price as traditional CPUs [8]. This thesis studies the necessity for such hardware in the hands of two key questions: What research has become possible due to emerging GPU technologies, and how impactful is GPU availability to efficiency across the broader HEP computing spectrum?

In the first part of my thesis, I introduce the reconstruction and classification of hadronically decaying tau leptons (τ_h), and demonstrate the performance of an updated TauTransformer (TauT). This Machine Learning (ML)-based transformer architecture is a successor candidate to the current DeepTau classifier. Chapter 2 introduces the necessary physics background of the SM and the reconstruction algorithms involved in τ_h reconstruction in CMS, including Particle Flow (PF) and Hadrons-Plus-Strips (HPS). It also outlines the architectures of both the DeepTau and TauT Neural Network (NN) architectures and their use in classifying reconstructed τ_h. Chapter 3 studies the TauT in detail, and provides a direct comparison to the current τ_h classifier DeepTau, as well as an estimation of the variance of rejection efficiencies derived from the TauT. It concludes with a set of ablation tests quantifying the impact of training data volume, convergence conditions, and limitations on the number of particle constituents.

In the second part of my thesis, I focus on the throughput and energy efficiency of three HEP applications and their performance on GPU resources. Chapter 5 reexamines the TauT architecture with a focus on hardware performance, including a comparison between CPU and GPU resources and a study of multi-GPU scaling. Chapter 6 analyzes CPU and GPU efficiency at the hands of a lightweight HEP application and proposes three multi-tenant GPU approaches to improve hardware utilization and collective throughput. Chapter 7 extends this study to a real-life High Throughput Compute (HTC) batch system and benchmarks the performance gains using an extended HEP GPU workload. Chapter 8 continues the CPU and GPU efficiency studies with an official MadGraph4GPU benchmark candidate developed by the HEPiX Benchmark Working Group. Finally, Chapter 9 concludes by stating the computing resources involved in the making of this thesis.

CHAPTER 2

Hadronic Tau reconstruction in High Energy Particle Physics

The field of HEP is renowned for the scale of its experiments and the accuracy of its measurements. The LHC [9], also known as the currently biggest scientific experiment on Earth, produces vast quantities of data on which physicists worldwide rely to test fundamental theories. One of the most crucial steps between data collection and physics analysis is the event reconstruction: the process of piecing together what happened in the millions of collisions that occur every second [10].

A central step in reconstruction is the identification of physics objects that can then be analyzed to probe the underlying laws of nature. This thesis focuses on the reconstruction process of tau leptons, specifically their hadronic decays and the methods used to identify them within the context of the CMS experiment. My work builds on extensive prior research and aims to improve background rejection efficiency further while maintaining the same level of signal efficiency.

This chapter introduces the necessary physics and experiment context to motivate this research and outline the reconstruction pipeline as implemented in the CMS collaboration. I begin with a brief overview of the SM and the role of the tau lepton in it, followed by the LHC and the CMS detector that captures our physics data. Finally, the PF algorithm [11], the HPS [12] algorithm, and the default DeepTau [13] classification NN are introduced. Advances beyond this baseline are presented in Chapter 3. Natural units are used throughout this thesis, setting both the speed of light c and Planck's constant $\hbar$ to one.

2.1 The Standard Model of Particle Physics

Half a century has passed since the term SM was coined [14], and by now it has established itself as one of the most successful theories in all of physics. It predicted multiple novel particles, all of which were found through various experiments, with the last one, the Higgs boson, being found at the LHC in 2012 [2, 3]. The theory has since stood the test of time and remains the most successful description of particle physics, although some phenomena, such as dark matter, are not yet fully covered by it [15]. Excellent descriptions of the SM can be found in literature [16–18], so I will only focus on the topics that directly relate to the physics topic of this thesis: the reconstruction of tau leptons in the context of CMS.

The members of the SM *Particle Zoo*, shown in Figure 2.1, can be sorted into two groups of particles: fermions that carry a half-integer spin and bosons that carry whole-integer spin. The fermions can then be further divided, based on their interactions, into quarks and leptons, each of which consists of three generations, with two members per generation. In particle physics, an interaction is defined as the exchange of forces mediated by one or more bosons, as described below. While quarks participate in all types of interactions, leptons are not influenced by the strong interaction, and only charged leptons interact electromagnetically. Across both fermion groups, the three generations differ mainly in their masses and flavor mixing [19, 20], though the composition of the individual generations is much different. For quarks, each generation consists of an up-type quark with an electric charge of $+\frac{2}{3}\,e$ and a down-type quark with an electric charge of $-\frac{1}{3}\,e$ [21]. Starting from the generation with the lowest mass, the quark pairs are up and down, charm and strange, and top and bottom. The lepton generations consist of the charged leptons, electrons, muons, and taus, and their corresponding neutral neutrinos, which have much lower masses.

As for the bosons, each fundamental force is carried by at least one gauge boson, though the scalar Higgs boson is not directly associated with any force. Gluons mediate the strong force associated with the color charge that is carried only by quarks. While photons γ are often directly associated with Electro-Magnetic (EM) interaction and both $\mathrm{W}^{\pm}$ and Z bosons with the weak interaction, the electroweak (Electroweak (EWK)) unification shows that all three arise as different manifestations of a single $\mathrm{SU}(2)_L{\times}\mathrm{U}(1)_Y$ symmetry. Neither the tightly confined gluons nor the wide-ranging photons carry mass, though all other bosons do [21]. Among all bosons, only the Higgs boson does not carry spin 1, but has spin 0, making it a scalar particle. In addition to the outlined particles, each of the fermions also exists as an antiparticle with inverted charges. Similarly, the $\mathrm{W}^{\pm}$ bosons are each other's antiparticle, though the remaining neutral bosons are their own antiparticles instead. Combined, the SM is formulated around a $\mathrm{SU}(3)_C{\times}\mathrm{SU}(2)_L{\times}\mathrm{U}(1)_Y$ symmetry group, with the $\mathrm{SU}(3)_C$ describing the strong interaction and the $\mathrm{SU}(2)_L{\times}\mathrm{U}(1)_Y$ group describing the EWK interaction. Finally, the Higgs mechanism provides mass to the W and Z bosons, as well as to all fundamental fermions via the Yukawa coupling, completing the SM framework [23–26].

2.1.1 Strong Interaction

The strong interaction of the SM is described by Quantum Chromodynamics (QCD) [18, 27], a non-Abelian gauge theory based on a $\mathrm{SU}(3)_C$ symmetry. The fundamental charges of this symmetry are referred to as *color*, carried by quarks and the gluon gauge bosons. Unlike photons, gluons carry color charge, leading to self-interactions and much of the complexity in the QCD field.

The strong force, opposed to the electro-weak force, experiences confinement: individual quarks or gluons cannot be observed on their own. Instead, a sufficiently energetic

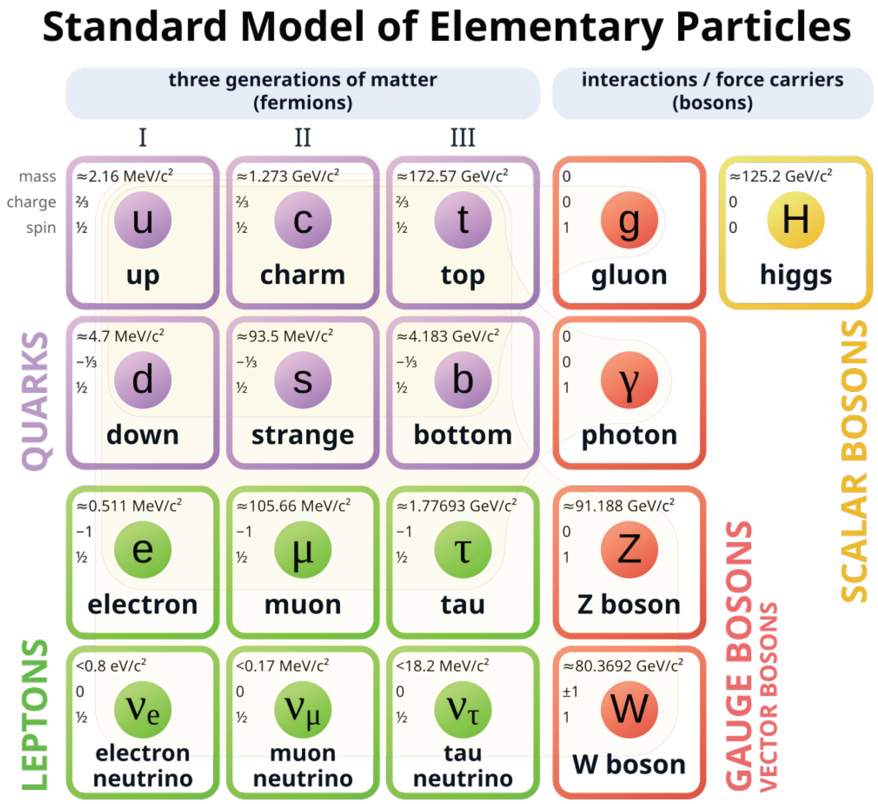


Figure 2.1: Overview of the elementary particles described by the SM [22].

quark or gluon will undergo hadronization, forming color-neutral constructs like baryons or mesons. These constructs usually decay while still inside the detectors, leading to cascading showers that may include resonances, such as pions, kaons, or heavier mesons, before reaching stable hadrons. As a result, a collimated stream (a jet) of hadronic particles is observed in the detectors instead of just a single particle.

The mass of hadrons, which includes both the constituent quark masses and the binding energy, limits the available hadronic activity. The lightest hadrons are the π^0 with masses of $\approx 134.98\,\mathrm{MeV}$ and the lightest charged hadrons are the $\pi^\pm$ with masses of $\approx 139.57\,\mathrm{MeV}$ [21]. Because of these mass thresholds, only the tau among the charged leptons has enough mass to decay both hadronically and leptonically via a $\mathrm{W}^\pm$ boson.

2.1.2 Electroweak Interaction

Initially, the EWK interaction was regarded as two distinct fundamental forces. On one side, EM interactions were associated with the photon field A_μ and associated with a U(1) gauge symmetry, under which electric charge q is conserved [18, 28]. On the other side, weak interactions were known for their special behavior regarding chirality and both charged and neutral currents. The (V-A) theory [29, 30] proposed that the weak force would only couple to left-chiral fermions and right-chiral anti-fermions. The theory was motivated by parity-violating observations in beta decays [31], which implied that the decay products of spin-1/2 particles are preferentially emitted relative to the particle's spin. Later, the weak force was formulated as a $\mathrm{SU}(2)_L$ gauge theory, introducing a conserved weak isospin I. Only left-chiral components of the fermion doublets ψ_i are affected, while right-chiral components remain. In this theory, three gauge bosons W_μ^i were predicted to mediate the weak interaction, though their physical combinations corresponding to charged ($\mathrm{W}^\pm$) and neutral (Z) interactions were still unknown.

The two forces were finally unified in the *Glashow-Salam-Weinberg* theory [32–34], combining both EM and weak interaction into one term. The theory replaces the U(1) and $\mathrm{SU}(2)_L$ symmetry groups with a combined $\mathrm{SU}(2)_L \times \mathrm{U}(1)_Y$ group, introducing the weak hypercharge Y in the process. This shift introduced an additional neutral gauge boson B_μ^0, allowing for the definition of the four EWK gauge bosons as we know them today. With the coupling constants of the weak isospin g, the coupling constant of the weak hypercharge g', and the Weinberg-angle $\theta_W = \arctan \frac{g'}{g}$, the EWK gauge bosons could be defined as a linear combination:

$$\begin{pmatrix} A_\mu \\ Z_\mu \end{pmatrix} = \begin{pmatrix} \cos\theta_W & \sin\theta_W \\ -\sin\theta_W & \cos\theta_W \end{pmatrix} \begin{pmatrix} B_\mu^0 \\ W_\mu^0 \end{pmatrix}, \tag{2.1}$$

$$W_\mu^\pm = \frac{1}{\sqrt{2}} \left(W_\mu^1 \mp W_\mu^2 \right) \tag{2.2}$$

The quantum number associated with electric charge q can then be expressed in terms of the weak isospin and hypercharge quantum numbers as $q = I + Y/2$.

With the addition of the spontaneous symmetry breaking facilitated by the Higgs mechanism [23–26], the three bosons associated with the weak force, $\mathrm{W}^\pm$ and Z, gain a non-zero mass ($m_W \approx 80.37\,\mathrm{GeV}$ and $m_Z \approx 91.19\,\mathrm{GeV}$) [21]. At the same time, the photon remains massless.

2.1.3 The Tau lepton

The tau lepton is the third-generation charged lepton of the SM, and with a mass of $m_\tau \approx 1.777\,\mathrm{GeV}$ [21], about 17 times the muon mass and 3500 times the electron mass. This significant mass results in multiple distinct properties that are primarily exclusive to the tau: significantly strong coupling to the Higgs boson, the possibility to decay hadronically, and a short lifetime.

Firstly, in the SM, fermion masses arise through Yukawa interactions with the Higgs field, and correspondingly, a higher mass also implies a stronger coupling to the Higgs boson. Due to the linear proportionality of the Yukawa coupling strength to the fermion mass, the Higgs boson couplings to the lighter leptons are very small. The branching ratio for the decay into muons is approximately 0.02%, while the decay into electrons is negligible [21]. As the heaviest of all known leptons, the decay into taus ($H \to \tau^+\tau^-$) is the only one that accounts for a significant portion of the Higgs boson branching ratio at 6.27% [21]. The decay of the Higgs boson into taus is considered an essential channel

Table 2.1: Leptonic and hadronic DMs of the tau lepton, including branching fractions $\mathcal{B}$ [21] and the hadronic tau DMI associated with the respective DMs. Charged pions and kaons are denoted as $\mathrm{h}^{\pm}$ and branching ratios are invariant under charge conjugation.

Decay Mode	$\mathcal{B}$ [%]	DMI
Leptonic	35.2	—
$\tau^{\pm} \to e^{\pm}\overline{\nu}_e\nu_\tau$	17.8	—
$\tau^{\pm} \to \mu^{\pm}\overline{\nu}_\mu\nu_\tau$	17.4	—
Hadronic	64.8	—
$\tau^{\pm} \to \mathrm{h}^{\pm}\nu_\tau$	11.5	0
$\tau^{\pm} \to \mathrm{h}^{\pm}\pi^0\nu_\tau$	25.9	1
$\tau^{\pm} \to \mathrm{h}^{\pm}\pi^0\pi^0\nu_\tau$	9.5	2
$\tau^{\pm} \to \mathrm{h}^{\pm}\mathrm{h}^{\mp}\mathrm{h}^{\pm}\nu_\tau$	9.8	10
$\tau^{\pm} \to \mathrm{h}^{\pm}\mathrm{h}^{\mp}\mathrm{h}^{\pm}\pi^0\nu_\tau$	4.8	11
Other	3.3	—

for precision Higgs studies, as it provides a direct probe of Higgs boson couplings to charged leptons. In addition, while the dominant Higgs boson decay channel $H \to \bar{b}b$ has to contend with an extensive jet background [35], the decay into taus results in much simpler leptonic and hadronic structures.

Secondly, the decay of a tau is always mediated by a $\mathrm{W}^{\pm}$ boson. Still, unlike lighter leptons, taus are sufficiently massive for the $\mathrm{W}^{\pm}$ to decay not just into lepton-neutrino pairs, but also pairs of up and down-type quarks. Kinematically, the tau can decay into mixtures of up, down, and strange quarks, though in practice, decays involving strange quarks are much rarer than those involving down quarks [21]. Table 2.1 lists the most prominent Decay Modes (DMs) for both the leptonic and hadronic decays, including the Decay Mode Integer (DMI) associated with them. The DMI summarizes both the number of charged pions and kaons (prongs) N_{chr}, and the number of π^0 hadrons N_{π^0} involved in the DM:

$$\mathrm{DMI} = 5 \times (N_{\mathrm{chr}} - 1) + N_{\pi^0} \tag{2.3}$$

Notably, the complete hadronic decay does not occur directly from the emitted $\mathrm{W}^{\pm}$ of the decaying tau, but instead proceeds dominantly via intermediate resonances such as ρ and a_1 mesons [21]. In addition, even for the hadronic decays of the tau, the presence of at least one ν_τ makes full tau momentum reconstruction difficult, as CMS cannot detect the neutrino directly. While approximately 35% of tau decays involve lighter leptons, the near 65% chance that taus decay hadronically underscores the importance of hadronic tau reconstruction.

Finally, the large tau mass results in a short lifetime compared to both the stable electron and the relatively long-lived muon. The availability of numerous hadronic decay channels and a wider range of possible final-state particle configurations leads to a mean lifetime of $\tau_\tau = 290.3\,\mathrm{fs}$, and a decay length of $s_\tau \approx 87\,\mu\mathrm{m}$ [21]. Even for highly boosted taus in collisions, as with the LHC, this translates to a flight distance on the order of millimeters, too short to be directly trackable in any current detector. Instead, indirect reconstruction methods are employed: the decay products of the tau lepton are used to determine a secondary vertex that is slightly displaced from the primary collision vertex [12].

2.2 The CMS collaboration

The CMS experiment [6] is one of the two general-purpose experiments at the LHC [9]. The collaboration encompasses the detector itself, the worldwide computing infrastructure necessary to process its data, and more than 4000 scientists, engineers, and technicians responsible for the design, operation, and analysis of the data [36]. As one of the leading experiments in HEP, CMS records, manages, and analyzes petabytes of particle physics data each year. These efforts have enabled landmark achievements, including the discovery of the Higgs boson [2, 3], precision measurements of SM processes [37], and extensive searches for new physics beyond the SM [38].

2.2.1 The Large Hadron Collider

The LHC serves as the world's largest proton and heavy-ion ring accelerator and is seen by many as humanity's largest experimental setup [9]. It is based at the CERN near Geneva, Switzerland, reusing the 26.6 km long circular tunnels of the former Large Electron-Positron Collider [39]. It consists of two anti-parallel beam pipes that cross over at four points, guided by a complex array of multipole magnets and accelerated by an array of radio-frequency cavities. Each beam reaches proton energies close to 7 TeV for a combined center of mass energy of 13 TeV for Run 2 (2015-2018) of the accelerator and 13.6 TeV for Run 3 (2022-present) [40].

There are four main HEP experiments located at the four crossover points of the beam pipes: CMS, A Toroidal LHC Apperatus (ATLAS) [41], Large Hadron Collider Beauty (LHCb) [42], and A Large Ion Collider Experiment (ALICE) [43]. Of these, CMS and ATLAS can be considered the most general-purpose detectors, while both ALICE and LHCb focus on more exotic parts of the HEP spectrum. At each crossover point, opposite-running bunches of accelerated particles can interact and cause collisions.

At these interaction points, protons or ions collide at unprecedented energies, producing events with all particles predicted by the SM and resulting in highly complex final states. In addition to reaching the highest center-of-mass energy of any HEP accelerator, the LHC also does so at an impressive luminosity. Especially rare interactions, such as the decay of a Higgs boson into muons [21], or potential processes beyond the SM, necessitate high event rates to gather sufficient statistical data.

The number of events N in which a respective process is observed is connected to both the cross section of that process σ and the integrated luminosity L_{int}. The integrated luminosity is given by the integral over the instantaneous luminosity $\mathcal{L}$ and describes the collision rate density [21]:

$$N = \sigma \cdot L_{\text{int}} \quad \text{with} \quad L_{\text{int}} = \int \mathcal{L}(t) dt \tag{2.4}$$

The colliding bunches of protons consist of $\mathcal{O}(10^{11})$ protons each, and collisions occur at a rate of 40 MHz [10]. With n_1 and n_2 as the number of protons per bunch, a_x and a_y as the overlap between the bunches in x and y direction and f as the crossing frequency, instantaneous luminosity is defined as

$$\mathcal{L} = f \frac{n_1 n_2}{4\pi a_x a_y}. \tag{2.5}$$

During Run 2, the LHC had delivered a total integrated luminosity of 159.8 fb^{-1}, while the ongoing Run 3 exceeded this integrated luminosity in September of 2024 [44] and continues until July 2026.

Looking further ahead, the HL-LHC upgrade of the accelerator [5], planned for 2030, will significantly raise instantaneous luminosity. Through a combination of a more concentrated interaction point, more protons per bunch, and improved beam stability, instantaneous luminosity is set to increase by up to 3.5× compared to Run 3. Over the remaining lifetime of the LHC, this upgrade is expected to yield a total integrated luminosity of 3000-4000 fb^{-1}, enabling unprecedented precision in probing rare processes.

2.2.2 The Compact Muon Solenoid

The CMS detector stands as the central detector of the CMS collaboration at the LHC. Physically, it stands roughly 100 meters underneath the surface near Cessy, France, as the northern-most experiment of the LHC. As one of the two general-purpose detectors of the LHC, it was constructed to record rich and detailed information on particle collisions. It has been described in detail in numerous publications, such as [45, 46], so I focus here on the details relevant to hadronic tau reconstruction and classification.

At a length of 21 m, a height of 15 m, and a total weight of 14000 t, the CMS detector is densely packed, worthy of the *compact* part in its name. The remainder of the detector's name reflects its other defining features: a strong superconducting solenoid magnet producing a 3.8 T magnetic field in its volume, and an extensive muon system. The detector consists of multiple subdetector modules, each focused on a specialized task, such as tracing charged particles or detecting hadronic particles.

The modules are placed concentrically around the crossing point of the beam pipes (also called the interaction point) in a cylindrical setup. This onion-like setup, illustrated in Figure 2.2, is completed with endcap components that extend coverage to large pseudorapidity. Pseudorapidity is part of the coordinate system used by CMS, which is outlined below.

From the innermost to the outermost detector part, there are the tracker, the Electromagnetic Calorimeter (ECAL), the Hadronic Calorimeter (HCAL), the solenoid, and the muon chambers. Each of these is essential for the CMS experiment, and their primary purposes are as follows:

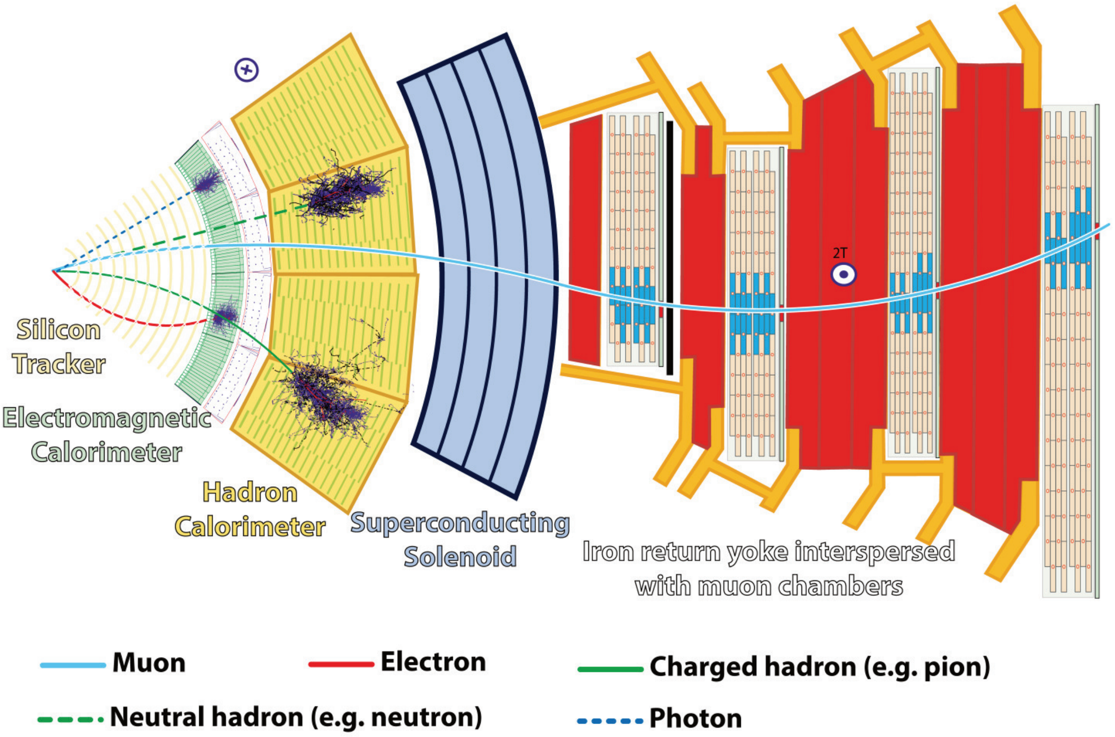

Silicon Tracker
Electromagnetic Calorimeter
Hadron Calorimeter
Superconducting Solenoid
Iron return yoke interspersed with muon chambers
2T
Muon
Electron
Charged hadron (e.g. pion)
Neutral hadron (e.g. neutron)
Photon

Figure 2.2: Slice of the CMS detector illustrating the subsystems and their role in event reconstruction [47]. The particle tracks in the trackers and showers in the calorimeters illustrate the behavior of different particle types in the detector.

- **Solenoid**: generates the magnetic field that bends charged particles' trajectories in the detector.
- **Tracker**: reconstructs the trajectory and momenta of charged particles, curved by the solenoid.
- **ECAL**: measures the energy of electrons and photons, stopping them in the process.
- **HCAL**: measures the energy of hadrons, similar to the ECAL.
- **Muon system**: detects and measures muons with high precision.

While not all systems are directly involved in hadronic tau reconstruction, they all contribute to the broader picture of hadronic tau identification. The tracker enables the reconstruction of 1-pronged and 3-pronged decays, while the ECAL provides precise measurements of neutral pions. The HCAL contributes information on shower shape, aiding discrimination against quark and gluon induced jets (QCD jets), and the muon system offers a powerful veto against muons, reducing misidentification rates in tau selection.

2.2.2.1 CMS coordinates

The coordinate system used throughout CMS is designed to highlight the physical properties of the particles recorded within the detector. Because colliding protons are composite objects, the exact fraction of total momentum carried by the individual quark or gluon constituents is unknown. Therefore, the longitudinal momentum p_z in the z-direction along the beam pipe is both highly boosted and not precisely known for the collision. Furthermore, the transverse momentum $p_\mathrm{T} = \sqrt{p_x^2 + p_y^2}$ of the partonic system is approximately zero before the collision.

Instead of classical Cartesian coordinates, the detector coordinate system is based on the cylindrical and Lorentz invariant rapidity y and pseudorapidity η measures, with E as particle energy, m as mass, and θ as the polar angle:

$$y = \frac{1}{2} \ln \left(\frac{E + p_z}{E - p_z} \right) \tag{2.6}$$

$$\eta = - \ln \left(\tan \frac{\theta}{2} \right) \tag{2.7}$$

For highly relativistic particles with $m \ll p$, rapidity and pseudorapidity converge, $y \approx \eta$.

Together with the azimuthal angle ϕ and the transverse momentum p_T, this coordinate system allows for an accurate description of particle properties. The main advantage of this system is that exact knowledge of the particle's longitudinal properties p_z is not necessary. The distance between two particles is commonly expressed in terms of the $\eta - \phi$ plane:

$$\Delta R = \sqrt{(\eta_1 - \eta_2)^2 + (\phi_1 - \phi_2)^2} \tag{2.8}$$

with η_1 and ϕ_1 denoting the coordinates of the first particle and η_2 and ϕ_2 denoting those of the second. As with rapidity, this distance measure is invariant under boosts along the beam direction and is widely used in clustering and isolation criteria.

2.2.2.2 Tracking

The silicon tracker is the innermost subsystem of CMS, closest to the particle beam pipe and the interaction point. It allows for the reconstruction of particle trajectories and, together with the 3.8 T magnetic field of the CMS solenoid, the determination of particle momenta and electric charge. Beyond reconstructing individual particle trajectories, the tracker provides precise vertexing possibilities, enabling the identification of secondary vertices. This is crucial for short-lived particles such as tau leptons, whose decays can produce displaced tracks.

To achieve the precision required for reconstruction, the tracker employs highly granular, fast, and radiation-hard silicon sensors. The tracker can be separated into two parts: a silicon pixel detector and a silicon strip detector.

The pixel tracking system was replaced between the data-taking periods of 2016 and 2017, enabling the addition of an additional detection layer closer to the interaction point [48]. Since then, the upgraded setup consists of four barrel layers and three endcap disks, with a total of 1856 modules, each equipped with a sensor with 160 ×416 pixels (100 µm×150 µm each). Combined, the pixel detector provides 124 million readout channels with four-layer coverage up to $|\eta| < 2.5$. The pixel detector achieves a typical spatial resolution of 9.5 µm in the transverse plane and 22.2 µm along the beam axis.

The strip detector, wrapped around the pixel detector, adds 10 additional layers of silicon sensors in the barrel region, as well as 12 layers in the forward and backward directions. The primary goal of the strips is to complement the pixel detector and extend the tracker's volume.

The strip modules differ from the pixels in their resolution: the individual stripes have a narrow pitch of 80-180 µm, but use a much coarser long axis of 10-20 cm. As a result, the modules are still susceptible to changes in the azimuthal angle ϕ, but much less so regarding their pseudorapidity η resolution. By combining the results of two modules mounted back-to-back, shifted by an angle of 100 mrad (*stereo modules*), 3D information is obtained for some layers.

The tracker reconstructs secondary vertices and provides fine-grained information on track multiplicity. Both of these play essential roles in distinguishing between 1-prong and 3-prong hadronic tau decays and the more general tau lepton reconstruction effort.

2.2.2.3 Electromagnetic Calorimeter

The primary function of the ECAL is to measure the energies of electrons and photons by developing EM showers, fully absorbing their energy within the detector volume. Central to the ECAL 's operation is the radiation length $X_{0,\mathrm{Pb}}$, which characterizes the mean distance over which an electron loses $1/e$ of its energy through interactions with the material. The subdetector consists of three regions: ECAL Preshower (ES), ECAL Barrel (EB), and ECAL Endcap (EE).

The ES spans the region of $1.653 < |\eta| < 2.6$ and is composed of two layers of lead with a total thickness of $3X_{0,\mathrm{PbWO_4}}$, as well as two layers of silicon strip detectors behind the absorbers. Their purpose is to increase spatial granularity in the forward region, allowing for the distinction between prompt photons and those produced in π^0 decays. This property is particularly relevant to hadronic tau reconstruction, as both 1-pronged and 3-pronged decays may involve neutral pions.

The remainder of the ECAL is constructed from lead tungstate ($PbWO_4$) crystals that act as both absorber and scintillator at once. The EB region ($|\eta| < 1.479$) consists of 61200 tapered crystals, 26 mm×26 mm at the rear face tapering down to 22 mm×22 mm at the front, with a length of 23 cm ($25.8X_0$). The EE region ($1.479 < |\eta| < 3$) is composed of 14648 thicker crystals (30 mm×30 mm tapering down to 28.6 mm×28.6 mm), each 22 cm long ($24.7X_{0,\mathrm{PbWO_4}}$).

The fine granularity and excellent energy resolution of the ECAL allow for precise reconstruction of π^0 in hadronic tau decays and improve the discrimination against the electron background.

2.2.2.4 Hadronic Calorimeter

The primary function of the HCAL is to measure the energies of hadronic particles. In CMS, this subdetector is composed mainly of alternating layers of brass or iron and plastic scintillator. The size of the HCAL components is provided in terms of nuclear interaction length λ_I, the mean distance a hadron can propagate without an inelastic interaction with a nucleus. The HCAL is split into four parts: HCAL Barrel (HB) in the barrel region, HCAL Endcap (HE) in the endcaps, HCAL Forward (HF) in the forward region beyond the HE, and HCAL Outer (HO) located just outside the solenoid.

The HB covers the region $|\eta| < 1.3$ with a thickness of $5.82\lambda_I$ to $10.6\lambda_I$. It is complemented by the HO to account for insufficient hadronic shower containment provided solely by EB and HB. For most of its range $|\eta| < 1.26$, the HO consists of one layer of absorber material ($3\lambda_I$) though directly around the interaction point ($|\eta| \approx 0$) there are two layers. The HE is located in the range of $1.3 < |\eta| < 3$ and provides between $10\lambda_I$ and $11\lambda_I$, depending on pseudorapidity. Finally, the HF is located outside of the main detector, covers the range $2.85 < |\eta| < 5.19$, and is made from steel embedded with quartz fiber.

For hadronic tau reconstruction, the HCAL is mostly complementary, as both charged hadrons and neutral pions are mainly covered by the tracker and ECAL modules. Information on the energy and transverse momentum of the charged hadrons from a tau decay provides a cross-check. However, the main benefit lies in reducing misidentification of jets during classification. The data obtained from the HCAL allow comparisons of shower shape, with hadronic tau decays producing narrower, shallower showers than QCD jets.

2.2.2.5 Muon System

The muon system of the CMS detector is located outside of the solenoid and is partly housed within the return yoke of the magnet. As the name CMS implies, its muon system is capable, enabling precise muon measurements, including momentum, energy, and event triggers.

In the barrel region up to $|\eta| < 1.2$, where a low particle rate is expected, multiple layers of parallel drift tubes are stacked, allowing for a precise three-dimensional tracking. At the two endcaps, covering the region $0.9 < |\eta| < 2.4$, cathode strip chambers are used instead, as they are more robust against the background in the forward region. In addition, fast-triggering resistive plate chambers are installed in both barrel and endcaps to complement the other systems, covering the region $|\eta| < 2.1$.

For hadronic tau reconstruction, the muon system is not directly involved, as hadronic tau decays do not produce muons. Instead, the main benefit lies in the misidentification of muons as tau leptons, as high-precision muon detection allows for a powerful veto and significantly reduces misidentification rates.

2.3 The Hadronic Tau Reconstruction

Before physics research is possible, the information recorded as electrical signals in the different subdetectors of the CMS needs to be aggregated. Depending on the signal origin and the target physics object, different algorithms are employed for reconstruction [12]. This section focuses on the explicit information used in the reconstruction and classification of hadronically decaying taus. I first introduce the initial RECO step that combines detector-level information into basic physics elements. Afterwards, I describe the PF [11] and HPS [12] algorithms that reconstruct the low-level information into more complex objects.

2.3.1 Basic Elements

The process of reconstructing a physics event involves multiple sequential steps that combine detector-level information into increasingly high-level features. The first step in this process is the RECO reconstruction step [49], in which electrical signals are aggregated into basic elements: calorimeter clusters, charged hadron tracks, electron tracks, and muon tracks. While the resulting elements are directly used in the PF reconstruction step, individually reconstructed electrons and muons (RECO electrons/muons) are also commonly used by themselves in analyses.

Calorimeter Cluster The reconstruction of calorimeter clusters is performed independently in multiple subdetectors of CMS in a similar fashion [11]: the EB and HB surrounding the barrel, and the ES, EE, and HE at the two endcaps. Initially, cluster seed calorimeter cells are identified based on energy-deposition thresholds. Then, the seed is grown by adding adjacent cells to the topological cluster if their deposited energy is sufficient. Once all relevant cells are added, separating the topological cluster from other, potentially overlapping, clusters is necessary to assign the energy to the respective decay products. For this, a Gaussian mixture model is used, as it is assumed that the topological cluster consists of multiple Gaussian-distributed energy clusters, each belonging to a single particle in the calorimeter. In the case of the ECAL, *superclusters* are possible: clusters that consist of multiple sub-clusters with significant spread along ϕ, but narrow in η, due to bremsstrahlung photons.

Charged Particle Tracks The reconstruction of charged particle tracks from individual detector signals (hits) follows a Kalman filter approach [50] with an additional iterative component. At the start, seeds are generated from small numbers of hits in the tracker's pixel and strip layers. Starting from a seed, the Kalman Filter algorithm is iterative: it propagates the trajectory layer by layer, adding compatible hits at each step. After all feasible hits are attributed to a track seed, a final fit and smoothing with a Runge-Kutta propagator are performed. Finally, a quality check decides whether to keep or discard the reconstructed track.

Due to the numerous particles present in a single collision, the computational complexity would be exceptionally high if all tracks were reconstructed at once. Instead, the iterative tracking strategy: tracks reconstructed with high confidence are removed from the available hit collection after each pass, and the criteria are loosened for subsequent iterations to recover more challenging tracks.

Electron Tracks Electron tracks can exhibit more significant changes in trajectory during their flight due to the emission of bremsstrahlung photons, which account for a considerable fraction of the electron's energy. Therefore, an extended variant of the charged hadron track reconstruction algorithm is used with a refitting step.

A Gaussian Sum Filter (GFS) [51], which is more tolerant towards sharp changes in momentum, replaces the Kalman fit for the electron tracks. Because the GFS approach is more computationally expensive than the Kalman approach, only a subset of all seeds is considered for this method. The seed selection occurs in two ways: a track is determined to be consistent with an electron during the Kalman Filter, or a cluster in the ECAL hints towards an electron. In both cases, a compatible track is then refitted with the GFS approach and matched to the corresponding ECAL clusters. Notably, for a track that passed both the Kalman Filter and the GFS, the GFS overwrites the results, though a reference to the Kalman filtered track is kept.

Muon Tracks Similar to other charged particles, muon tracks are initially reconstructed in the tracker with a Kalman Filter approach [52]. The high precision of the muon system of the CMS detector also allows for a second set of reconstructed tracks in the muon system, following a similar method (*Standalone muon*). Once both sets of tracks are constructed, they can be matched to form full tracks from the center of the detector to its outermost edges. There are two approaches: an inside-out method that extrapolates tracks from the inner tracker to at least one active muon segment (*Tracker muon*) and an outside-in method that matches *Standalone muon* with tracks from the inner tracker (*Global muon*). Especially *Global muon* tracks are of high quality, allowing for excellent reconstruction of muon properties.

2.3.2 Particle Flow

The dominant reconstruction algorithm for final-state particles used by the CMS collaboration is the PF algorithm [11]. This holistic approach to reconstruction is enabled by the high granularity of the CMS detector in $\eta - \phi$ space, combining information from all subdetectors: the tracker, ECAL, HCAL (including the HF), and the muon system. The information measured in the detector is first used to construct basic elements, and then to compile and sort the particle candidates into different Particle Flow candidate (PFcand) categories.

Objects, both tracks and clusters from the calorimeters, are considered *PF elements*, which are then linked to combine into *PF blocks*. PF blocks are groups of *PF elements* that collectively describe a particle or decay, optimized for minimal distance in $\eta - \phi$ space. For instance, an ECAL cluster may be linked to an electron's GFS-reconstructed track if the track's trajectory points toward the cluster, which could represent a bremsstrahlung photon emitted by the electron. The block then contains both the track and the cluster as linked elements, forming the basis for the final PFcand reconstruction.

Finally, after the *PF elements* are linked into *PF blocks*, they are classified as one of seven types with the following conditions:

- **Muon** μ: tracker track linked to a *Standalone muon*, or a *Global muon*.
- **Photon** γ: ECAL cluster with no associated track.
- **Electron** e: GFS track linked to an ECAL cluster or supercluster.
- **Charged hadron ($\mathbf{h}^{\pm}$)**: tracker track linked to calorimeter clusters in the ECAL and/or HCAL. In addition, the track may not be matched to a GFS electron or muon track.
- **Neutral hadron ($\mathbf{h}^{0}$)**: calorimeter cluster in the ECAL and/or HCAL, but without an associated track.
- **HF electromagnetic $\mathbf{HF_{EM}}$**: signal in the HF associated with EM activity.
- **HF hadronic $\mathbf{HF_{h}}$**: signal in the HF associated with hadronic activity.

Of these, the two HF types stand out, as they are not linked with any other *PF elements* due to their coverage in only the high η range. An example of how basic variants of the non-HF types are detected in the CMS detector is illustrated in Figure 2.2.

For many analyses, the PFcands are only the first step in reconstructing more complex objects such as jets or high-level quantities like missing transverse energy. For hadronic tau reconstruction and identification, PFcands are well-suited, as they provide fine-grained information that enables prong counting and π^0 reconstruction. In addition, precise reconstruction of electrons and muons allows for efficient rejection of lepton backgrounds. The PFcands serve as both the main input objects of the HPS algorithm and as context for the ML-based classification approach outlined in Section 2.4.

2.3.3 Hadron-Plus-Strips Algorithm

After reconstructing the basic particles through PF, most analyses go on to utilize them for the reconstruction of higher-order interactions. The reconstruction of hadronically decaying tau leptons is challenging, as their decays are difficult to distinguish from leptonic and QCD jets backgrounds. Still, the limited range of available DMs allows for discriminating criteria. The PFcands involved in hadronic tau decays are typically more collimated in a jet than in QCD jets, and the involved neutral pions that decay into photons allow for a unique identification handle.

The goal of the HPS algorithm [12] is to reconstruct hadronically decaying tau leptons along all prominent DMs. As outlined in Section 2.1.3, hadronic tau decays occur mainly in one of five variants, with $\mathrm{h}^{\pm}$ denoting either a charged pion or kaon:

- DMI=0: $\tau^{\pm} \rightarrow \mathrm{h}^{\pm}\nu_{\tau}$
- DMI=1: $\tau^{\pm} \rightarrow \mathrm{h}^{\pm}\pi^{0}\nu_{\tau}$
- DMI=2[1]: $\tau^{\pm} \rightarrow \mathrm{h}^{\pm}\pi^{0}\pi^{0}\nu_{\tau}$
- DMI=10: $\tau^{\pm} \rightarrow \mathrm{h}^{\pm}\mathrm{h}^{+}\mathrm{h}^{\pm}\nu_{\tau}$
- DMI=11: $\tau^{\pm} \rightarrow \mathrm{h}^{\pm}\mathrm{h}^{+}\mathrm{h}^{\pm}\pi^{0}\nu_{\tau}$

They can be split into 1-pronged and 3-pronged DMs, based on the number of charged hadrons involved.

The reconstruction of the jets relies on an anti-k_t ($q = -1$) algorithm [53] with $R = 0.4$:

$$d_{ij} = \min\left(p_{\mathrm{T},i}^{2q}, p_{\mathrm{T},j}^{2q}\right)\frac{\Delta R_{ij}^2}{R^2}, \qquad d_{iB} = p_{\mathrm{T},i}^{2q} \tag{2.9}$$

With $p_{\mathrm{T},i/j}$ denoting the transverse momentum of the i-th and j-th particles and ΔR_{ij} denoting the distance between them in $\eta - \phi$ space. The algorithm combines particles sequentially: if d_{ij} is the global minimum of all distances, then particles i and j are combined into a new particle. Otherwise, if d_{iB} is the minimum, then i is identified as a jet and the merged set of participating particles is removed from the particle collection. Because of the negative exponent, hard particles (high p_{T}) act as stable seeds that preferentially attract softer particles (low p_{T}). Therefore, soft particles tend to merge with nearby hard particles rather than with other soft particles. This process is then repeated until all particles are clustered into jets.

The HPS algorithm refers to the charged hadrons as well as the *strips* that are recorded due to the behavior of π^0 particles, which promptly decay into two photons each. Afterwards, photons may convert to e^-e^+-pairs, which in turn may radiate bremsstrahlung

[1] Generally reconstructed as DMI=1 with the HPS algorithm

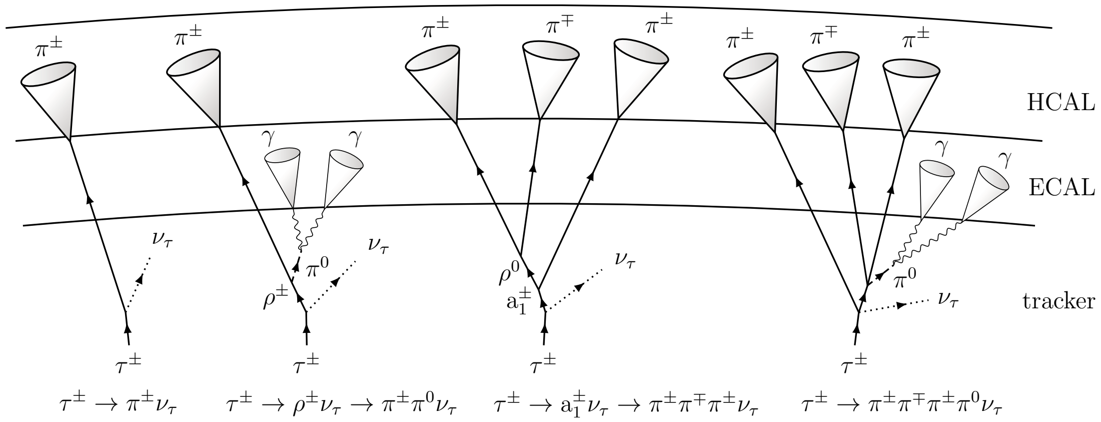


Figure 2.3: Sketch of the dominant HPS reconstructed hadronic tau decay channels and the subdetectors in which the decay products are predominantly detected [13].

photons, resulting in the early stages of an EM shower. While the trajectories of photons remain unaffected by the magnetic field, those of electrons and positrons are altered. During this cascading process, the spread in ϕ-direction therefore increases, while the spread in η-direction remains small. On the surface of the ECAL, in which the majority of the EM shower occurs, the resulting shape of the electrons and photons resembles *strips*.

The HPS algorithm uses anti-k_t- identified jets as seeds. All PFcands in a radius of $\Delta R = 0.5$ in $\eta - \phi$ space around the seed jet axis are passed on as valid τ_h candidates. HPS starts each strip with the highest p_T PF photon or electron and then iteratively adds nearby PF electrons and photons with a window to the strip ($\Delta\eta \times \Delta\phi \approx 0.1 \times 0.2$). The momentum of the strip is then given by the combined sum of all included particles, and the position is determined from a momentum-weighted average. The process is repeated until all electrons and photons of the jet are assigned to strips.

After reconstructing both charged hadrons and neutral pions, HPS combines the information to either reject the candidate or determine a DMI for it. For this combination, only the *strips* and charged particles with the highest p_T are considered for each seedling jet. Acceptance requirements include a mass window for the resonances of the 3-pronged DMs and a charge requirement of ±1. The charge requirement is lifted for reconstructed τ_h candidates with a missing charged hadron (DMI = 5, 6).

For the HPS algorithm specifically, the $\tau^\pm \to \mathrm{h}^\pm \pi^0 \nu_\tau$ and $\tau^\pm \to \mathrm{h}^\pm \pi^0 \pi^0 \nu_\tau$ DMs are effectively merged as the additional π^0 is generally smeared with the initial π^0. Therefore, reconstruction based on the HPS algorithm generally omits references to the DMI 2, that is associated with the $\tau^\pm \to \mathrm{h}^\pm \pi^0 \pi^0 \nu_\tau$ decay. Instead, the DMI 1 refers to the combination of both modes. Figure 2.3 illustrates DMI 0, 1, 10, and 11, including their final state particles and in which subdetector they are typically reconstructed in.

At this point, HPS returns a list of reconstructed τ_h candidates. While this list is likely to contain genuine τ leptons, there is also a significant background of other origins:

- **Quark and gluon induced jets**: can mimic tau decay patterns as they may contain both charged hadrons and neutral pions.
- **Electrons**: can be misidentified as 1-pronged τ_h decays with bremsstrahlung potentially resulting in additional *strips*.
- **Muons**: occasionally undergo hard interactions or radiate photons, producing signals in the ECAL or HCAL that resemble τ decay products signals.
- **High pileup**: can result in overlapping collisions and a composite reconstruction.

Therefore, an additional identification step is required to validate and discriminate the reconstructed τ_h candidates.

2.4 Machine Learning-Based τ_h Identification

The identification of particles from particle candidates has been an invaluable topic in HEP research, especially as individual collisions became increasingly complex. Modern reconstruction algorithms focus on more inclusive reconstructions, which also include significant fractions of false positives from non-signal sources. In exchange, discriminants are employed to filter through those candidates and improve the signal-to-background ratio.

Before the introduction of deep learning technologies, these classifiers were based on hand-picked cuts, Fisher discriminants, or decision trees [12]. Since then, ML-based techniques have gained widespread adoption in the field and shown significant gains over traditional methods. For the identification of hadronically decaying tau leptons, one such approach came in the form of the DeepTau NN-architecture. DeepTau V2.5 [13] is currently used in the reconstruction effort of Run 3, while its predecessor (DeepTau V2.1 [54]) was the standard during Run 2. Further research into Transformer-based architectures led to the development of the TauT [55], a potential update to the existing classifier, which could find application during the HL-LHC era.

This section outlines the datasets used for the training and testing of both DeepTau V2.5 and TauT models, their respective architectures, and how they are applied to discriminate data. Physics performance comparisons of the two architectures are shown as part of Chapter3.

2.4.1 Datasets

Three datasets contribute to the training and evaluation of the DeepTau and TauT models: the Shuffle and Merge (S&M) dataset for primary training, a testing dataset for evaluation, and an adversarial dataset for fine-tuning DeepTau V2.5. Each of them consists of HPS-reconstructed τ_h candidates, as well as PF and RECO level information of the events they originate from. They mainly consist of simulated data based on 2018 detector conditions, though the adversarial dataset discussed in Section 2.4.2.3 necessarily includes real data.

The number of examples contained in the three datasets was:

- **S&M dataset**: 96 million
- **Testing dataset**: 9.3 million
- **Adversarial dataset**: 187500[56]

The samples that make up the training and testing datasets were selected to provide sufficient examples of each of the four output classes: genuine hadronic taus, electrons, muons, and QCD jets. The selection of samples between the training and testing datasets relied on separate datasets to prevent training information from seeping through to the evaluation of the NN. Details on the sample types, their selection criteria, and the corresponding output classes are detailed in Appendix A.1.

After selecting τ_h candidates and assigning them to one of the four output labels, an additional step is performed on the training data to improve batching behavior. Any two batches selected from the dataset should be statistically similar. In addition, every batch should be sampled in an unbiased way from a uniform distribution $p_\mathrm{target}(x_s)$ over a set of *spectrum variables* $x_s = \{p_\mathrm{T}, \eta, \mathrm{class} \in \{e, \mu, \tau_\mathrm{h}, \mathrm{jet}\}\}$. The goal of this approach is to allow for a flat distribution that shows no oversampling for any region of the *spectrum variables*.

The algorithm to compose such a dataset consists of a loop over two steps: first, a random τ_h candidate τ_c is picked from one of the input samples, with probability proportional to the number of candidates in that sample. Then, the candidate is included in the dataset with a probability of

$$p(\tau_c) \propto \frac{p_\mathrm{target}(x_s(\tau_c))}{N(\mathrm{bin}, \mathrm{class})}, \tag{2.10}$$

where $N(\mathrm{bin}, \mathrm{class})$ is the number of samples already present in the bin for that output class. An additional correction ensures that the ratio between the number of τ_h candidates in the final p_T-bin and all other p_T-bins per class is higher than 10^{-3}.

After the algorithm completes, a Kolmogorov-Smirnov test [57] is used to validate the compatibility of randomly selected subsets. The S&M dataset created with this approach contains roughly 10^8 τ_h candidates, split over 499 files.

2.4.2 DeepTau

The DeepTau NN architecture is based on multiple novel concepts, including the use of low-level features, convolutional NN layers, customized training loss, and adversarial domain adaptation. It has demonstrated significant improvements over previous methods, lowering misidentification rates by up to 97% in some regions [58].

The two standout features of the DeepTau V2.1 NN model were the low-level data features it used as input and its convolutional architecture. In combination, they enable a more comprehensive description of an event than conventional approaches that utilize only high-level features.

2.4.2.1 The DeepTau Architecture

The architecture of DeepTau is based on multiple one- and two-dimensional convolutional layers. These layers are well-known in the field of Computer Vision and can be understood as filters that combine small image segments into broader concepts. For example, such a layer might be able to identify a curved shape of pixels as a curve, or multiple curves as a face. The use of such layers in HEP stems from the motivation that detectors, such as CMS, act as *cameras* that capture an image of the particle states for a collision in $\eta - \phi$ space.

Therefore, convolutional layers should be capable of capturing the complex relationships among interacting particles, thereby implicitly reconstructing the decay history. For this approach to function, two conditions have to be fulfilled: the inputs must be structured as an *image* that allows for convolution, and the input data must be sufficiently low-level to observe dependencies between the constituent particles.

Input Shape The architecture's input shape is reminiscent of an image, in that all particles are mapped to a two-dimensional $\eta - \phi$ plane. There are two separate grids in this space, both of which get filled with the particle information, based on the position of the particles, as seen in Figure 2.4. The inner grid relates to a signal cone with a radius $\Delta R_s = 0.1$, while the outer grid relates to an isolation cone with a radius $\Delta R_i = 0.5$. Although the grids themselves extend beyond these radii, no data is entered into the grids outside those limits. The architecture is therefore in line with the HPS algorithm as both only take particles within $\Delta R < 0.5$ into account. The spatial granularity of the grids is 0.02 and 0.05 for the inner and outer grids, respectively.

Each grid cell is filled based on the positions of the constituent particles, with zero padding if no particle is present. If there is more than one particle located in a single cell, then the particles are ranked by their transverse momentum, and only the top-ranked ones are kept.

Low-Level Features While high-level features, such as those derived from the HPS algorithm, are well motivated in physics, they also significantly condense the available information. Especially in the face of rising pileup, such features can obscure critical details present in lower-level reconstruction products. Instead of relying solely on hand-crafted features like those from HPS, DeepTau also uses RECO information for reconstructed leptons and PFcand information. Between these two, PFcands offer a broad picture of the reconstructed final state particles, but RECO particles provide a more refined and detailed description of the individual particles. For most RECO particles, a PFcand also exists, though the two reconstruction methods may differ in some details of the particle. A summary of the input features can be found in Appendix A.2.

Particles from both the RECO and the PF steps are combined into one of three categories:

- e/γ: contains PF electrons, PF photons, and RECO electrons
- μ: contains PF muons and RECO muons

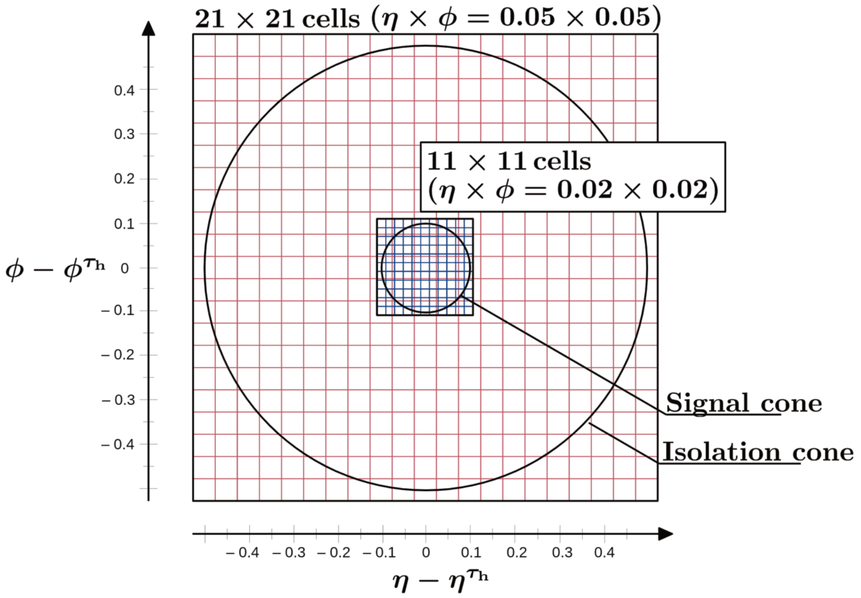


Figure 2.4: Layout of the DeepTau input in $\eta - \phi$ space [13]. The two circles signify the inner and outer cones, while the red and blue grids show the spatial resolution. Cells outside of the respective cones remain empty and are zero-padded.

- $h^{\pm}/h^0$: contains PF charged and neutral hadrons

Each of these particles is feature-rich with $\mathcal{O}(20 - 30)$ features per particle, depending on whether they originate from PF or RECO.

Notably, the features of different particle types are not merged directly but concatenated. As a result, an input feature vector may contain only information on one of the types (like PF photons) while the remainder of the vector (PF/RECO electrons) is padded with zeros. Each cell of both grids is filled in this manner, resulting in a detailed, but often sparse description of a collision.

Each input feature is scaled, shifted, and clamped to improve convergence and prevent numerical instability caused by overly large feature values. This process is performed on a per-feature basis, and while most features are scaled and limited to $[-1, 1]$ or $[-5, 5]$, categorical features remain unscaled [59]. Missing features are handled through masking: if an invalid value is found for a feature, a flag feature is added for all inputs that indicates the missing information, and the invalid value is replaced by a feature-dependent default value [60].

Convolutional Structure The main architecture of the DeepTau NN can be seen as an encoder-decoder pair: an encoder based on a sequence of convolutions and concatenations, and a decoder with fully connected layers, as seen in Figure 2.5.

In both the inner and the outer grid, particle data is embedded for each cell individually: the concatenated electron/photon, muon, and hadron collections are processed by multiple fully connected layers. Then, their respective outputs are concatenated and processed through additional fully connected layers to condense all particle information of a grid cell into a single embedded vector. The fully connected layers used in the embedding process are common between all cells of a grid, but are not shared between the two grids.

After embedding the particle information for each cell in both the inner and outer grids, the cell data is convolved using two-dimensional convolutional layers. The whole data richness of all particles across the two grids is therefore encoded in just two tensors. In addition to the two grids with low-level particle information, the network also receives high-level information as a third data stream. Unlike the grid information, these variables are encoded directly using fully connected layers. The information from the two grids and the high-level variable information is then concatenated, yielding an encoded representation of the candidate. Decoding is then performed in a much simpler manner using a set of dense layers followed by a final softmax. The decoded output is a vector with four entries, one for each of the electron, muon, genuine tau, and QCD jets classes.

2.4.2.2 Loss Function

The loss function of the DeepTau training initially consisted of three terms:

1. a Categorical Cross-Entropy (CCE) loss [61] that discriminates between the signal and the combined three background classes
2. a focal loss [62] that separates τ_{h} from all other classes by penalizing high confidence misidentification with an $F(1 - y_{\tau}^{label}, 1 - y_{\tau}^{pred})$-approach
3. a focal loss that rewards non-tau categorical high confidence discrimination under the condition that the prediction score for the τ_{h} class is at least 0.1

The goal of this combined loss is primarily to separate genuine τ_{h} from background. In the second order, it also encourages separation between the background classes when the τ_{h} scores are sufficiently high. The full loss and gradient functions, including the adversarial loss described in Section 2.4.2.3, are shown in Appendix A.4.

2.4.2.3 Adverserial Training

Motivated by discrepancies between the performance on simulated and genuine collision data, an adversarial approach was introduced with DeepTau V2.5 to fine-tune the model. After an initial training without adversarial training, the model is fine-tuned to achieve greater consistency, especially for the region of high-confidence discrimination against jets.

Figure 2.5, illustrates the NN architecture, where the encoded candidate data is separated into two different decoders: one for the signal-background separation and one for the simulated-real data tuning. The model output's four signal and background classes, plus an additional adversarial output y_{adv} that predicts whether a candidate originated from simulated or real data.

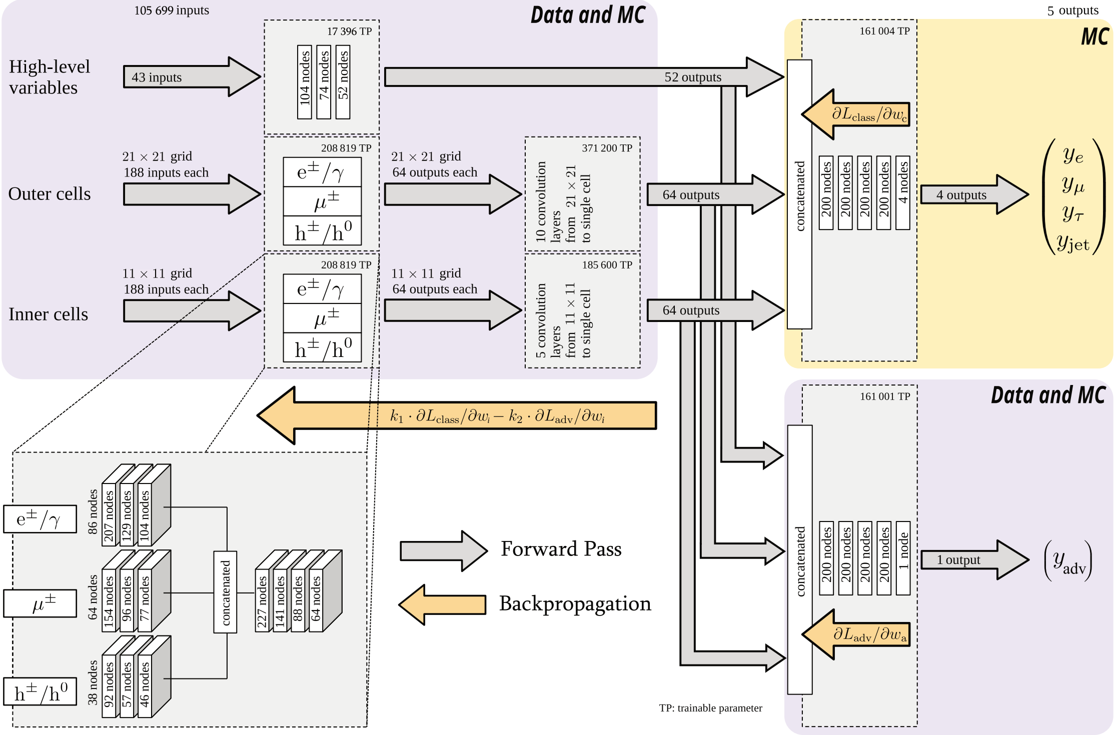


Figure 2.5: Architecture of the DeepTau NN, adapted from [13, 58]. Each candidate's input consists of high-level features and two grids with low-level features. Each grid cell takes in three collections of data for the different particle types. Information per grid cell is merged by a series of fully connected layers, a concatenation of the three categories, and another set of fully connected layers. Afterwards, the cell results are convoluted with two-dimensional layers for both grids. The high-level variables are instead processed through multiple fully connected layers. The concatenated data from these three streams represents the encoded candidate. Decoding occurs via a set of fully connected layers and a final softmax. The output describes the network's confidence that the candidate originated from an electron, a muon, a QCD jets, or a genuine tau. With DeepTau V2.5, an additional output was added for the adversarial approach described in Section 2.4.2.3. For this, a separate decoder is added, and the gradient propagation is adjusted accordingly.

The loss related to this prediction is a binary cross-entropy term [61], though it is not directly added to the remaining loss terms. Instead, it is processed separately and only merged at the gradient level via gradient reversal, as described in Appendix A.4. This ensures that the signal-to-background separation is largely retained, with the additional requirement that the performance of simulated data matches that of real data.

To train the DeepTau model with the adversarial component, an additional dataset with both simulated and high-purity genuine τ_h candidate data was necessary. The main challenge was that generator-level information was not available for real data, making it difficult to construct a sufficiently pure τ_h sample. Details on the selection criteria are outlined in the original publication [13].

The gain from the adversarial approach was a reduction in the relative difference between simulated and real data in the highest-confidence jet classification bin from 17.4% to 0.9%.

2.4.3 TauTransformer

The TauT architecture is an in-place reimagining of the DeepTau architecture that relies on the same PF and RECO input data, with adjusted data loading. Instead of convolutional layers, the TauT relies on the transformer architecture [63], which has attracted significant attention for its relevance in large language models. Like DeepTau, it also relies on low-level particle information, but with significantly reduced zero-padding and greater sensitivity to particles with slight spatial separation.

2.4.3.1 Motivation

While the DeepTau architecture has demonstrated remarkable performance compared to previous identification methods, its convolutional structure introduces biases that do not optimally reflect the physics of τ_h decays. Several limitations motivate the identification architecture.

Mismatch with Convolutional Assumptions Convolutional NN layers are typically used in the description of images, as they are capable of constructing higher-level information from nearby data points in a multi-dimensional space. This includes invariance under translation, a property that does not necessarily apply to the τ_h identification scenario in $\eta - \phi$ space. Unlike the high-multiplicity jets resulting from QCD processes, τ_h-jets rely much more on the explicit correlations between constituents for identification.

In contrast to most pictures, an event also consists of a limited number of points with concentrated information that relate across potentially large ranges. Convolutional filters only take in information from nearby constituents and can therefore miss the more crucial, wide-range relations. While all data is eventually convolved across the multiple convolutional layers, this does not necessarily reflect the decay order intended by the convolutions.

Sparse and Coarse Input Grids As outlined in Section 2.4.2.1, the DeepTau input is mainly shaped as two grids. These grids remain mostly unfilled because the number of cells differs from the number of constituents per candidate. Roughly 99% (90%) of cells in the inner (outer) grid are padded with zeros. The main information gained from the *picture* approach is the implicit positional data required by the convolutional layers. Jets resulting from τ_h decays are generally low-multiplicity and rely on fine-grained constituent information. In the DeepTau architecture, some constituents might not be included in the NN input due to the granularity of the two grids.

Pileup Sensitivity The DeepTau input structure only selects the top particle constituents, ordered by p_T, for each cell and type. All other constituents are discarded, leading to a loss of information for interactions with many densely located particle constituents.

Even with particles of different types, where no information is explicitly lost, another issue remains. Embedded cell information represents the mixture of constituents in the cell, not necessarily the individual participating particles in a local process. In high-pileup conditions where particle locations may overlap without direct interaction, this can misrepresent the factors contributing to a cell's status in the following layers. Especially with the expected rise in pileup during the HL-LHC phase, this limitation needs to be overcome.

Implicit Encoding of Particle Types The information of the particle type in the cells of the grid is only provided implicitly through the position of the particle information in the input categories of the cells. Even among the non-empty cells, many of the seven types that contribute to the input are not filled at the same time, leading to more zero-padding in the input.

Limited Range of Input Grids The DeepTau architecture adheres to the $\Delta R < 0.5$ condition used in HPS, and constituents outside this range are not entered into the input grids. While this limitation is well motivated in the collimated nature of τ_h jets, it still limits the information available to the network. The impact of higher allowed ΔR constituents is uncertain and challenging to study with the DeepTau architecture, due to the quadratic increase in grid size with ΔR.

2.4.3.2 The TauTransformer Architecture

The TauT architecture, like the DeepTau architecture, consists of an encoder-decoder pair, but with an entirely different encoding scheme. Figure 2.6 illustrates the TauT architecture, including a zoom-in of the particle embedding step. Instead of embedding particle information cell-specifically, the TauT architecture embeds it particle-specifically. Afterwards, all particles are processed through a series of self-attention layers to infer meaning to the individual particles in relation to each other.

These layers, commonly associated with transformer architectures, have gained much renown for their use in large language models [64]. The layers can be described as

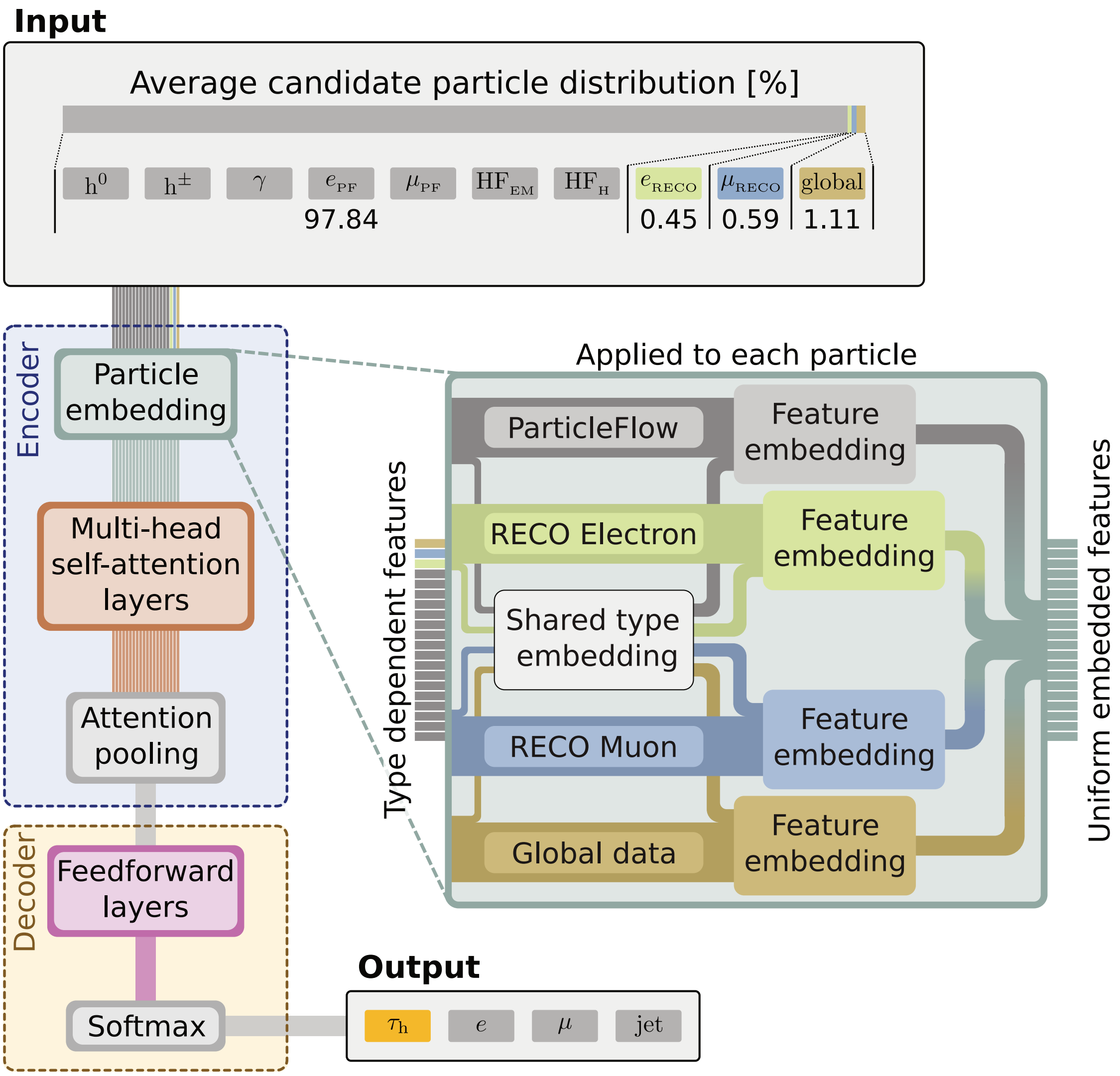


Figure 2.6: Diagram of the TauT architecture. The input mainly consists of PFcand particles with only a small fraction of other collections: RECO e/μ, and global detector information. Each particle of a candidate is embedded with a shared type embedding (simple embedding layer) and a separate feature embedding (feedforward dense layer), depending on the collection. Afterwards, the particle array of dynamic length is encoded with a series of self-attention blocks and a pooling layer. The decoding consists of multiple fully connected layers and a final softmax. The output consists of confidence scores regarding the four signal and background categories.

an estimation of how important each input constituent is to each other constituent in the given context. In addition, multiple *attention heads* enable these layers to determine importance from multiple *points of view*, providing a broader representation of the connections.

This architecture has found many applications, even outside text-based usage scenarios, as the principles of *importance* are widely applicable, including in the HEP field. Some topical examples include the PARTICLETRANSFORMER [65] used in general jet tagging, transformer use in shower simulation [66], and their use in charged particle track reconstruction [67]. Similarly, the TauT architecture employs self-attention layers due to their well-matched properties for jet identification and their highly parallelizable computational model. Finally, after the self-attention layers and after pooling all the constituents, a dense decoder is employed, similar to the DEEPTAU model. This approach addresses concerns about the DEEPTAU architecture and reduces the number of NN parameters in the process.

The initial work [55] included studies regarding the impact of the four input collections, a comparison between categorical and binary NN, and an investigation into a token pruning scheme. The architecture I employed aligns with the best-performing architecture from the previous study, with minor adjustments to accommodate data normalization and a slightly revised set of input features. Appendix A.3 states the architecture hyperparameters used in this thesis.

Adjusted Input Data Over the course of the TauT development, additional PF, RECO, and global detector features were added to enhance the identification accuracy. New coordinates were added to all particle constituents, though most other additional features relate to RECO electrons exclusively. The position of each constituent is stated in the form of a distance in $\eta - \phi$ space ΔR and a polar angle relative to the jet axis θ_{rel}:

$$\Delta\eta = \eta_{const} - \eta_{candidate}, \qquad \Delta\phi = \phi_{const} - \phi_{candidate} \tag{2.11}$$

$$\Delta R = \sqrt{(\Delta\eta)^2 + (\Delta\phi)^2}, \qquad \theta_{rel} = \text{arctan2}(\Delta\phi, \Delta\eta) \tag{2.12}$$

where arctan2 is the two-argument arc tangent function that provides full angular coverage. This treatment allows for arbitrary ΔR-limitations, since computational complexity scales quadratically with the number of particles, not with the size of a mostly empty grid. However, it is not equivalent to explicit *positional embedding*, as the ΔR feature is treated in the same manner as all other features. This approach allows for the inclusion of event information up to the $\Delta R < 0.8$ cut inherent to the S&M dataset itself.

Just as with the DEEPTAU input pipeline, missing features were handled through masking. In addition to the performance-oriented features added during the original development of the TauT architecture, I made more minor adjustments: I removed some flag features that contained identical values across the entire training dataset, and added validity flags for features that contained invalid information for some constituents. A complete list of all input features for the TauT architecture is available in Appendix A.2.

The original implementation of the TauT architecture was trained without the data scaling described in Section 2.4.2.1. For this thesis, I reimplemented the scaling using a simplified approach: a standard scaler trained on the respective training data was applied to all inputs, without clamping or regard for categorical data. During model application, the respective scalers are also applied to the inference data.

Particle Embedding The particle embedding of the TauT architecture is a significant departure from the rigid *global feature+grids*-approach of the DeepTau architecture. As mentioned, the often sparsely populated cell embeddings were replaced by a list of particle embeddings of a dynamic size. Each PF and RECO particle is embedded individually. The global detector information is also embedded as an abstract particle to enable direct encoding within the transformer architecture, though it lacks a ΔR value. In this regard, the global information should be understood as *context* for all other particles.

The embedding step starts with the 10 types of particle-like tokens in the TauT architecture:

- five previously utilized PFcand types (e, μ, γ, $\mathrm{h}^{\pm}$, h^{0})
- two previously not utilized PFcand types (HF tower identified as a hadron or EM particle)
- RECO electrons and muons
- global detector information

These types are split into four collections: RECO electrons, RECO muons, global detector information, and the PFcands, which include all other types. The particle type information of the input particles is embedded via a common learnable embedding layer into a two-dimensional feature, which is then added as an additional feature to each particle.

Separate dense feedforward layers are used for the four collections to account for the differences in input shape. The tokens from the four collections are then concatenated and passed through a dropout layer, creating the embedded context for the following transformer layers.

This change to the embedding resolves sparsity in the input dimensions and increases sensitivity to pileup. Furthermore, with explicit encoding of particle types, each constituent particle can be expressed independently of the others, allowing for a tighter, potentially more accurate description of the input state. Finally, it allows the full scope of reconstructed particles to participate in the identification process, not just an arbitrary subset.

Self-Attention Layers The encoding of the embedded particle constituents happens through a series of self-attention layers, each consisting of the typical building blocks used in transformer architectures [63]: a multi-head attention layer, followed by a dropout layer, a layer normalization, a fully connected layer, and another pair of dropout and normalization layers. Residual connections are used with both layer normalizations. This construct is then stacked sequentially until a final pooling layer combines the attention scores of the processed tokens.

The properties of transformers align well with HEP jets, as they allow unsorted collections of constituents to interact freely and form distinct relationships. Such interactions might relate to a common origin particle, their alignment in $\eta - \phi$ space, or more complex higher-dimensional relationships. Instead of the implied translational invariance of convolutional NN architectures, transformers provide an invariance under token permutation without positional encoding.

An embedding with a dynamic size that depends on the number of input particles is difficult to handle for many NN architectures, which often require static shapes. The transformer architecture allows for such dynamic shapes and further accommodates unordered input particles from different sources, such as PF or RECO.

The transformer encoding approach also allows auxiliary information, such as global detector features, to be added as context tokens rather than requiring them as separate inputs.

Decoder Similar to the DeepTau architecture, decoding into classification confidence is performed by a set of fully connected layers followed by a final softmax layer.

2.4.3.3 Other Differences

While the TauT architecture can, in theory, be used as a direct replacement for the DeepTau architecture, multiple parts of the training pipeline are not yet fully implemented for the TauT model. In this thesis, I refrain from implementing them as I value the comparably pure state of the TauT architecture for my data-related performance studies.

Loss Function Opposed to the complex loss function employed for the DeepTau training process, the TauT training relies on a simple CCE H_{cat} with no additional terms:

$$H_{cat}(\mathbf{y}^{label}, \mathbf{y}^{pred}) = -\sum_i y_i^{label} \log y_i^{pred}, \tag{2.13}$$

where $\mathbf{y} = (y_e, y_\mu, y_\tau, y_{\text{jet}})$ are the predictions and generator-level truth. While this loss does focus on categorical separation, it does not do so in the finely-tuned manner that the DeepTau loss function, shown in Appendix A.4, does. A greater separation between background categories could therefore result in a diminished performance in the more relevant τ_h vs background scenarios.

Adversarial Training Similarly, the adversarial training was not part of the TauT training. My comparisons in this thesis rely only on simulation data, and discrepancies in performance between simulated and real data would not be visible. As the overall structure of the encoder-decoder architecture remains unchanged, adversarial training is feasible.

Optimized Batching Scheme As part of previous studies [55], a *smart-batching* scheme was implemented to reduce zero-padding during training further. The primary source of computational complexity is the self-attention layers, where computational costs scale quadratically with the number of tokens per τ_h candidate. In typical mixed batches, all candidates are padded to align with the candidate that provides the most tokens.

Because the variance in the number of tokens per candidate is high, as seen in Section 3.7, a substantial amount of padding and therefore unnecessary computation occurs. The *smart-batching* approach improves on this by grouping batches by their number of tokens, which reduces zero-padding and lowers the average number of computations by more than a factor of two. This approach is performance-oriented and could potentially introduce bias during training, as the batches are no longer entirely random. I assume that the impact on discrimination performance is negligible and outweighed by the increase in runtime speed.

2.4.4 Application of Hadronic Tau Identification

To utilize a trained NN model in binary classification tasks (e.g., hadronic tau lepton vs. QCD jets), the output scores must be projected onto the respective classes. The binary discriminant between a hadronic tau and any of the background classes α is given by:

$$D_\alpha(\mathbf{y}) = \frac{y_\tau}{y_\tau + y_\alpha}, \tag{2.14}$$

with the NN scores after the softmax layer as $\mathbf{y} = (y_e, y_\mu, y_\tau, y_\mathrm{jet})$.

A standard method of performance comparison is Receiver Operating Characteristic (ROC) curves [68], which plot the misidentification probability over the τ_h identification efficiency. The τ_h identification efficiency, or True Positive Rate (TPR), is defined as

$$\mathrm{TPR} = \frac{N_\mathrm{sig}^\mathrm{pass}}{N_\mathrm{sig}^\mathrm{total}}, \tag{2.15}$$

where $N_\mathrm{sig}^\mathrm{pass}$ denotes the number of genuine τ_h candidates passing the discriminant threshold, and $N_\mathrm{sig}^\mathrm{total}$ the total number of genuine τ_h candidates in the sample. The corresponding misidentification probability, or False Positive Rate (FPR), is given by

$$\mathrm{FPR} = \frac{N_\alpha^\mathrm{pass}}{N_\alpha^\mathrm{total}}, \tag{2.16}$$

where the background class $\alpha \in \{e, \mu, \mathrm{jet}\}$ is evaluated separately for each discriminant. ROC curves therefore visualize the trade-off between signal efficiency and background rejection as the discriminant threshold is varied.

Table 2.2: Target τ_h identification efficiencies for the WPs of the three DeepTau discriminants against electron, muon, and QCD jets backgrounds. The target efficiencies are evaluated with the $\mathrm{H} \rightarrow \tau\tau$ event sample for τ_h with $p_\mathrm{T} \in [30, 70]$ GeV. WPs recommended for analyses by CMS are marked in bold, though the WPs of the discriminant against muons are generally considered equally valid.

Working Point	D_e	D_μ	D_jet
VVTight	60%	—	40%
VTight	70%	—	**50%**
Tight	**80%**	**99.5%**	**60%**
Medium	90%	**99.8%**	**70%**
Loose	95%	**99.9%**	**80%**
VLoose	98%	**99.95%**	90%
VVLoose	**99%**	—	95%
VVVLoose	99.5%	—	98%

Such curves provide a complete picture of the performance that a discriminant achieves across the purity spectrum, though in many cases Working Points (WPs) are utilized instead. Such WPs are only defined for a point along the TPR axis and allow for a less flexible, but standardized usage of a discriminant in an analysis.

Centralized WPs are provided for use in physics analyses, based on the DeepTau V2.5 discriminants, along with suitable corrections [69]. The WPs are defined, based on the hadronic tau identification efficiency of genuine $\mathrm{H} \rightarrow \tau\tau$ event samples, reconstructed as τ_h candidates with $p_\mathrm{T} \in [30, 70]$ GeV, and summarized in Table 2.2. The reduced number of WPs for the discriminant against muons stems from its excellent discrimination performance, even at very low purity.

Strictly speaking, the identification efficiencies that the WPs are based on only directly apply for samples with the same data cuts (e.g., $p_\mathrm{T} \in [30, 70]$ GeV). As a result, both WP identification efficiency and misidentification probability shift when the respective thresholds on the DeepTau output scores are applied to other ranges. In this thesis, I chose to compare the performance of the WPs using identical cuts to those used for the threshold calculation. I also rely on ROC curves to compare the performance of competing ML setups, although explicit WP efficiencies function as distinct points for numerical comparisons at critical purities.

CHAPTER 3

Performance Study of an Attention-Based Hadronic Tau Classifier

The TauT architecture introduced in Section 2.4.3 presents a promising alternative to the currently employed DeepTau architecture. Despite being computationally lighter, initial studies demonstrated impressive performance, often matching or surpassing that of DeepTau. However, those preliminary investigations leave room for improvement, which will be discussed in this chapter.

In this thesis, I extended the studies of the TauT architecture by exploiting the full available training dataset and allowing for sufficient patience to ensure convergence. Beyond confirming and refining the earlier findings, I also took this opportunity to further study the TauT discriminants across two topics: NN training variance using statistically similar datasets; and the effects of non-architectural hyperparameters.

A key motivation was improvements in classification performance during inference without model modifications, referred to as architecture-independent performance improvements. Classifiers like DeepTau are applied to events during the reconstruction process to compute one of many high-level variables for the objects in such events. Therefore, this process should be as lightweight and efficient as possible, given that the full CMS dataset comprises over a trillion examples across measured and simulated event data. Architecture-independent performance improvements stand out against architectural changes, as such hyperparameters allow for measurable performance gains at zero computational cost during inference. This approach complements the studies presented in Chapter 5 onward, focusing on computational efficiency. The architecture-independent performance approach inspired my choice of ablation parameters[1]: training data volume, training epochs, and the maximum number of input particles per training example.

After an initial introduction to the training process in Section 3.1, this chapter addresses the outlined topics in three steps. I begin by surveying the variance of the discriminant derived from the TauT architecture in Section 3.3. Next, I compare the fully trained TauT with the DeepTau V2.5 baseline in Section 3.4. Finally, I present

[1] In the context of ML, ablation studies involve removing or dampening specific features to quantify their impact on the performance.

three ablation studies in Sections 3.5 to 3.7 and conclude the chapter with a summary of the results in Section 3.8.

3.1 TauTransformer Training Setup

The TauT architecture, including the modifications to the input shape introduced in this thesis, is described in Section 2.4.3. This section summarizes the software and hardware environment used for training, along with the configuration choices specific to this work. Deviations from the previous study [55] are highlighted where relevant. The training software was adapted from the TauMLTools git repository [70].

Software and Hardware Environment All training runs were conducted using Tensorflow V2.10 [71, 72], managed through an HTC batch system [73] and executed on the Throughput Optimized Analysis System (TOpAS) local GPU compute cluster introduced in Section 4.2. Training processes ran on a mixture of NVIDIA V100S, V100, and A100 GPUs with 32-40 GB of Video Random-Access Memory (VRAM). The computational limitations and scaling behavior of this setup are discussed in detail in Chapter 5, which analyzes the runtime and energy characteristics of the baseline TauT training process.

Training Configuration and Parameters A complete list of hyperparameters, including training-specific parameters such as the learning rate, is provided in Appendix A.3. The input data was stored as tf.Records, a binary Tensorflow-specific data format optimized for efficient I/O operations. The TauT model was trained in batches of 280-350 candidates, depending on the available VRAM, using the Adam optimizer [74] and a standard CCE loss. Dropout [75] and residual connections [76] were employed to improve training stability, and early stopping [77] was applied with a patience of ten epochs. While the training data volumes varied across the benchmarks presented in this chapter, the validation and test datasets were kept identical across configurations to ensure direct performance comparability.

Differences from the Previous Study In addition to the expanded input feature set, several key differences distinguish this work from the original TauT study [55]. The initial study used only 20% of the available 10^8 training examples (10% for training, 10% for validation). It relied on a different Tensorflow-internal data format (`tf.data.Dataset.save`), which introduced I/O limitations[2]. The initial trainings were restricted to V100 GPUs, requiring approximately 2.5 hours per epoch [78]. In contrast, the rerun performed for this thesis employed the full dataset (90% training / 10% validation) and was primarily executed on A100 GPUs, requiring 3.6 hours per epoch[3]. Total

[2] These limitations were likely caused by a large number of open file handles resulting from the interaction of many input files with a high number of available CPU threads on the TOpAS cluster.

[3] Comparable runtimes on V100 GPUs could not be measured, as full-scale trainings were only performed on A100 hardware.

throughput was therefore increased by up to a factor of 3.5 in the number of training examples processed per second.

Furthermore, the previous study employed a harsh early stopping limit of 3 epochs, compared to the 10 epochs of patience in this rerun. In summary, these changes resulted in total training durations roughly five times longer than those of the initial study, though with a significantly improved discrimination performance.

3.2 Comparison Metrics

To quantify and compare the performance of the trained TauT models, the evaluation used the ROC curve and WPs introduced in Section 2.4.4. For consistency with established CMS conventions, the results were mainly compared at the CMS-recommended DeepTau working points [69]:

- **Against electron**: Tight(80%), VVLoose(99%)
- **Against muons**: Tight(99.5%), Medium(99.8%), Loose(99.9%), VLoose(99.95%)
- **Against jets**: VTight(50%), Tight(60%), Medium(70%), Loose(80%)

for which the percentages state the target TPR of the discriminant against a specific background at threshold. Tables with the precise FPR values at these WPs are available in Appendix A.5.

In contrast to the standard CMS definition of working points, which were derived for τ_h candidates with $p_\mathrm{T} \in [30, 70]\,\mathrm{GeV}$, this study employed an extended range of $p_\mathrm{T} \in [20, 120]\,\mathrm{GeV}$ for both threshold determination and visualization. This adjustment mitigated small shifts along the TPR axis that can occur when thresholds derived from a restricted p_T window are applied to a broader test distribution, as observed in some DeepTau publications. The ROC plots use the DMI defined in Section 2.1.3 to describe the generator-level truth DMs.

The knee point of the ROC curves served as landmarks to quantify the position of the curve along both the TPR and FPR axes. The knee point marks the position of greatest distance between the TPR/FPR-diagonal and the ROC curve, serving as a measure of where the middle of the ROC curve is. These additional points were necessary as the WP performance itself can only quantify scaling in the FPR direction, and shifts in the ROC curve also occurred along the TPR axis. All knee points were located with a simplified Kneedle algorithm [79].

3.3 Performance Variance

The variance of ML-based approaches is often neglected because quantification is challenging. It is possible to minimize the variance between runs of an NN training through fixed seeds and other options to enforce deterministic behavior. But this does not indicate the possible range of performance achievable between non-fixed seeds and statistically identical training datasets. As the studies aimed to compare multiple similarly trained NNs, I first had to quantify what constituted a statistically significant difference between two configurations.

3.3.1 Training Setup

The TauT variance study was conducted as an ensemble of training runs with two levels of randomness: first, all training runs used a unique seed. Secondly, the training runs were performed on subsets of the S&M dataset. Similarly to the original study, a 90% training / 10% validation split was used. While the validation data remained static, the training data was selected at random, with 50 of the 450 available input files used for each run. The variation of the training data served as both a test of the training variance and a cross-check of the statistical properties for the S&M dataset.

I performed this variance study using only a reduced training dataset, and limited the number of separate runs to $N = 11$, due to the significant computational cost associated with rerunning such trainings. While this ensemble test was compatible with the data conditions of the original TauT training publication, complete generalization to other setups is not guaranteed. Nevertheless, a high degree of generalization is expected, as the underlying NN architecture remained identical across all studies, and the variance in input data was reduced when the full training dataset was used. Therefore, I considered the observed variance as guidance for other, similar configurations. An exact analysis of the performance variance would require a similar ensemble for each training setup, which would be computationally intensive.

3.3.2 Classifier Results

The ROC curves of the discriminants defined by the trained NN of the ensemble are shown in Figure 3.1 for the discrimination against electron, muon, and jet backgrounds, respectively. Tables with the precise FPR values at the most relevant DeepTau WPs are available in Appendix A.5. The plots also include the ensemble's mean performance with a 68% Confidence Level (CL) based on the conducted ensemble.

WP Efficiency Variance Results The curves showed that all three discriminants achieved low observed variation across the conducted ensemble for the relevant WPs. The electron FPR had a considerable relative variation at low TPR with up to 18.5%. This variation diminished at higher TPR, and around the **Medium** WP, it reduced to 5.3%. From there on, it remained stable around 5% up to the **VVVLoose** WP. The muon FPR showed an inverse behavior: at the **Tight** WP, it reaches 4.2%. The variation decreased to 2.5% with rising TPR, but then increased again for the **VLoose** WP, to 9.6%. Finally, the jet FPR showed variance in line with the electron FPR. At most, the variance reached 11.7% at low TPR, and then gradually decreased to 4.7% with higher TPR.

ROC Shape Features In addition to the ensemble's variance behavior, this study also allowed an analysis of the ROC curve's shape. Figure 3.2 shows the full TPR

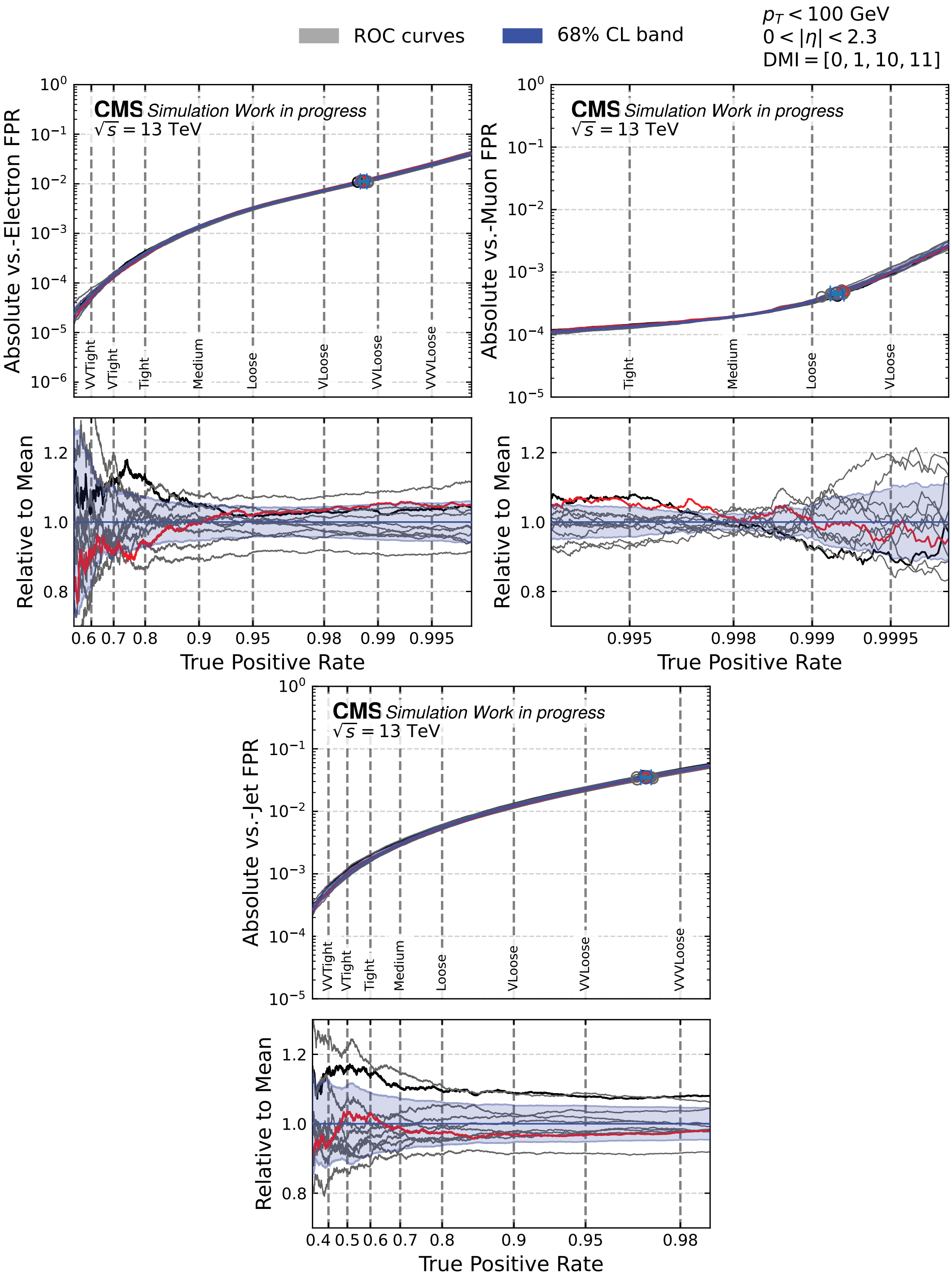


Figure 3.1: Comparison of absolute and relative classifier ROC performance of an ensemble against electron, muon, and jet backgrounds. Individual TauT training runs are shown in light grey, and the 68% CLs based on the conducted ensemble is shown in blue. Red and black curves indicate two example curves with Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee points.

and FPR ranges of the three ROC curves with both linear and logarithmic scaling to illustrate this behavior. Appendix A.6 goes into further detail on how the ROC shape relates to the score distribution.

The four-segmented shape of the ROC curve was visible with logarithmic scaling:

1. (TPR < 0.5) The initial segment with steep slope.
2. ($0.5 <$ TPR $<$ knee) A segment with gentler slope.
3. (knee $<$ TPR $< 1 - 10^{-5}$) A segment with a similar slope to the second segment when mirrored along the (1-TPR)/FPR-axis.
4. ($1 - 10^{-5} <$ TPR) The final segment with nearly flat slope, mirroring the first segment.

The segments transitioned smoothly into each other, creating up to three bends along the whole curve. Although the slope of the first two mirrored the slope of the final two, the lengths of the segments were not necessarily the same.

For the *vs.-electron* curve, the two center segments were of nearly identical slope, while the *vs.-muon* curve was concave in the center, and the *vs.-jet* curve was convex. Most of the relevant WPs resided between the transition from the first to the second segment and the transition between the second and third segments, though the loosest WPs extend into the third segment. These two transitions between segments are referred to as *bends* in this chapter.

3.3.3 Results Discussion

Despite the degrees of freedom in the ensemble training procedure, the resulting discriminants were exceedingly stable. In the range relevant to the WPs, two locations with comparably high variance stood out: the low-TPR region of the *vs.-electron* discriminant before the *bend* of the ROC shape, and the high-TPR region of the *vs.-muon* discriminant past the *bend* of the ROC shape. My explanation for this behavior is based on the effect of focal misalignment between the CCE and the ROC curve WPs, as well as an amplified sensitivity due to steep ROC slopes

Loss vs. ROC Misalignment The contributions of an example to the CCE loss are given by:

$$L_{\mathrm{CCE}}\left(\mathbf{y}^{label}, \mathbf{y}^{pred}\right) = \sum_{i \in \{\tau, e, \mu, \mathrm{jet}\}} -y_i^{label} \log y_i^{pred} \tag{3.1}$$

emphasizing the exact score of the examples, especially for confident misclassification.

In contrast, the performance at a specific point on the ROC curve is determined solely by the number of signal and background examples in the respective subsample, not by the distribution of the example scores. This allows trained NNs with identical final validation losses to differ in the performance of their WPs. In particular, the distribution in the

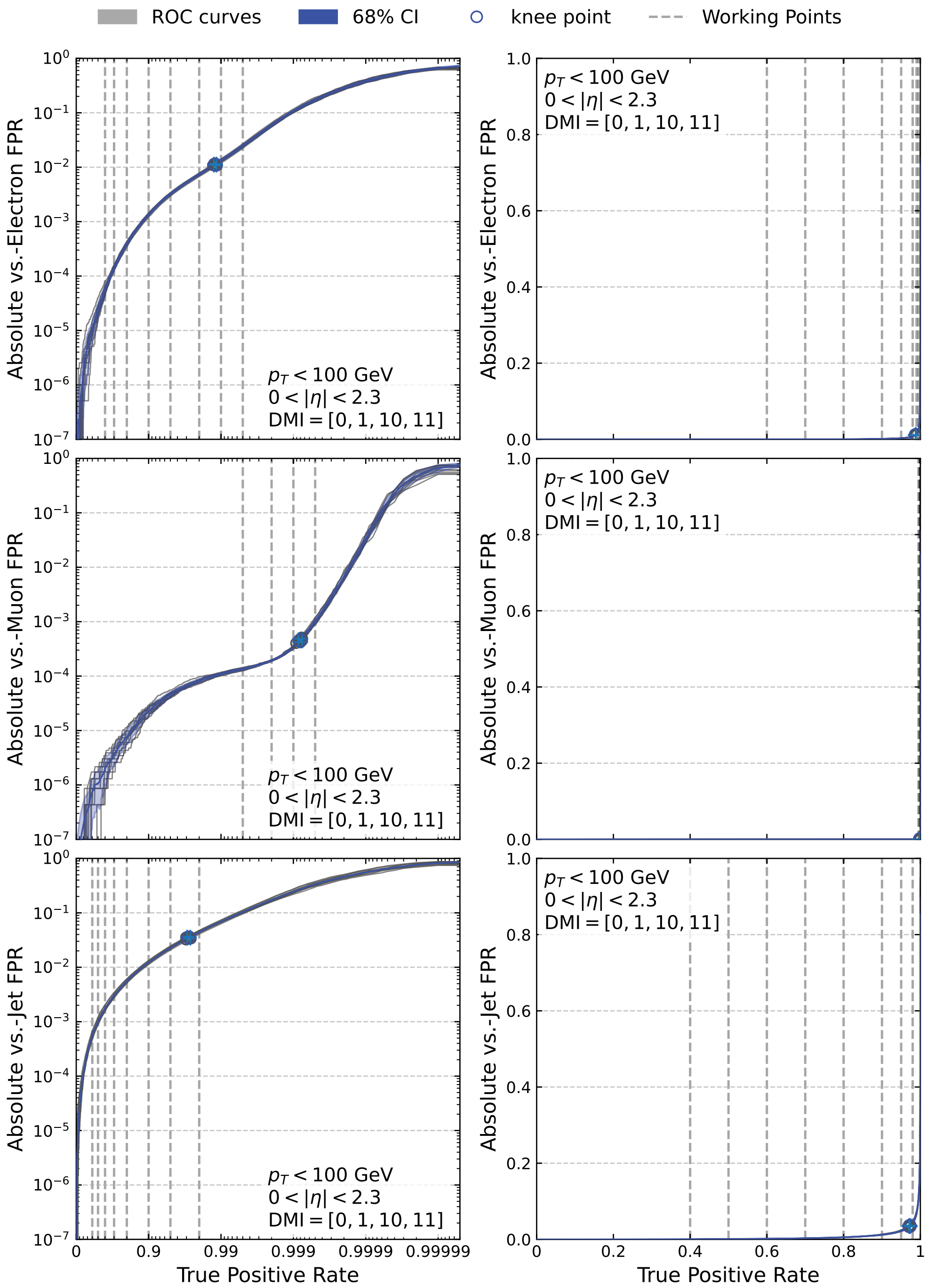


Figure 3.2: Comparison of classifier ROC performance of an ensemble against electron, muon, and jet backgrounds with logarithmic (left) and linear scaling (right). Individual TauT training runs are shown in light grey, and the 68% CLs based on the conducted ensemble are shown in blue. Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee points.

high-confidence background region has little influence on most physics-relevant WPs. A classifier may do exceptionally well at classifying background examples as background with high confidence, but in exchange, struggle with a higher rate of background being misclassified as signal. In this case, the total loss would remain the same, but the performance at WPs, defined by a low TPR, would worsen.

The variance induced in this way is not just theoretical, as it is well visible when two of the ensemble classifiers (red and black in Figure 3.1) are compared directly: Although the final validation loss of the red and black curves differed by less than 0.1%, the FPR at the **VTight** *vs.-electron* WP was 20% lower for the red curve than for the black curve. For higher TPR, this trend then reversed, and at the **VVVLoose** *vs.-electron* WP FPR was 1% lower for the black curve than for the red curve. Similar crossover points were visible across many of the curves, but not necessarily at the same location on the TPR axis.

High-Sensitivity ROC Regions A second phenomenon that amplifies the performance variance between loss-identical NN models is the steep slopes in the first ROC segment. The steep slopes are a direct result of the extreme signal-to-background ratios near the edges of the classification score. An overview of the connections between ROC shape and score distribution is provided in Appendix A.6

The CCE loss does not distinguish between improvements to the signal and background distributions. Even if the signal class's score distribution remained flat and only the background class distributions changed, this would still improve the loss value. Improvements to the loss alone, therefore, cannot be attributed to a specific shift along the TPR or FPR axes in the ROC curve. The performance metric used in physics analyses is based on the FPR at a specific TPR on the ROC curve. In classifier comparisons, it is common practice to only compare this ratio, even though an equal improvement in signal and background separation results in a diagonal shift of the ROC curve. As a result, the misidentification efficiency of WPs in the vicinity of such steep slopes is affected by both vertical and horizontal shifts. In contrast, the misidentification efficiency of WPs in regions of gentler slopes is less affected by the horizontal shifts. Examples for both of these possibilities include the **VVTight** and **VVVLoose** WPs of the *vs.-electron* discriminant. Further illustrating examples of this topic are part of Section 3.5.3.

Implications for Further Studies Based on these results, I conclude that the TauT architecture is stable within 6% across most regions, though low-TPR regions can exhibit higher variations. While these uncertainties cannot be directly applied to other architectural studies due to differences in the training data, the results show that the necessary stability for meaningful comparisons is given, even with random seed selection. This result also confirms the assumption that the S&M dataset is well shuffled and sufficiently homogeneous to yield comparable discriminant performance across subsets of the entire dataset. The study also highlights regions where the discriminant's performance is most sensitive to changes in NN.

3.4 Comparison with DeepTau

After the performance-variance study of the TauT architecture, I compared its performance with other setups. While I mainly focus on relative comparisons of the TauT performance in this chapter, I also took the opportunity to compare the fully trained TauT against the default DeepTau discriminants. Such a comparison was also part of [55], though with a reduced training dataset and a shorter limit on the number of training epochs.

3.4.1 Training Setup

For this study, I compared four sets of discriminants:

- **DeepTau V2.1**: the legacy iteration of the DeepTau architecture used in Run 2
- **DeepTau V2.5**: the current iteration of the DeepTau architecture used in Run 3
- **TauT partial**: a TauT discriminant that was trained with a 10%/10% training/-validation data split and three epochs of patience in the early stopping condition
- **TauT full**: a TauT discriminant that was trained with a 90%/10% training/validation data split and ten epochs of patience in the early stopping condition

Of these, the TauT partial setup was a rerun based on the settings of the original TauT publication [55] as outlined in Section 3.1.

The DeepTau WPs were defined based on thresholds calculated for $p_{\mathrm{T}} \in [30, 70]$ GeV, whereas I studied the discriminant performance over $p_{\mathrm{T}} \in [20, 100]$. Therefore, the DeepTau WPs did not perfectly align with their intended TPR on this test sample. To allow for a more even comparison between all discriminants, I interpolated the FPR of the DeepTau WPs from their ROC curves on this sample. As a result, the DeepTau FPR differ slightly from the values shown in other publications like [13].

3.4.2 Classifier Results

The ROC curves of the discriminants defined by the trained TauT and DeepTau NNs are shown in Figure 3.3 for the discrimination against electron, muon, and jet backgrounds.

On the most general level, the relative signal-to-data separation across the background classes was well in line with previous studies [13, 55, 58]. Discrimination against the muon background was the most efficient, followed by the discrimination against electrons, and finally against jets. Additionally, in the *vs.-electron* and *vs.-muon* cases, the TauT performance exceeded the DeepTau performance by a significant margin, whereas in the *vs.-jet* case, the performance was comparable. In all three cases, the full TauT training exceeded the performance of the partial TauT training.

In the comparison of the *vs.-electron* discriminants, the TauT models stand out with notable performance improvements. Especially at low TPR, the improvement was dramatic with more than 9× lower FPR for the full TauT discriminant compared to the

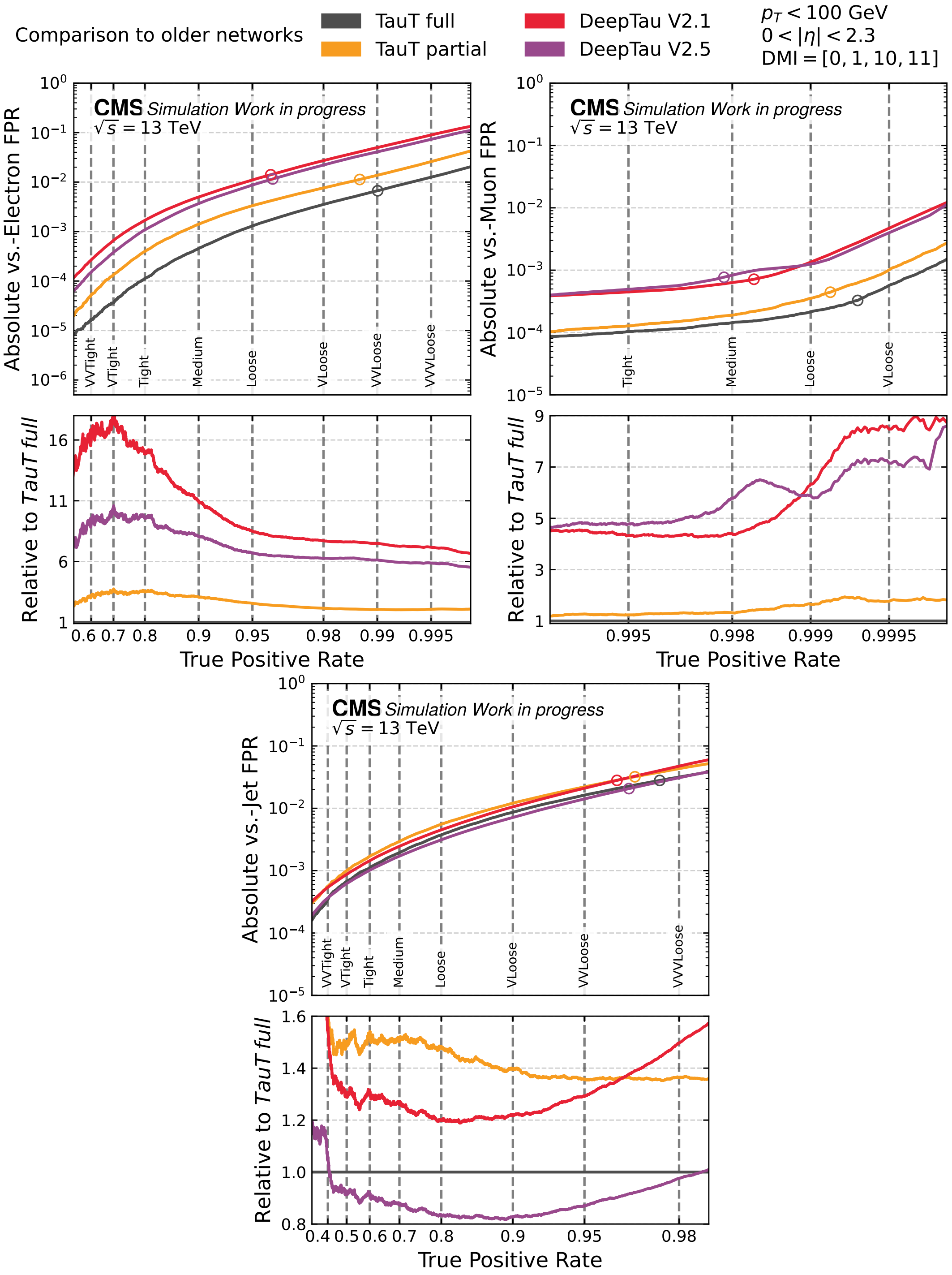


Figure 3.3: Comparison of absolute and relative classifier ROC performance against electron, muon, and jet backgrounds. The full-scale TauT training is shown in grey, a partial TauT training with the setting in line with previous TauT studies in [55] is shown in yellow, and the DeepTau V2.1 and DeepTau V2.5 performances are shown in red and purple, respectively. Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee point.

DeepTau V2.5 discriminant. This difference decreases at higher TPR, but even at the highest TPR, the TauT architecture achieves a 6× lower FPR.

For the comparison of the *vs.-muon* discriminants, the results were similar, though less pronounced. At lower TPR WPs, though still with more than 99%, the fully trained TauT discriminant showed a 4.8× lower FPR than the DeepTau V2.1 discriminant. Then, for exceedingly high TPR of more than 99.9%, the performance gap grows to a 7.1× and 8.5× lower FPR compared to the DeepTau V2.5 and DeepTau V2.1 discriminants, respectively.

Finally, in the comparison of the *vs.-jet* discriminants, the picture is mixed. At low and high TPR, the full TauT performance is similar to the DeepTau V2.5 discriminant, with a near identical FPR, but at medium TPR the TauT is outperformed. For TPR around 80-90%, the fully trained TauT discriminant misidentified up to 25% more jets as hadronic taus as DeepTau V2.5.

3.4.3 Results Discussion

The comparison of the discriminants of DeepTau, and the partially and fully trained TauT highlights significant trade-offs between architectures and the training process. In this section, I focus on the direct comparison of the two architectures. The performance changes between the partial and fully trained TauT model due to the changes in data volume and patience are the subjects of Sections 3.5 and 3.6, respectively. It is worth noting that the volumes of training data for the DeepTau V2.5 and the TauT were nearly identical. Both relied on the S&M dataset, though the adversarial training step of the DeepTau V2.5 training procedure added 0.2% of training data relative to the S&M example volume, as described in Section 2.4.2.3.

vs.-Electron The gain in performance compared to the DeepTau discriminant is most significant for the *vs.-electron* case, where the FPR reduces by factors of 6 to 9. A combination of feature-level and architectural factors can explain this improvement. Firstly, as detailed in Section 2.4.3.2, additional input features were introduced to better characterize the candidate topology. RECO electrons, in particular, benefit from the larger feature-set, most of which relates to their shower shape in the ECAL. These provide relevant handles for distinguishing genuine electrons from hadronic τ_h activity [55].

Secondly, the TauT architecture captures long-range interactions between particle constituents, whereas the local structure of the DeepTau convolutional architecture limits them. This global sensitivity is especially valuable in pileup-dominated scenarios, where the absence of long-range interactions between particle constituents enables more accurate rejection.

Finally, the aggregation of particles within the same $\eta - \phi$ cell of the DeepTau grids imposes a granularity limit. Only the highest-p_T particle of each type per input cell is retained, discarding information from softer particles before they enter the NN training. This is particularly relevant for genuine τ_h, where neutral pions decay into two photons, the showers of which may overlap within the ECAL. The reduction to a single photon candidate in such cases removes part of the characteristic π^0 signature, weakening the network's rejection capability.

In addition to the overall reduction in FPR across the whole TPR scope, a more pronounced improvement in the high-purity regime around the **VTight** WP is observed in the ROC curve. This shift reflects a reduction in residual decision uncertainty as the *bend* of the segmented ROC shape is pushed towards higher TPR values.

With finer granularity and no information loss from per-cell particle reduction, the TauT network retains all of the relevant information for each candidate. As a result, it can assign higher confidence to genuine τ_h where DeepTau V2.5 is limited, especially for highly complex or nuanced candidates. This behavior is in line with the increased sensitivity observed in the variance study of Section 3.3.

vs.-Muon The gain in performance compared to the DeepTau discriminant in the *vs.-muon* case was less pronounced than for the *vs.-electron* case, though still high at $4.3 - 7.8\times$ lower FPR.

Some differences exist compared to the *vs.-electron* case: unlike the RECO electron features, RECO muon features were not significantly expanded on in the TauT architecture. In addition, the regime of *vs.-muon* discrimination was already highly efficient beforehand, with all target WPs having a TPR above 99% and a FPR below 1%.

The fact that the TauT discriminant reduced the background by half an order of magnitude in this region indicates that the outlined systematic limitations prevented the DeepTau architecture from achieving greater purity at this TPR. Still, most of the argumentation outlined for the *vs.-electron* case also holds for the *vs.-muon* case. Therefore, the significant performance improvement is reasonable.

While there was a notable difference in relative FPR against muons for the extremely high-TPR region around the **VLoose** WP, I attribute this mostly to characteristics of DeepTau and not an explicit advantage of the TauT architecture. Notably, the region had already shown higher than average variance in the stability study of Section 3.3.3, caused by the elevated slope.

vs.-Jet Unlike the other two discriminants, the *vs.-jet* discriminant of the TauT architecture did not consistently improve upon the existing DeepTau V2.5 discriminant, with an up to 25% higher FPR across the TPR spectrum. This can be understood by revisiting the TauT benefits, showing why they apply only partially to jet discrimination. Firstly, discrimination against jets primarily relies on hadronic PFcand constituents. While RECO electrons can appear in QCD jets, they constitute only a small fraction of the particles, while the majority consist of charged and neutral hadrons. Therefore, the extra particle information introduced as part of the TauT has little effect on *vs.-jet* FPR.

Secondly, long-range interactions are less relevant for high-multiplicity jet decays, as a jet inherently consists of multiple local substructures. As a result, the locality of the convolutional DeepTau architecture facilitates stepwise traceback of the jet decay.

Finally, the large multiplicity of particles in QCD jets increases robustness to missing or coarsely granular information. In contrast to the sparse particle patterns relevant for electron or muon rejection, jet discrimination benefits from statistical redundancy: even if some low-p_T constituents are lost during aggregation or suffer from coarse granularity, the overall shower shape and high-multiplicity structure are preserved. This reduces the potential advantage offered by the TauT architecture's improved granularity and long-range interactions for the discriminant.

Apart from fundamental changes to the TauT architecture, adjusting the loss function is a low-commitment option for future studies in this direction. As listed in Appendix A.4, focal terms with heterogeneous weighting are used in the DeepTau training, which allows the NN to focus on a specific background over others. A similar approach would allow the TauT to prioritize jet rejection over electron or muon rejection, though the impact of such focal terms is difficult to estimate beforehand. Such a step is not part of this thesis to retain an unbiased variant of the architecture, allowing direct comparisons of impact between backgrounds in the following ablation studies.

3.5 Training Data Scaling

This section examines the effects of data scaling, the primary architecture-independent scaling method for physics performance, on the TauT discriminant. This study was directly motivated by the difference in classification efficiency observed between the partially and fully trained TauT discriminants in Section 3.4. Data scaling is a well-known property of ML approaches in many fields [80–82], and multiple studies show empirically that an increase in training data results in a measurable increase in NN performance.

These empirical trends are often summarized in the form of scaling laws, typically expressed in a power-law form of the application loss:

$$L(N) \approx A \cdot N^{-\alpha} + L_\infty, \tag{3.2}$$

where $N = N_{abs}/N_0$ is the number of training examples N_{abs} relative to the starting number N_0, α is the scaling exponent, A is the scaling coefficient, and L_∞ is the irreducible error. Values for the scaling exponent vary significantly with the type of ML application, resulting in a relatively wide scaling range ($\alpha \in [0.05, 0.35]$) [81, 83, 84]. Part of this variance stems from the possibility of performance saturation, which depends heavily on the application's type and complexity.

The main downside of data scaling approaches is that large volumes of accurately labeled data are challenging to acquire. In HEP, such data is usually simulated to provide perfect certainty about the labeling, though this can introduce biases if the simulation and detector conditions diverge. The simulation of data is also computationally expensive, especially for the order-of-magnitude increases relevant to scaling laws. This aspect makes it all the more important to gain a proper understanding of the data scaling we can measure in our existing training data: high investment in compute capacity for orders of magnitude more training data requires a high level of confidence in performance gains to be sufficiently motivated.

3.5.1 Training Setup

The ten TauT training runs of this study varied only in the training data available to them. The training data was doubled until all available data was used, starting with an initial $2 \cdot 10^5$ examples and ending with the complete $8.7 \cdot 10^7$ examples. While the training data was randomly selected from the available pool, the validation data (10^7 examples) remained identical across all training runs. Of these ten runs, the final run with all of the training data available to it was identical to the full TauT training run shown in the comparison with DeepTau in Section 3.4.

As discussed in Section 3.3.3, the CCE training loss and the ROC efficiency metrics did not fully align. Therefore, I had to modify Equation (3.2) to directly reflect the HEP-relevant performance metric of FPR at WPs with fixed TPR:

$$\mathrm{FPR}_{\mathrm{WP}}(N) \approx A_{\mathrm{WP}} \cdot N^{-\alpha_{\mathrm{WP}}} + \mathrm{FPR}_{\infty,\mathrm{WP}}, \tag{3.3}$$

where the fit for each WP is performed independently. I employed a least-squares fit [85] for this purpose, and imposed a lower bound of 0 for the irreducible FPR $FPR_{\infty,WP}$, as a negative FPR would be unphysical.

3.5.2 Classifier Results

The ROC curves of the discriminants defined by the trained TauT are shown in Figure 3.4 for the discrimination against electron, muon, and jet backgrounds.

Relative Performance Changes The background rejection scaling with additional training data was visible across all three background classes. While the effect was not equally measurable across the WPs, rejection efficiency improved consistently across background categories and the observed TPR range.

For all backgrounds, the improvement was characterized by two visible features in the behavior of the absolute ROC curves: firstly, increases in the amount of training data available to the NN resulted in a vertical offset. Secondly, the increase in training data caused a horizontal shift in the curve's location, leading to a change in the *bend* at the transition between the ROC segments introduced in 3.3.3.

The effect was most visible with the *vs.-electron* discriminants, and especially with its tighter WPs in the TPR range of 60% to 80%. Here, decreases in the FPR of up to 38× were measured between the discriminants based on the least and most training data. This pronounced scaling subsided for higher TPR values, but even at the **VVVLoose** WP, the performance was improved by a factor of 9.

With the *vs.-muon* discriminants, the results showed a flipped progression, with the most pronounced impact measured for the loosest WP. The FPR was lowered by 3.9× for the **Tight** WP, while the difference in performance reached 14.3× at the **VLoose** WP.

Finally, for the *vs.-jet* discriminants, the general progression resembled the *vs.-electron* case, though the performance gains were more moderate. At the **VVTight** WP, the difference in performance was up to 4.4×, with the difference diminishing slightly for higher TPR to 3.6× at the **VVVLoose** WP.

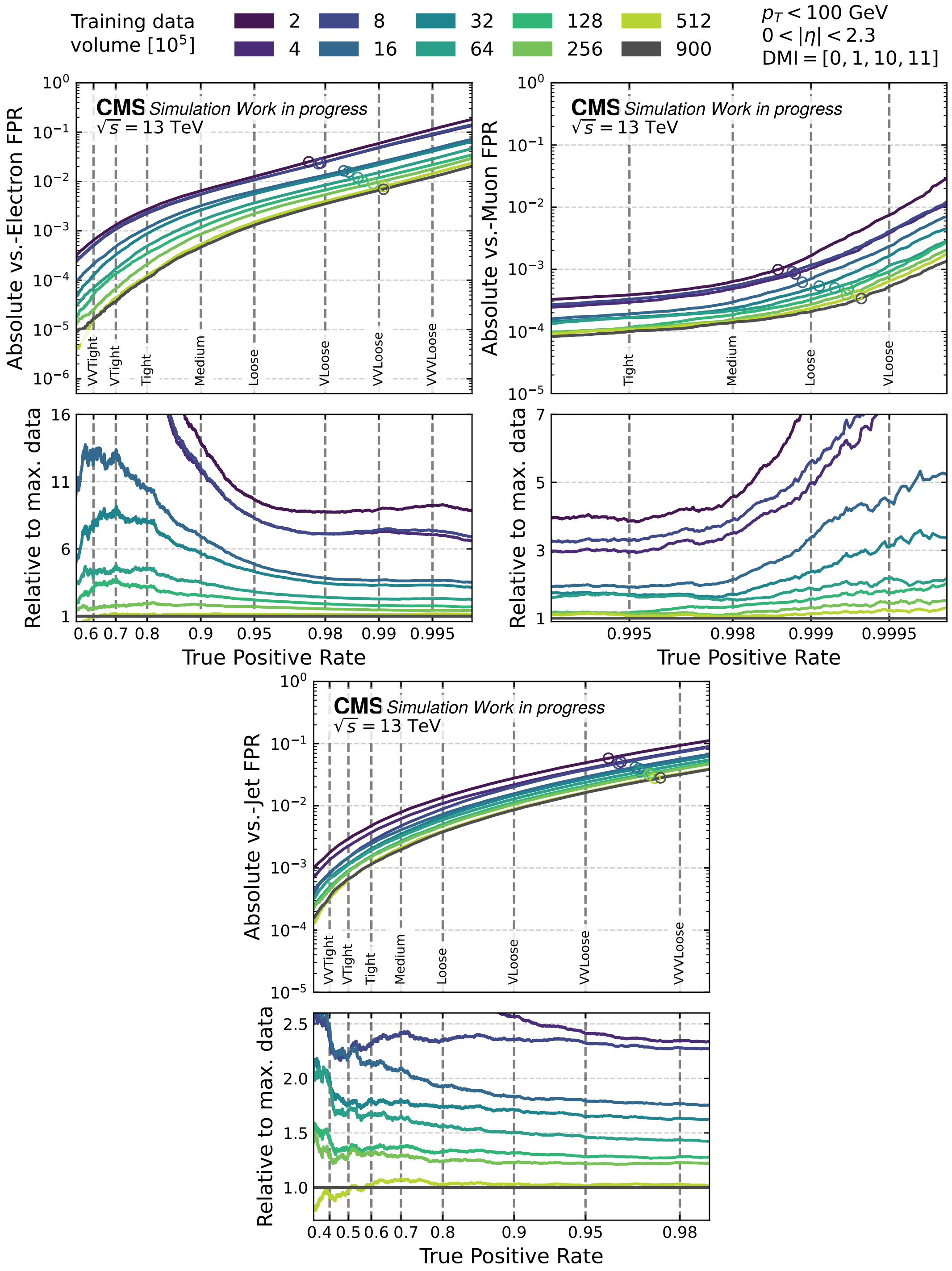


Figure 3.4: Comparison of absolute and relative classifier ROC performance against electron, muon, and jet backgrounds. The runs are shown in a gradient from green to blue as more training data becomes available, and the full-scale TauT training with all available data is shown in grey. Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee points.

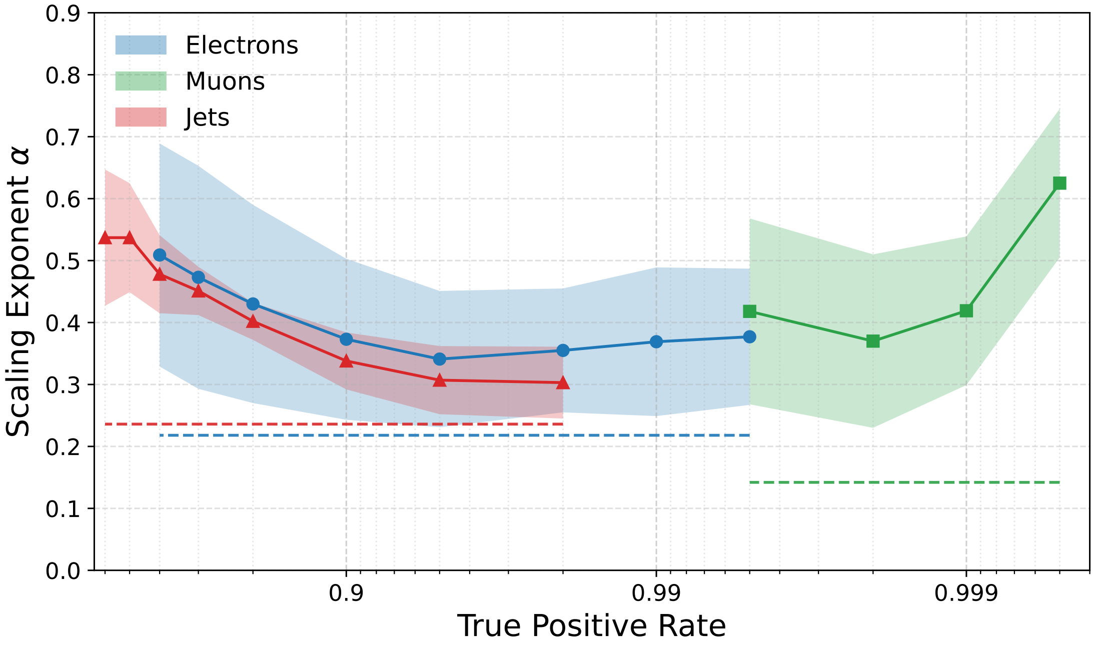


Figure 3.5: Scaling exponents of FPR power law fit according to Equation (3.3) for each WP and for each background category. Dashed lines show the FPR power law fit of the respective knee-points.

Power Scaling Fit The fit of the WP performance to the power law model to each of the WPs is detailed in Appendix A.7, and the fit results for the scaling exponents α_{WP} are shown in Figure 3.5. All fits were well-behaved, though the low absolute values of the *vs.-electron* WPs at low TPR and high training data volume resulted in significant relative uncertainties.

The resulting range of scaling exponents varied significantly across the TPR range. The *vs.-electron* and *vs.-jet* scaling factors matched well in their shared TPR range and the *vs.-muon* scaling is similar to the *vs.-electron* scaling in their shared WP. For the electron background, the exponents α_{WP} were located in the range of 0.341 ± 0.115 to 0.509 ± 0.177, following the trend observed with the relative ROC curves. The value was highest for the **VVTight** WP, then decreased around the **Medium** WP, and finally flattened out towards the **VVVLoose** WP.

A similar adherence to the relative ROC curve shape was observed for the other two background categories: For the exponent in the muon background rejection, values ranged from 0.370 ± 0.139 to 0.625 ± 0.124, rising with TPR. For jet background rejection, the picture was similar to that of the electron background, as the exponent was highest at the **VVTight** WP (0.537 ± 0.108) and then decreased towards higher TPR, down to 0.303 ± 0.058 at the **VVVLoose** WP.

It is worth mentioning that the irreducible FPR was inconsistent. For the *vs.-electron* WPs the values were compatible with zero within the fit uncertainty. For the *vs.-electron*

WPs, where the fitted irreducible FPR was of the same magnitude as uncertainty. This behavior is not strictly nonphysical, as a FPR of zero for the rejection of a specific background could be achieved with sufficient separation of signal and background data. Only for the *vs.-jet* WPs irreducible FPR significantly exceeded zero, which could imply a saturation of the *vs.-jet* classification. The fit therefore implies a potentially perfect separation between signal and background for the *vs.-electron* and *vs.-muon* cases as training data increases, but an imperfect separation for the *vs.-jet* case, even with greater data volumes.

Curve Shifting Figure 3.6 illustrates how the full-scale ROC curve is changed by shifts in TPR and FPR direction, and how it compares to the TauT variant, with minimal data volume. The plot shows that the topologies of the two discriminants are nearly identical, except for these shifts. The power-law fit results for the knee points in both the TPR and FPR directions are shown in Appendix A.7. The derived fit exponents for the FPR position of the knee point were consistently lower than the lowest scaling exponents of the WPs themselves: 0.217 (*vs.-electron*), 0.222 (*vs.-muon*), and 0.225 (*vs.-jet*). The exponents for the TPR were of similar size within the uncertainties: 0.218 (*vs.-electron*), 0.582 (*vs.-muon*), and 0.236 (*vs.-jet*).

3.5.3 Results Discussion

The data scaling study highlighted the substantial impact of training data scaling on discrimination performance for all three background categories. It showed two distinct, but complementary improvements that affected the performance. The vertical offset reflected an improvement in the background score distribution as the amount of training data increased. This effect was visible across the entire TPR range of the ROC curves and was not confined to specific WPs. The horizontal offset reflected an improvement in the distribution of the signal score. The impact of this TPR shift depended on the background category, as the slope topology between the initial and final TPR position of the WPs determined the resulting FPR change.

Figure 3.6 illustrates that the direct vertical offsets are the primary driver for relative performance gains in flat-slope regions. Furthermore, it shows that the horizontal offsets have a greater impact than the vertical offset in regions of greater slope, such as the **VVTight** WP for the electron and jet backgrounds, and the **VLoose** WP for the muon background. The two effects multiply, resulting in the elevated relative performance gains observed in this training data scaling study.

The variable scaling exponents of the power law fit in Figure 3.5 suggest that a single exponent cannot fully describe the combined TPR and FPR shifts across the TPR range. I therefore propose an expanded power-law model that accounts for shifts along both the TPR and FPR axes. Based on the results of the power law fit for the knee points, both the FPR and TPR position (TPR_K and FPR_K) can be expressed as a power law:

$$1 - \mathrm{TPR}_K(N) = (1 - \mathrm{TPR}_{K,0}) \cdot N^{-\alpha_\mathrm{H}} \tag{3.4}$$

$$\mathrm{FPR}_K(N) = \mathrm{FPR}_{K,0} \cdot N^{-\alpha_\mathrm{V}} \tag{3.5}$$

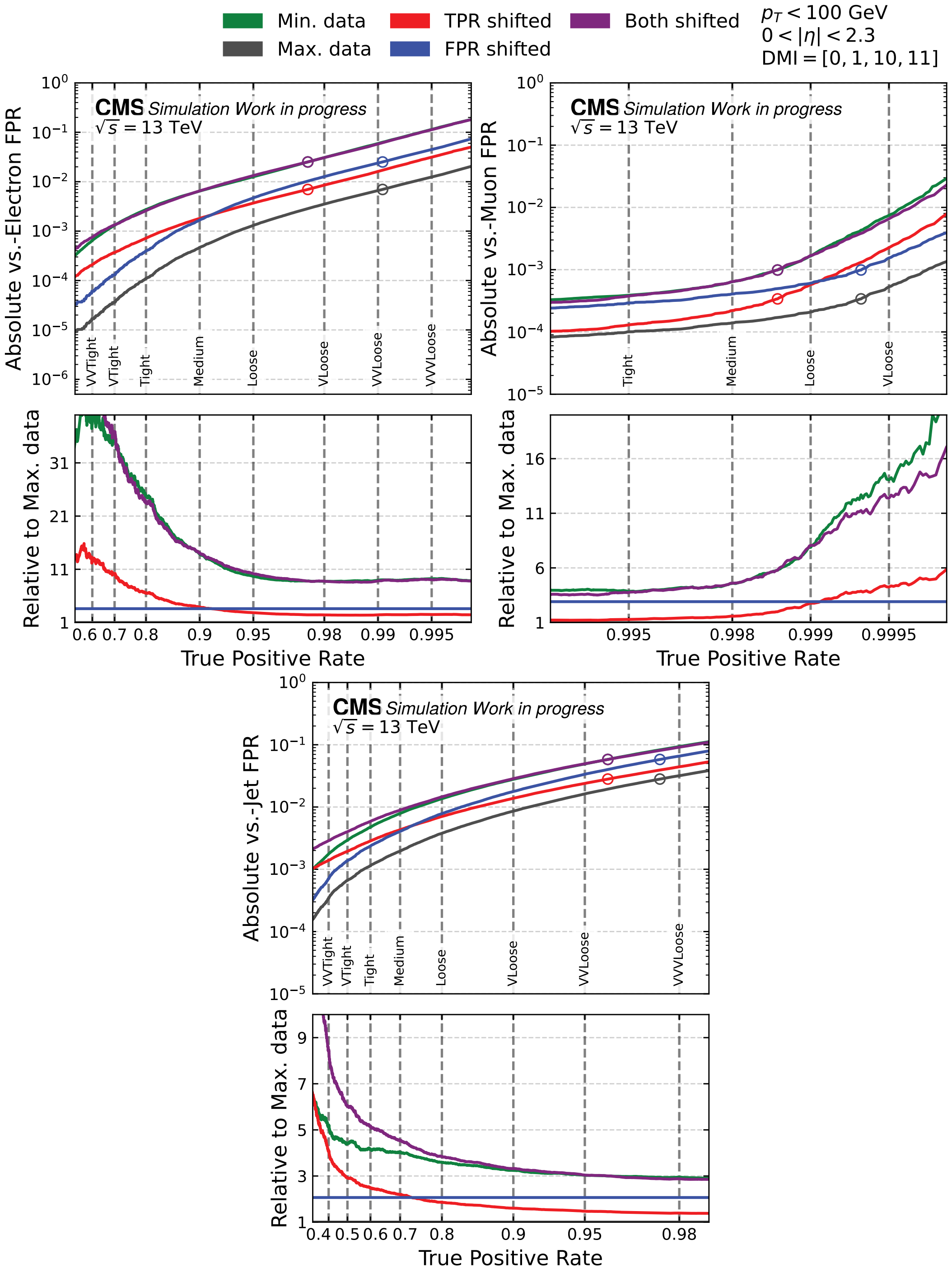


Figure 3.6: Comparison of absolute and relative classifier ROC performance against electron, muon, and jet backgrounds. The full-scale TauT training with all available training data in grey, and the TauT training with minimal data volume in green. The red, blue, and purple curves show the full-scale curve, shifted along the TPR and FPR axes by the difference between the full-scale and minimal-run knee points. Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee points.

where $\mathrm{TPR}_{K,0}$ and $\mathrm{FPR}_{K,0}$ are the baseline positions, and α_{H} and α_{V} are the respective constant scaling exponents. It is helpful to employ logarithmic scaling for both TPR and FPR at this point:

$$S = -\ln\left(1 - \mathrm{TPR}\right) \quad ; \quad K = \ln\left(\mathrm{FPR}\right), \tag{3.6}$$

transforming Equations (3.4) and (3.5) into

$$S_K(N) = \alpha_{\mathrm{H}} \ln\left(N\right) \underbrace{- \ln\left(1 - \mathrm{TPR}_{K,0}\right)}_{S_{K,0}} \tag{3.7}$$

$$K_K(N) = \alpha_{\mathrm{V}} \ln\left(N\right) \underbrace{+ \ln\left(\mathrm{FPR}_{K,0}\right)}_{K_{K,0}}. \tag{3.8}$$

Figure 3.6 suggests that the overall ROC topology remains approximately invariant, except at the extreme tails of the TPR/FPR ranges, even after shifts along the TPR and FPR axes. This allows for a shift along the original curve to determine the relevant FPR-value after the TPR-shift. Together with the TPR-independent definition of the knee point, and an additional shift in FPR Equation (3.7) generalizes to

$$K\left(N\right) = K_I(S_0 - \alpha_{\mathrm{H}} \ln\left(N\right)) - \alpha_{\mathrm{V}} \ln\left(N\right) \underbrace{- K_I\left(S_0\right) + K_0}_{K_C}, \tag{3.9}$$

where K_I is the initial ROC curve before shifts, and S_0 and K_0 are the initial TPR and FPR positions.

This form of data scaling does not allow for a general scaling exponent across all WPs of the ROC curve, as $K_I(S)$ is generally non-linear. The dynamic scaling exponent α_{WP} is determined from the combined horizontal and vertical terms:

$$\alpha_{\mathrm{WP}} = \frac{dK(N)}{d\ln(N)} = -\alpha_{\mathrm{H}} \frac{dK_I(S)}{dS} - \alpha_{\mathrm{V}}. \tag{3.10}$$

This exponent is consistent with all observations.

For a region of locally constant slope in $K - S$ space, $dK_I(S) = CS + \mathit{const.}$ the exponent simplifies to $\frac{dK(N)}{d\ln(N)} = -\alpha_{\mathrm{H}} C - \alpha_{\mathrm{V}}$. The horizontal contribution merges with the vertical contribution, as seen with high TPR *vs.-electron* WPs, like **VVVLoose**, or with low TPR *vs.-muon* WPs, such as **Tight**. Connected to this, regions with steep, but constant slopes, such as the **VVTight** WP in the *vs.-electron* case, or the **VLoose** WP in the *vs.-muon* case, show scaling exponents up to twice as high as in the flatter regions.

Regions where one slope transitions into another do not possess a static scaling exponent and are instead subject to a dynamic factor depending on the mean slope that the horizontal shift traverses. For any given shift, an effective scaling factor $\alpha_{\mathrm{WP,eff.}}$ can be computed as

$$\alpha_{\mathrm{WP,eff.}} = \frac{\Delta K}{\Delta \ln(N)} = -\alpha_{\mathrm{H}} \left[\frac{K_I(S)}{S}\right]_{S=S_0}^{S=S(\Delta \ln N)} - \alpha_{\mathrm{V}} \tag{3.11}$$

A fully analytical solution is challenging though, as the exact shape of the ROC curve is non-trivial.

Based on this model, the absence of sharp plateaus in the FPR fit data is understood: Instead of a sudden transition from one exponent to another, the exponents change gradually, both between the WPs and among the individual data points of those WPs.

Comparison to Other Fields A comparison of the measured scaling exponents with the scaling expected in other ML fields, such as computer vision or large language models, shows that the effective scaling is outstanding. The purely vertical scaling behavior is average, with values in the range of $0.217 - 0.255$, compared to the expected range of $0.05 - 0.35$ [81, 83, 84]. The additional effect of horizontal scaling pushes the effective scaling of the relevant WPs to the range of $0.303 - 0.625$, depending on the ROC curve topology they are derived from. Based on the scaling factors measured at the **Medium** WPs, increasing the training data by two orders of magnitude would lower the FPR by factors of 5.6, 5.5, and 8.0 for the electron, muon, and jet backgrounds, respectively.

3.6 Patience Scaling

Section 3.5 demonstrated that the volume of training data available to an NN training greatly influences the final discriminant performance. In this section, I examine how discriminant performance evolves during the tail-end of training.

Typical ML algorithms rely on gradient descent to reach their optimal performance after multiple epochs of training, in which the available data is reused until the performance of the NN converges. If training is stopped before convergence, performance will be suboptimal; training beyond convergence risks overtraining [86], which also reduces performance. *Early stopping* is a common approach to this problem: the validation loss at the end of an epoch is monitored, and once a pre-determined number of epochs without further improvement (patience) has passed, the training is concluded.

All training runs of the TauT architecture were performed with a patience of ten epochs. The progression of the validation loss for each of the training runs of Section 3.5 is shown in Figure 3.7. While I mainly focused on the discriminants' performance, the validation loss is a helpful surrogate performance metric that enabled a more straightforward comparison of the NN as a whole. The plot indicates that, in each run, most of the performance improvements occurred in the initial epochs, while the remaining epochs lead to smaller improvements.

On the one hand, the impact of additional training time on overall performance seems small, but on the other hand, training time is among the most cost-effective parameters to tune. Like the training data scaling of Section 3.5, computational cost is entirely located in the training step, while there is no cost during application time. Unlike data scaling, no additional effort is required to accommodate longer patience, as it only affects the number of epochs needed to complete the training.

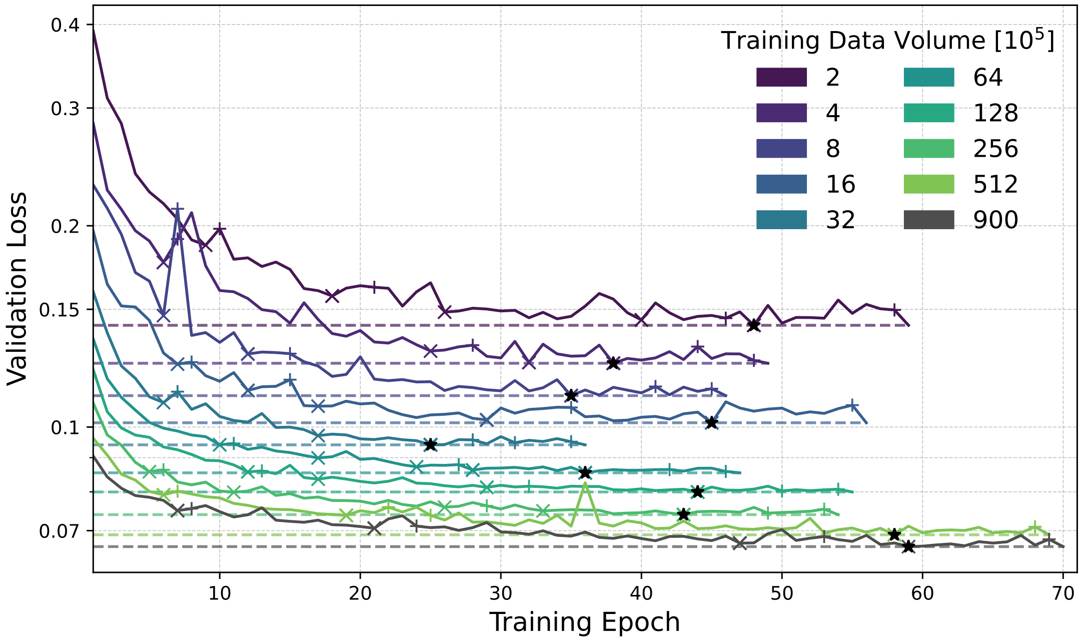


Figure 3.7: Validation loss progression throughout the training process for varying volumes of training data. Horizontal lines indicate the final validation loss. Black stars indicate the epoch at which that loss was achieved. Colored crosses indicate thresholds, where an increase in patience resulted in a lower final validation loss. Colored pluses indicate the final training epoch with 1, 3, 6, and 10 epochs of patience.

3.6.1 Training Setup

For this study, I did not perform any new TauT training runs but instead utilized the checkpoints of the fully trained TauT introduced in Section 3.4. I selected checkpoints based on the validation loss value that would have been reached for specific patience values. As shown in Figure 3.7, there were four such thresholds after 9, 25, 54, and 70 epochs, corresponding to patience values of 1, 3, 6, and 10. I evaluated the performance of the TauT at each of these breakpoints to quantify the impact of additional training epochs on the overall performance.

3.6.2 Classifier Results

The ROC curves of the discriminants defined by the respective TauT checkpoints are shown in Figure 3.8 for the discrimination against electron, muon, and jet backgrounds.

A difference in performance between the individual checkpoints was present, though less pronounced than the full range of the data scaling observed in Section 3.5.

With a patience reduced to a single epoch, some performance differences to the fully trained case with ten epochs of patience were measurable. As before, the effect was most pronounced in the *vs.-electron* case, with an 1.5× to 1.9× lower FPR. For both *vs.-muon*

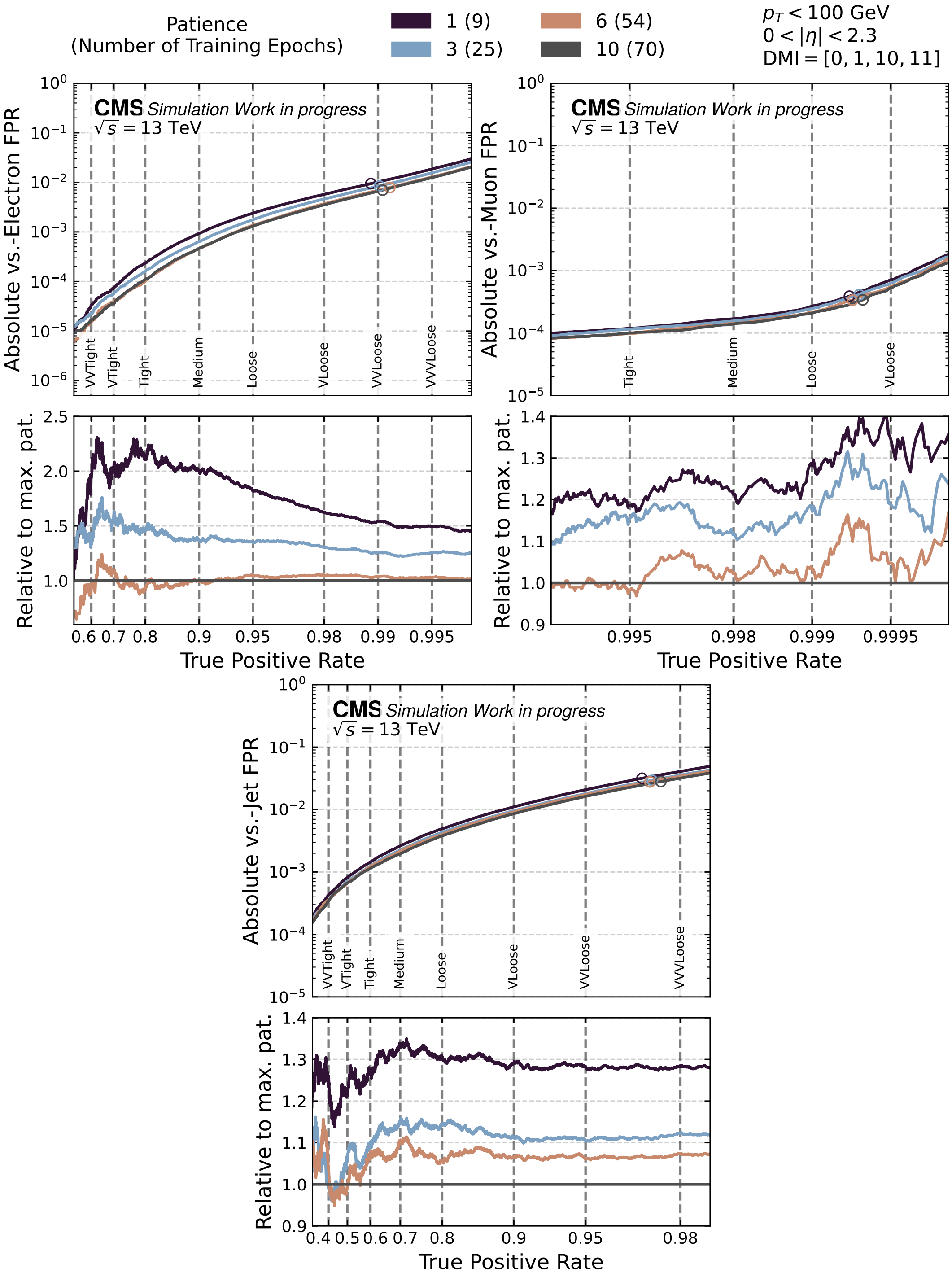


Figure 3.8: Comparison of absolute and relative classifier ROC performance against electron, muon, and jet backgrounds. The full-scale TauT training is shown in grey, and the performance of three TauT discriminants at reduced training patience are shown in black, beige, and blue, respectively. Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee points.

and *vs.-jet* discriminants, the effect was less impactful with the FPR lowered by 16% to 25%. Similar to data scaling, the TPR regions where the sensitivity was increased were around the **VTight** WP of the *vs.-electron* discriminants. In contrast, the *vs.-muon* did not exhibit any regions of exceptional sensitivity.

With roughly 13% of the total number of training epochs, 52% to 65% of the *vs.-electron*, 75% to 84% of the *vs.-muon*, and 76% to 82% of the *vs.-jet* performance was reached. The remaining performance was then achieved over the remaining 87% of the training epochs, with roughly half of the remaining performance difference achieved after 36% of the maximum 70 training epochs.

3.6.3 Results Discussion

The results of the study showed a small but measurable impact of patience beyond a single epoch on the discriminant performance. In regions of consistent ROC slope, this effect was further reduced to a 50%, 20%, and 30% increased FPR for the electron, muon, and jet backgrounds, respectively. While this effect was small compared to the impact of data scaling observed in Section 3.5, it remains relevant for optimal physics performance.

By evaluating the observed performance differences from the obtained data-scaling law (Eq. 3.3 and Table A.23[4]), I estimate that the gain from extended patience corresponds to a 1.5×-3× increase in effective training data. The increase in the number of training epochs from 9 to 70, therefore, compares poorly with an increase in training data by the same factor, since both affect the computational training effort linearly. Intermediate patience values showed similar ratios, as their performance improvements were offset by the additional training epochs required.

These results imply that the majority of TauT's performance was achieved in the initial few epochs, while the remaining training epochs improved performance only marginally. Especially near the tail end of the training process, changes to the ROC curve were of the same order of magnitude as the 68% CL derived in Section 3.3.

These findings lead to three key considerations when selecting an appropriate patience parameter:

- Data scaling outperforms increases in patience significantly
- Patience increases do not require any additional production steps, as additional training data does
- Maximum performance for any NN can only be reached with sufficient training data and patience

From these points, I conclude that if the training data is already available or relatively easy to obtain, increasing the training data is preferable to prolonged patience during training. If this is not the case, patience offers a less potent but available route to

[4] Scaling was compared at WPs of consistent slope: **VVVLoose** for the electron and jet backgrounds, **Tight** for the muon background

maximize performance without changes to the architecture. Finally, while lower patience values might be preferable for quicker turnaround, a finalized NN intended for use at scale, as DeepTau is, should be trained with sufficient patience to converge fully.

3.7 Token Pruning

My studies in this chapter have probed the properties of the TauT architecture with respect to stability and scaling. In this final study of the TauT physics performance, I examined the relevance of complete particle constituent information, a transformer-specific architectural property. Throughout this section, I use the ML term *tokens* to describe the particle constituents that define the involved PFcands, RECO, and global information of an event.

Unlike classical architectures, such as the convolutional approach used in DeepTau, transformers allow for a dynamic number of tokens. This property is convenient for use in HEP, as the number of reconstructed PFcands in a particle collision can vary. The available event information can be presented compactly and with minimal zero-padding, unlike the often sparsely populated approaches used with other architectures.

A significant downside of such a dynamic approach is that the resource requirements during training and application directly depend on the maximum number of tokens among the candidates. Although the size-binned batching described in Section 2.4.3.3 reduces the necessary zero-padding to a minimum, candidates with many tokens still limit the possible batch size. Due to the dense NN layers in transformer-type architectures, increasing the number of tokens results in a quadratic increase in memory requirements. In many cases, this property clashes with the often-tight VRAM limitations of GPU resources, leading to underutilization of hardware.

One approach to this problem is to standardize the number of tokens across all examples [87]. The idea of token pruning originates primarily from the field of vision transformers [88], which share many conceptual similarities with the TauT architecture. Quantifying the impact of such pruning is inherently challenging, as it does not remove entire examples but instead selectively modifies each example's token information.

Unlike the training data and patience scaling in Sections 3.5 and 3.6, modifying the token data affects the computational cost during application. With both training and application, a larger number of allowed tokens resulted in a decreased maximum batch size. Changes to the batch size can have widely varying computational performance impacts, depending on the hardware and the GPU's initial saturation. Consequently, while a larger token count may increase the discriminant's accuracy, it also introduces limitations on throughput and hardware efficiency.

3.7.1 Training Setup

Figure 3.9 shows the distribution of token counts in the S&M dataset. The number of tokens varies widely, with a median of 85 and a maximum of nearly 500. Most tokens originate from PFcands, with RECO electrons and muons contributing only small fractions. The mean number per category is 88.1, 0.5, and 0.4 for PFcands, RECO

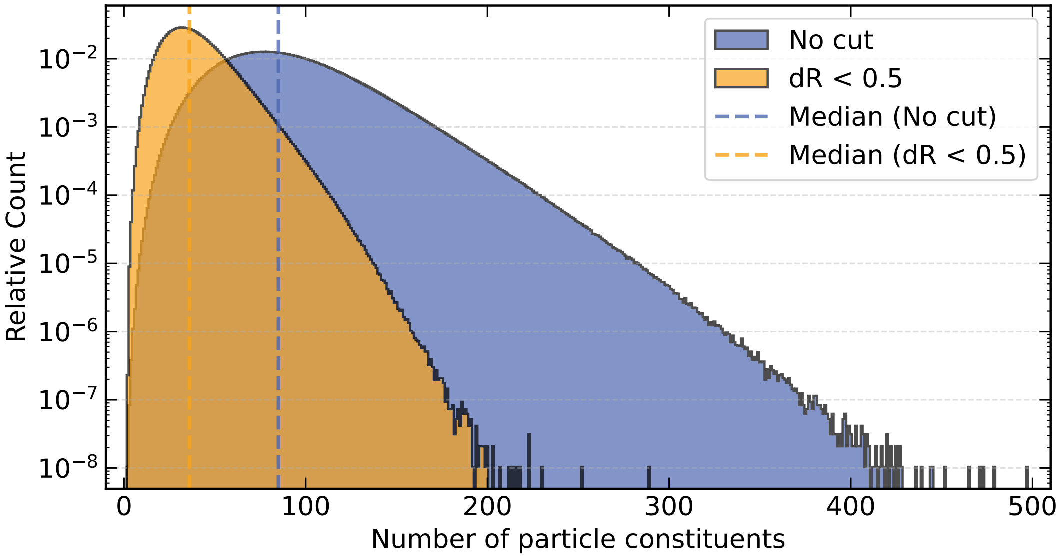


Figure 3.9: Distribution of token counts among the full training dataset, with (orange) and without (blue) a cut on $\Delta R < 0.5$. Dashed lines indicate the medians of the respective distributions.

muons, and RECO electrons, respectively, while the global detector information always contributes one pseudo-particle.

In this study, I investigated two approaches to token reduction:

- **Feature-based pruning**: Cut tokens, based on a predefined cut to a physical variable
- **Random pruning**: Randomly discard tokens from input categories that exceed a threshold

Feature-Based Pruning The first method of token reduction was proposed in the original TauT study [55] and reduced the number of tokens by applying a $\Delta R < 0.5$ cut in $\eta - \phi$ space to each token. As a result, the median number of tokens was cut from 85 to 36, as shown in Figure 3.9.

This approach was motivated by both the HPS algorithm and the original DeepTau architecture, which apply a similar ΔR cut. It suppressed distant tokens, which were presumed to contribute little to the classification performance. Previous studies of the architecture showed that more stringent cuts to this variable affected the performance and determined $\Delta R < 0.5$ to be an ideal threshold [55].

In this study, I re-examined these findings, as the results of Section 3.4 could indicate a performance degradation relative to a discriminant trained without such a cut. There DeepTau V2.5 achieved a significantly better FPR in regions of high TPR, while towards low TPR, this gap was reduced.

Random Pruning The second method served as an ablation study to isolate the effect of token count reduction itself, independent of any physically motivated selection. A limit was imposed on each of the four token categories (PFcand, RECO e/μ, global detector information). If the number of tokens in a category exceeded this threshold, tokens were randomly removed from this category until the threshold was met.

Due to the distribution of tokens across the input categories, this effectively meant that only PFcand input information was ever pruned, as it contributed 97.8% of all tokens. In contrast to an entirely random pruning selection, this approach prevented the already sparse non-PFcand tokens from being wholly removed [5]. Beginning at a threshold of 16 tokens per input class, I doubled the allowed token count until 256, after which subsequent doublings resulted in no tokens being cut.

In terms of how many candidates had at least one PFcand removed due to this, while the loosest threshold at 256 tokens affected 0.5% of examples, thresholds at 128 and 64 affected 12% and 74%, respectively. For even tighter cuts, nearly all candidates had at least one token removed (98% for 32 and 99.9% for 16).

3.7.2 Classifier Results

The ROC curves of the discriminants defined by the trained TauT are shown in Figure 3.10 for the discrimination against electron, muon, and jet backgrounds.

Feature-Based Pruning Across the *vs.-electron* and *vs.-muon* backgrounds, the discriminant trained with the token cut on ΔR showed nearly identical performance to the baseline with no tokens removed. For the *vs.-jet* case, especially at high TPR WPs, performance diverged significantly, with an up to 2.8× higher FPR compared to the discriminant with no cut.

I inspected this particular result in more detail, by separating the testing data along the DMs outlined in Table 2.1. The respective ROC curves are shown in Figures 3.11 to 3.13. [6] The *vs.-jet* curves show a strong separation in performance at low TPR among the DMs: The performance for the one-pronged DMs (DMI 0 and 1) was similar to the uncut discriminant, while the performance for the three-pronged decay modes (DMI 10 and 11) diverged heavily.

The effect was visible for DMI 10 of the *vs.-jet* discriminant, where the FPR was increased by up to a factor of 5.5, and DMI 11, where the FPR was increased by up to a factor of 3.8. This effect was not present with the *vs.-electron* discriminant, and for the *vs.-muon* case, the target TPR region was exceedingly loose and the population far too low to provide a clear picture.

Random Pruning Among the evaluated discriminants, the one with a maximum of 64 tokens showed clear signs of improper convergence, as its performance diverged in both

[5] In a candidate example with 300 PFcand tokens and 5 non-PFcand tokens, a completely random pruning to 32 would result in a 57.8% chance that all non-PFcand tokens are removed and a 91.5% chance that at least four are removed.

[6] Outside of the token pruning study, TauT ROC curves showed no divergent behavior across DMs.

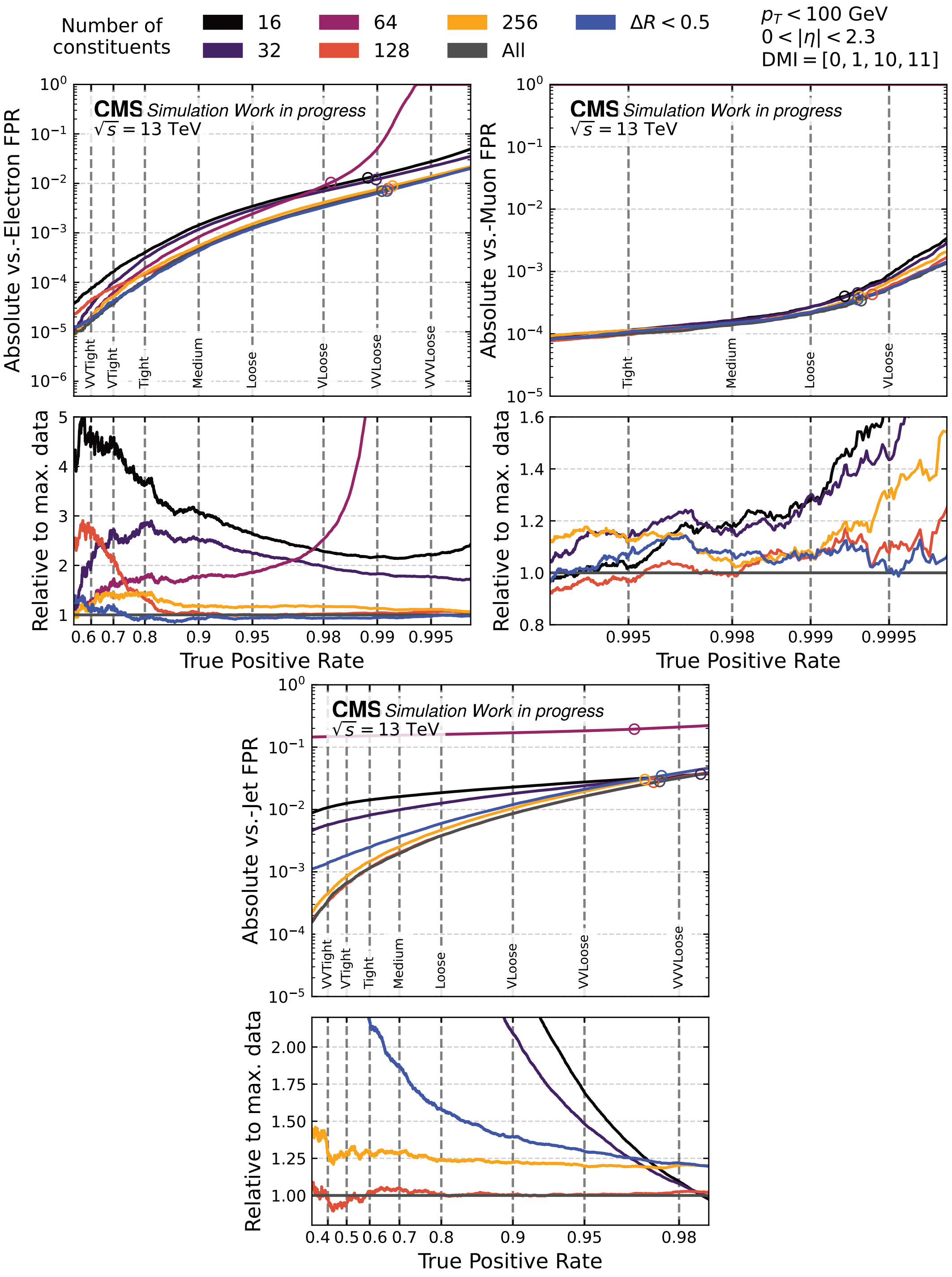


Figure 3.10: Comparison of absolute and relative classifier ROC performance against electron, muon, and jet backgrounds. The full-scale TauT training is shown in grey, discriminants with varying maximum tokens per class are shown in shades of magenta, and a classifier where a token cut of $\Delta R < 0.5$ was applied is shown in blue. Dashed vertical lines mark the target TPR of the respective DeepTau WPs. Circles on the ROC curve indicate the curves' knee points.

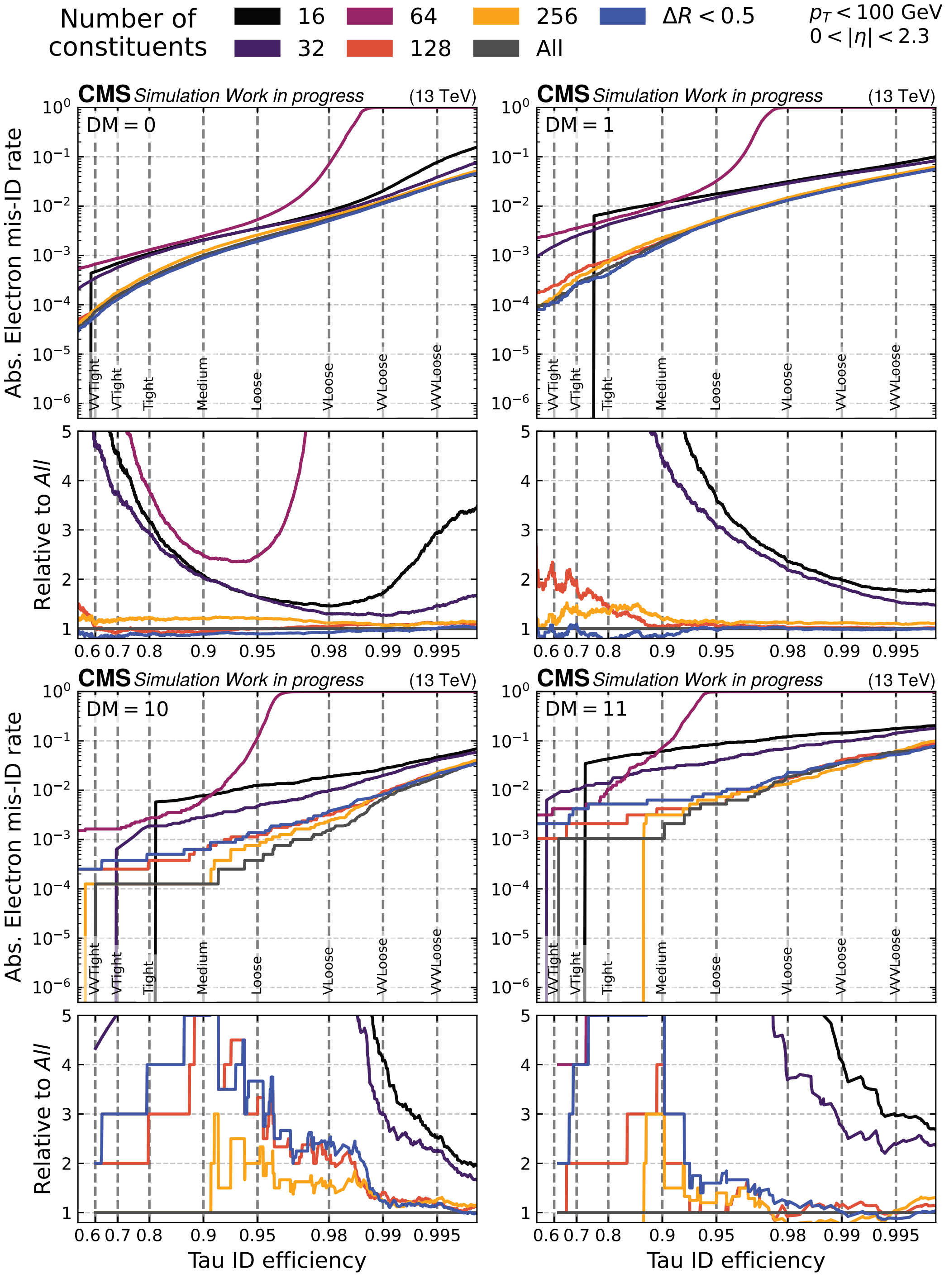


Figure 3.11: Comparison of absolute and relative classifier ROC performance against electron background with varying DMs. The full-scale TauT training is shown in grey, discriminants with varying maximum tokens per class are shown in shades of magenta, and a classifier where a token cut of $\Delta R < 0.5$ was applied is shown in blue. Dashed vertical lines mark the target TPR of the *vs.-electron* DeepTau WPs.

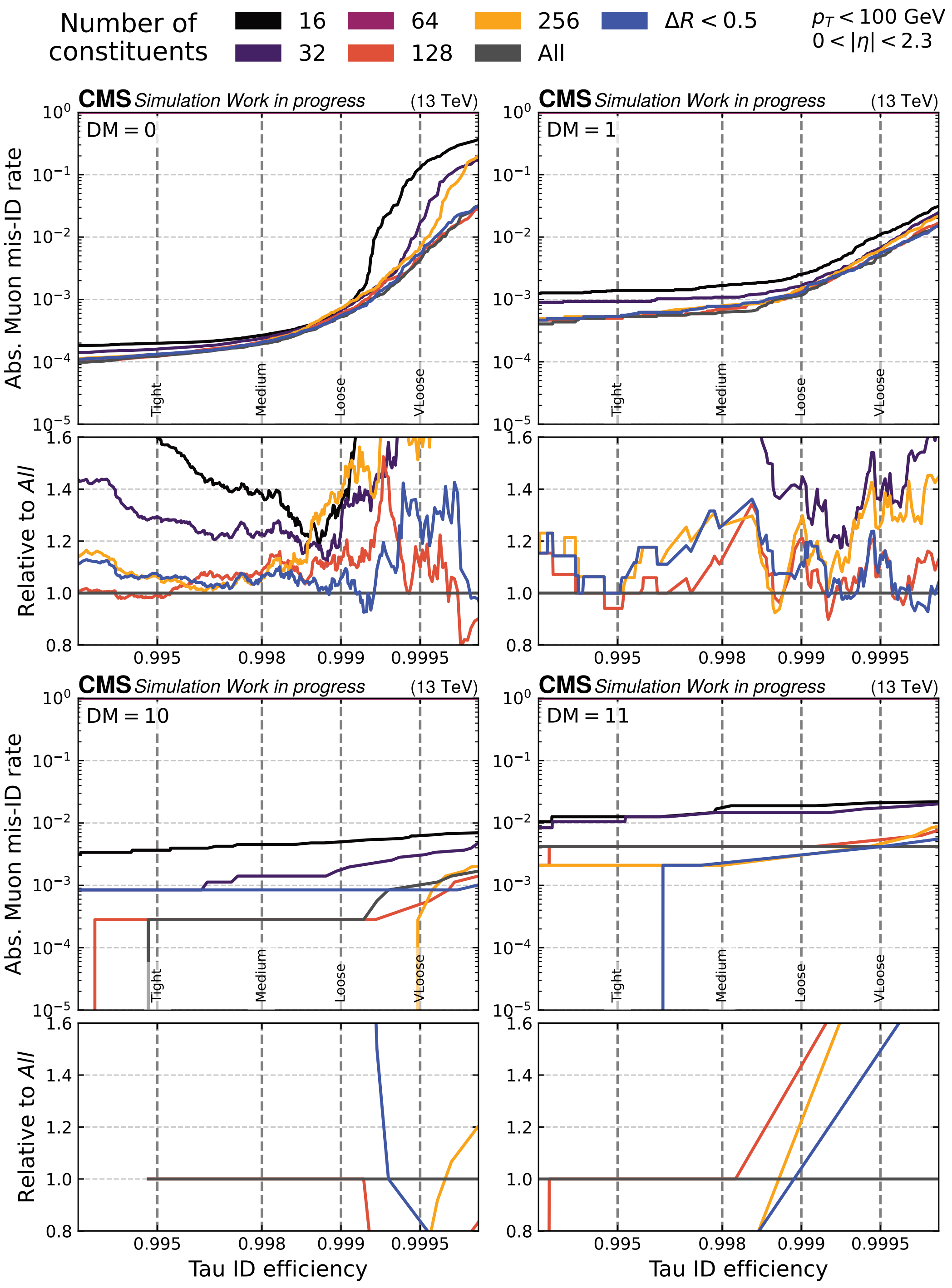


Figure 3.12: Comparison of absolute and relative classifier ROC performance against muon background with varying DMs. The full-scale TauT training is shown in grey, discriminants with varying maximum tokens per class are shown in shades of magenta, and a classifier where a token cut of $\Delta R < 0.5$ was applied is shown in blue. Dashed vertical lines mark the target TPR of the *vs.-muon* DeepTau WPs.

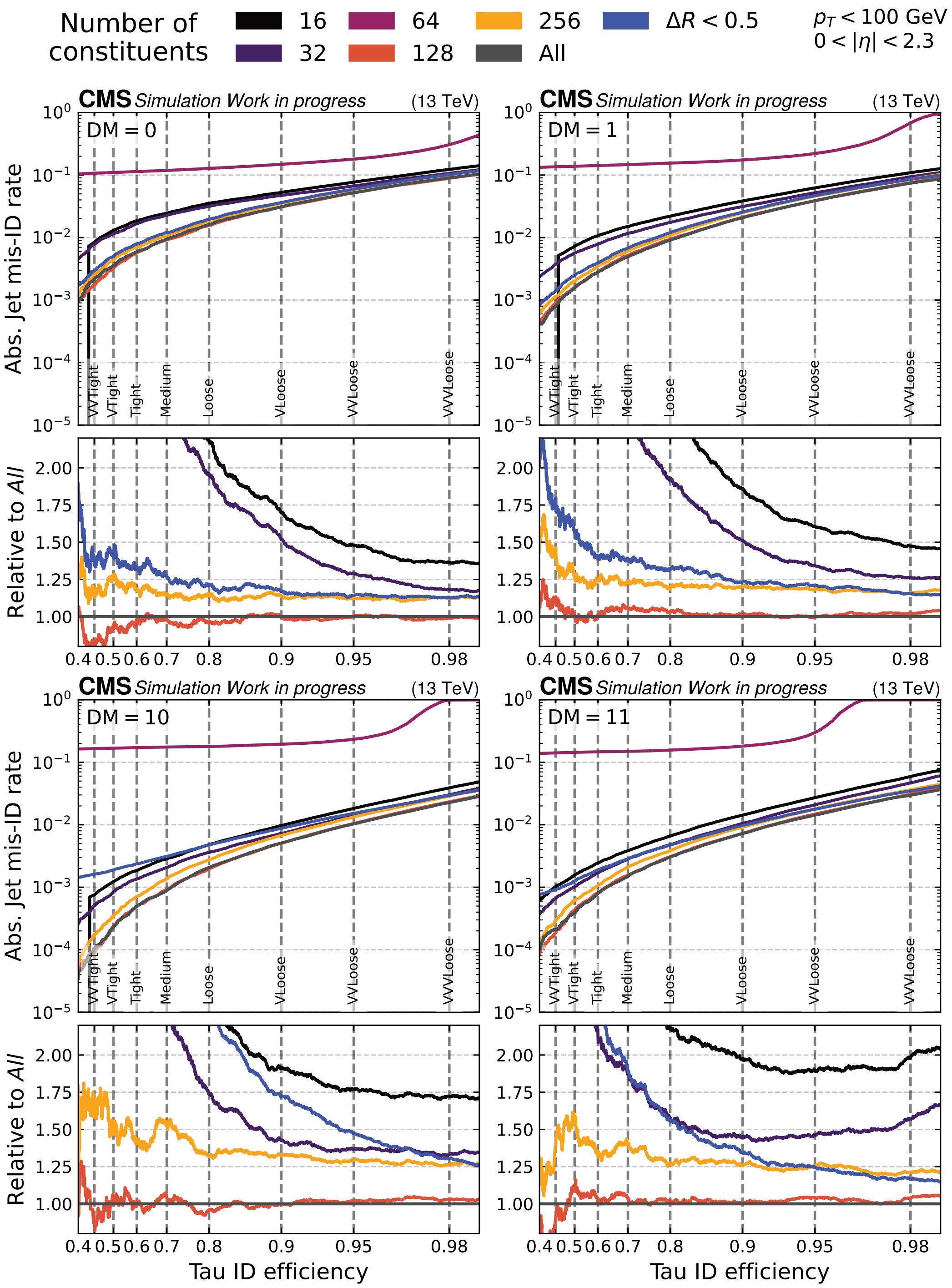


Figure 3.13: Comparison of absolute and relative classifier ROC performance against jet background with varying DMs. The full-scale TauT training is shown in grey, discriminants with varying maximum tokens per class are shown in shades of magenta, and a classifier where a token cut of $\Delta R < 0.5$ was applied is shown in blue. Dashed vertical lines mark the target TPR of the *vs.-jet* DeepTau WPs.

the high TPR *vs.-electron* region and across the entire *vs.-jet* spectrum. It is therefore not considered further in this study. For the discriminants with random pruning, the performance degradation at sufficiently low thresholds for the number of tokens was clearly visible across all three background classes.

The degradation in performance with a reduced numbers of tokens was most prominently visible for the *vs.-jet* case, where the FPR was increased by 19.0× for a threshold of 16 permitted tokens at the **VTight** WP. This difference gradually lowered for looser TPR, down to a difference of 1.1× at the **VVVLoose** WP. As the maximum number increased, the performance difference decreased, with a significant jump measured for the step from 32 to 64 maximum tokens. For 128 or more tokens, the performance difference was similar at high TPR, though some variance in performance remained around the **VVTight** WP.

The *vs.-electron* case also showed a consistent performance scaling with the token threshold across the relevant TPR range. While thresholds with more than 128 tokens maintained most of the discriminant's performance, tighter cuts greatly impacted the results. The FPR at the **VTight** WP was up to 4.4× higher than for the uncut discriminant, with the difference decreasing to 2.2× at the **VVLoose** WP.

In the *vs.-muon* case, performance degradation was also observed with lower token thresholds, though the difference was far less pronounced than for the *vs.-electron* case. The change in FPR at the **Tight** WP was at most 20% and rises towards the **VLoose** WP, up to 60%. Outside of the increased sensitivity to performance changes around the steep ROC slope at the **VLoose** WP, no notable features were visible.

3.7.3 Results Discussion

The token pruning study revealed physics performance-relevant details for both tested methods:

- For a physics-motivated cut to the tokens, significant degradation for specific subsamples of the test data was found.
- For the random cuts based on pre-selected thresholds, performance degradation was measured across the board, though comparably high thresholds preserved most of the discrimination power.

Feature-Based Pruning The token-pruning based on features resulted in a significant decrease in the median number of tokens per candidate, from 85 to 36. However, the long-tailed token distribution remained the main bottleneck, as shown in Figure 3.9. The distribution of tokens per candidate still showed a significant tail towards high values, up to 200. Many tokens from candidates with fewer than 200 tokens were also cut, with little impact on computational limitations. Therefore, this approach is computationally comparable with an upper threshold of roughly 200 tokens across the candidate input categories.

In terms of physics performance, this approach aligned with that of the TauT classifier trained with unchanged data outside of the *vs.-jet* case. The main area of divergence, the low TPR region of the *vs.-jet* case, is dominated by candidates that require high confidence thresholds to be included in analyses. Performance degradation in this region implies that removing high-ΔR tokens reduced the interpretability of more ambiguous candidates.

A look at the ROC curves split by the simulated DMs of the τ_h in the Figures 3.11 to 3.13 points towards three-pronged decays as the major source of the increase in FPR for *vs.-jet* case. The discrepancy between the DMs, points to a physical rather than a statistical motivation. Because the rate of three-pronged τ_h is lower than that of one-pronged τ_h, the absolute number of misidentified candidates is also expected to be lower. Instead, after the cut to ΔR, misidentification of this DM increases by close to an order of magnitude and dominates the overall FPR of the *vs.-jet* case.

Values of $\Delta R > 0.5$ imply that a constituent particle is not directly involved in the jet of a candidate, as τ_h jets are generally tightly collimated. Still, excluding such particles entirely removes information about the statistically unlikely events where τ_h particle constituents do end up outside of the $\Delta R < 0.5$ cone. Additionally, a cut on a specific ΔR value is likely to bias discrimination, since the topologies of the three backgrounds differ. Electron and muon backgrounds are tightly collimated, but the jet background, and especially the background originating from gluon jets, can have non-negligible energy deposits at wider distribution [89]. Similarly, while unlikely, decay products from τ_h decays may end up outside of the $\Delta R < 0.5$ cut for the wider, three-pronged DMs.

The information lost in this manner is compatible with my observations. On the one hand, one-pronged τ_h decays are usually tightly collimated, and particles outside of the $\Delta R < 0.5$ cone are highly unlikely to be part of the object. On the other hand, three-pronged decays are wider, involve more particles, and therefore have a higher chance of involving final state particles outside this cone.

For both the *vs.-electron* and *vs.-muon* cases, the absence of such discrepancies is mainly explained by the structure of their backgrounds. The probability for such an event to be classified as a three-pronged τ_h is already tiny. Even if three-pronged decays were reconstructed at an elevated rate, the impact on the collective rate vanishes. It is therefore reasonable that a further increase in cone width had little effect on the FPR of the muon background.

Random Pruning The main result of the random pruning approach is that the heavy dependence of the *vs.-jet* discriminant became visible. For the most extreme cuts to the number of tokens, performance worsened by nearly two orders of magnitude. In contrast to feature-based pruning, this differences in performance was independent of the DMs.

The decrease in performance was most pronounced towards lower TPR, while performance towards higher TPR was mostly retained. Such behavior implies that this background relied heavily on the detailed information provided by the PFcands information to classify the more complex candidates. This result fits well with the original DeepTau assumptions that a detailed description of the candidate surroundings is necessary for an accurate classification against jets.

At the same time, this result puts into question how much of the context can be reasonably removed in the name of computational efficiency. For the *vs.-jet* scenarios with token cuts of 128 or higher, performance was mostly retained. This corresponds to the majority of candidates remaining unchanged while only the more extreme outliers receiving modification.

In the case, performance was degraded, though the effect was less pronounced compared to the changes in *vs.-jet* performance. This can be seen as the electron discriminant relying on jet information as an exclusion criteria, as the expected electron signature is mostly of electromagnetic nature. Similar to the *vs.-jet* case, performance was mostly restored with a token cut of 128 or higher.

Finally, in the *vs.-muon* case, the performance was least affected by the change in token limits. While some effect was measurable, the influence was a magnitude smaller than even the effect on the *vs.-electron* case. This can be explained by the high certainty of muon reconstruction in the CMS detector and the exceedingly small probability for non-leptonic PFcand information to be related to background muons.

From these results I conclude that random token pruning affects the *vs.-jet* stronger than both the *vs.-electron*, and *vs.-muon* cases, as it predominantly affects the mixture of input categories most crucial for jet discrimination: Both electron and muon backgrounds have handles to retain classification ability without complete PFcand information, in the form of RECO electrons and muons. The jet background does not.

My recommendation is therefore that a selected token threshold should leave a majority of the data unmodified, though even then, increased variance might hinder model convergence. On the one hand, this random-pruning method offers a significant advantage over the feature-based pruning approach, as it is less likely to introduce biases toward specific subsets of the candidate data. On the other hand, a well-selected token cut based on prior physics knowledge might achieve better performance than a random selection with the same number of tokens involved.

It should be noted that the feature-based cut and random pruning based on a token threshold can be combined. Similarly, there is the option to prune tokens, not with a fixed cut of a feature, but by sorting them by a metric and removing tokens beyond a threshold of this metric. Both options have the potential to improve pruning while retaining performance, though they also keep the risk of bias seen with the feature-based approach.

3.8 Summary

This chapter presented my study of the attention-based hadronic tau classifier (TauT), extending the research of the architecture and comparing an improved version to the established DeepTau discriminants. The focus was on evaluating the training variance and performing ablation studies of architecture-independent scaling behavior. The impacts of training data volume, early stopping patience, and multiple token reduction schemes were studied. All results were measured using physics-relevant metrics, such as ROC curves and background rejection efficiencies at selected WPs.

The TauT architecture demonstrated low variance across independent training runs using randomized, but statistically identical, subsets of the main training dataset and random weight initialization. Observed instabilities remained mostly below 6%, relative to the mean performance, confirming the reproducibility and generalization of the NN approach. Residual instabilities were linked to a misalignment between the training loss and physics-applicable WP properties, where the loss did not fully capture the relevant physics performance behavior. Methods to improve this loss-performance alignment were proposed.

Regions of higher variance were found to coincide with high ROC curvature, motivating an extended power law model based on shifts along both the FPR and TPR axes. The properties of this model were validated alongside the ablation studies and were found to be consistent with the measured data.

A direct comparison with the DeepTau discriminant revealed significant performance improvements for the *vs.-electron* (*vs.-muon*) backgrounds, with FPR reductions of up to 9× (7×). Performance against the *vs.-jet* background was largely retained, with matching performance at very high and very low TPR and at most a 25% increase in FPR at medium TPR WPs. Contributing factors to the observed background-specific behavior were discussed from both architectural and physics-motivated perspectives.

Ablation studies quantified the influence of non-architectural hyperparameters on the discriminant performance. These parameters were chosen for their known substantial impact on physics performance, while their contribution to computational cost during inference remains small.

The data-scaling study showed effective scaling exponents above 0.3 for all backgrounds. Scaling effects were further enhanced in regions of steep slopes in the ROC, caused by shifts along the TPR axis. Fitted exponents in those regions reached up to 0.5, 0.6, and 0.55 for electron, muon, and jet backgrounds, respectively.

The results of the early stopping patience study yield comparably minor, yet computationally cheap, improvements in the discriminants, requiring no additional data processing. Observed improvements were equivalent to increases in training data of 1.5× to 3×, at the cost of an 8× longer training process.

Investigations of token pruning showed that physically motivated cuts to the input data can induce biases in the discriminant. For the tested $\Delta R < 0.5$ cut, the discrimination performance of three-pronged τ_h candidates against the jet background worsened towards low TPR, with up to a 5× increase in FPR. An alternative random-pruning approach did not exhibit this effect, indicating a physical rather than statistical origin.

For random pruning, performance deterioration was observed mainly with excessive token pruning, and the resulting magnitude depended strongly on the background class. In comparable regions, the *vs.-jet* case showed the strongest response, followed by the *vs.-electron* and *vs.-muon* cases, separated by roughly an order of magnitude each. Most of the performance was recovered for moderate pruning, where most training examples remained unaltered, though increased variance at training time remained.

This series of studies provides a comprehensive set of guidelines for the TauT architecture:

- Training data volume is a primary driver of performance improvement without architectural modification.
- Sufficient patience is required to reach optimal discriminant performance, though its impact remains smaller than that of data scaling.
- Token pruning can improve computational throughput, but methods must be physically well motivated to avoid bias and performance degradation.

Increases in training data volume, patience, and input token count each demand additional computational resources, just as extensions to the underlying NN architecture do. My results demonstrate that considerable performance gains can be achieved in a manner that efficiently scales to the massive volumes of physics data used in HEP analyses. The computational effort is concentrated on the training phase, while applying the discriminants remains inexpensive, despite the substantially higher efficiency.

CHAPTER 4

Computational Benchmark Background

In Chapter 3, I explored how training data volume and quality affect the performance of the hadronic Tau classifier TauT. On the one hand, classifiers benefit from additional training data and more extensive training procedures. On the other hand, these improvements come at the cost of increased computational demand, as training time increases.

These observations naturally extend to HEP computing as a whole, where the advances in precision are often achieved at the cost of increased computational efforts. The analogy extends further: the process of deriving training data for an ML effort follows the same steps as for typical HEP analyses. In the case of the TauT training, all steps from simulation to hadronic tau reconstruction are necessary. Likewise, the final product of both a HEP analysis and an NN training can be understood as condensed knowledge: a physics analysis yields a measurement with quantified uncertainties, while an NN training yields an NN model capable of classifying unseen data. Therefore, in the interest of both statistical precision and discriminant performance, the focus of this thesis shifts from physics performance to computational considerations in this chapter.

The HL-LHC phase, set to begin in 2030, promises a steep rise in instantaneous luminosity for the CMS experiment. On the physics side, such vast increases in event data improve the statistical precision needed to detect subtle effects and rare physics processes. On the computing side, such increased demand requires a scale-up of processing power, as each physics event requires reconstruction and results in additional simulation, analysis, and potentially ML efforts. Meeting this demand is not achievable by scaling the resources linearly: new hardware must provide higher efficiency, and algorithms must make effective use of it. Without significant advances, projections indicate that the HEP computing capacity will fall short during the HL-LHC era [8].

In the following chapters of this thesis, I focus on the role that GPU accelerators play in the current and future HEP computing landscape. I investigate how various CPU and GPU configurations, including CPU-only, single-GPU, multi-GPU, and partial-GPU, impact runtime, throughput, and energy efficiency across a wide range of HEP workloads. Through this, I aim to assess the viability of the setups and quantify the potential tradeoffs for GPUs at the global scale.

Section 4.1 of this chapter outlines the necessary HEP computing concepts, and Section 4.2 introduces TOpAS, the hardware cluster on which all benchmarks were performed. Section 4.3 discusses the benchmark assumptions of this thesis, and Section 4.4 introduces the measurement methods employed throughout the benchmarks. Finally, Section 4.5 concludes with an idle power draw study across all benchmark runs of this thesis.

4.1 HEP Computing Concepts

The field of modern HEP has become inseparable from large-scale computing, and the scale of the LHC and its experiments requires vast amounts of resources for processing, simulating, and analyzing event data. Between July 2024 and July 2025 alone, over a billion CPU hours were consumed in support of HEP research [90]. While already an impressive figure, it will rise further during the HL-LHC phase, when event rates increase by another factor of 3.75 compared to current Run 3 values [5].

Operation at this scale requires not only massive compute infrastructure but also efficient global workload distribution and careful planning of future computing architectures. Benchmarks play a central role in this process: they quantify how well hardware supports HEP workloads, guide procurement decisions, and motivate software optimization. As the role of heterogeneous hardware, such as GPU accelerators, grows, benchmarking also informs long-term strategy.

4.1.1 Hardware Benchmarking

Evaluating whether a software process is using hardware efficiently requires more than the CPU utilization of a single application. Low CPU utilization can stem from slow I/O, memory contention, or even misconfigured worker machines. Without controlled measurements, determining the cause of lagging performance is difficult at best and misleading at worst. Benchmarking workloads in a controlled environment is therefore essential. These workloads may be based on existing analyses or on artificial but representative tasks, and serve three key purposes: identifying bottlenecks, establishing baseline performance, and enabling safe testing of future workloads.

Traditional performance benchmarking has been focused on runtime and utilization assessments, with cost-per-compute considerations gaining significance in recent years. In particular, energy consumption has taken on a central role as it is both the primary contributor to operational costs and a significant factor in environmental considerations. As an example for the shifting focus in computing: while rankings like the Top500 list [91] focus on pure throughput, the Green500 list [92] was introduced to rank efficiency over brute-force performance. HEP as a whole is following this shift from maximum throughput towards more cost-efficiency motivated considerations out of necessity.

In HEP, representative benchmarks are crucial due to the scale of production workloads, including millions of reconstruction and simulation tasks running worldwide. The HEPix Benchmarking Working Group [93] focuses on providing well-defined and reproducible benchmark scenarios that cover the range of HEP-relevant production workloads. The HEP Benchmark Suite [94] provides a framework for running a variety of

HEP benchmarks and for gathering benchmark results, as well as hardware metadata in a centralized bookkeeping system.

The HEPScore23 benchmark [95] is the successor to the older HEP-SPEC06 benchmarks [96] and has been in use since April 2023. It covers the full range of HEP computing with workloads suggested by multiple experiments, including CMS, ATLAS, LHCb, ALICE, and Belle II. Benchmarks such as these are stable and trustworthy sources of information for all of HEP computing, enabling comparisons across different compute sites using a standard measure. One aspect that is still in development is benchmarks compatible with GPU resources. While some such benchmarks exist, there is no agreed-upon centralized GPU HEP benchmark as of writing.

4.1.2 The Use of GPU Hardware in HEP Computing

Historically, HEP computing has relied primarily on CPU-based resources. The mostly homogeneous CPU landscape allowed for predictable performance, straightforward scaling, and easy resource planning. In the past decade, GPU accelerators have emerged as complementary resources, offering three key advantages: lower upfront cost per unit of compute power, higher energy efficiency, and specialized features that accelerate ML.

Firstly, GPUs have evolved from their graphical origins into highly parallel compute engines optimized for data-intensive workloads. Unlike CPUs, which prioritize low-latency execution and complex instruction sets, GPUs maximize throughput through massive parallelism in a Single-Instruction Multiple-Data (SIMD) compute model [97]. GPUs employ kernels, functions written in GPU programming languages such as CUDA [98], which are executed in parallel by multiple threads. This approach allows each thread to operate independently on distinct data elements, while sharing the same kernel code. For example, an NVIDIA A100 GPU, launching 2^{10} threads per block and 2^{20} blocks per grid, can process over a billion elements in parallel [99]. This massive parallelism aligns naturally with the event-based approach in HEP, where each physics event can be processed independently. Early benchmarks of GPU throughput in the CMS high-level trigger farm demonstrated that GPU hardware can offer up to 2.8× the compute capacity of a CPU at the same price point [8].

Secondly, GPUs achieve higher energy efficiency than CPUs by trading latency for simpler cores and high-bandwidth memory, offering a higher number of operations per Watt. The Green500 [92] also reflects this: As of June 2025, the highest-ranking GPU-free machine is 99th, while the top-ranking system, JEDI, outperforms the best CPU-only system, PRIMEHPC FX1000, by a factor of 4.39 in energy efficiency.

Finally, the increasing use of ML in HEP further highlights GPUs' value. Hardware innovations such as Tensor Cores [100], which accelerated low-precision matrix operations, and support for sparse matrix multiplications [101], directly enhance ML workloads. This trend mirrors broader global developments in ML-driven science and industry, allowing HEP researchers to leverage specialized GPU features for both already existing and novel computational tasks.

The combination of throughput, energy efficiency, and ML-specific features has driven HEP computing towards a more heterogeneous hardware landscape, where both CPU and GPU resources are employed. As of writing, many local physics groups have access to GPUs, particularly for ML-based studies. While these small-scale resources facilitate rapid prototyping and ease of management, they are limited by uncertain usage patterns and restricted resource availability. Federated computing, as demonstrated during the inception of the Worldwide LHC Computing Grid (WLCG) [102], provides a scalable solution to this challenge by allowing shared access to costly resources.

One significant complexity arises from the fast-moving and heterogeneous nature of GPU architectures. Unlike CPUs, which are relatively standardized, GPUs differ across vendors, generations, and feature sets. Successful GPU utilization may depend on specialized capabilities, such as sufficient VRAM or dedicated tensor pipelines. Libraries like ALPAKA [103] provide unified interfaces between CPU and GPU architectures, achieving good performance across a range of hardware [104], though they fall short of addressing more fundamental limitations, such as VRAM boundaries.

Another HEP-specific challenge is the varying use of GPUs in the workloads. Heavyweight ML tasks require large amounts of VRAM and specialized pipelines, fully occupying GPUs, while lightweight workloads may underutilize large devices. Event-based processing is typically scalable and can be efficiently distributed across multiple resources, making cost-effectiveness the central concern for infrastructure planning.

The heterogeneous and rapidly evolving GPU landscape presents a challenge reminiscent of the early days of the WLCG: Similar to the past, efficient usage requires shared access to the expensive GPU hardware. Pilot projects like TOpAS (described in Section 4.2) have demonstrated the viability of shared GPU resources. Now, the following steps are to scale up GPU access and determine which HEP workloads benefit most from acceleration. An important step towards this goal is national facilities, such as German National High Performance Computing (NHR) centers like the Hochleistungsrechner Karlsruhe (HoreKa) cluster [105], which provide access to hundreds of high-performance GPUs, including high-end NVIDIA H200s. Research is ongoing to verify both the accessibility and the viability of GPU-accelerated HEP workloads and the usage of NHR centers.

4.1.3 Batch Systems

Whether it is local clusters for users of a single group, country-wide access to national research clusters, or even global access to the WLCG, management is necessary to automate which work runs where. Such management comes in the form of batch systems for HEP computing. They allow users to share large pools of hardware resources and orchestrate the execution of millions of jobs across thousands of worker nodes.

Originating in the 1950s, the term batch system originally referred to a system that accepted a batch of work requests, processed them without further human interaction, and returned the results [106]. At the time of punch card programming, these were significant advances in computer science, as they allowed computers to process work without human input. Since then, the general concept of batch systems has undergone great shifts as they have become increasingly dynamic and scalable.

The core idea of non-interactive jobs remains, although modern batch systems differ in many ways from their predecessors. Today, batch systems can asynchronously accept work requests (jobs) from multiple users and delegate them to a network of worker machines. Modern scheduling and bookkeeping enable multiple jobs to run in parallel on the same machine. The execution order is dynamically adjusted to maximize hardware utilization and ensure fairness between users.

There are multiple implementations of the batch system concept. I focus on the HTCondor [73] software in this thesis, as it is the system of choice for computing in the CMS collaboration. With HTCondor, batch jobs are submitted via a submission file that defines the workload, required hardware, and environment.

A job sent to the HTCondor scheduler first enters a system-wide job queue, where it waits to be matched. The scheduler then assigns the job to a worker machine within the distributed resource pool. Each of those machines might provide different amounts or types of computing hardware or require specific properties from a job to match. Once a match is found, the job is allowed to execute its workload on the matching machine, potentially concurrent with hundreds of other jobs. In addition to matching resources and executing jobs, such batch systems also need to enforce sufficient isolation between jobs to ensure fairness and stability.

Concurrent processing has come a long way since the inception of batch systems, and at present, many hardware resources are provided from a shared pool. Instead of a specific CPU thread, jobs are assigned an allowance of the total available CPU compute capacity on a machine. Similar approaches exist for other resources, such as disk or memory, but some non-standard resources remain non-sharable and must be assigned explicitly. Because of this communal sharing of resources between jobs, isolation between jobs is crucial. The presence of one job should never influence the result of any other job. Modern tools such as containerization and Linux kernel control groups [107] enforce these limitations by removing jobs that overreach their permissions or try to claim more resources than they originally requested.

Beyond resource isolation, HTCondor also offers a comprehensive set of customization options. Job preprocessing allows an HTCondor scheduler to deny or modify the request of a submitted job before it's executed. HTCondor also supports the definition of entirely custom resource categories for jobs that require non-standard features, like access to specific file systems.

Job slots advertise resources, but in many cases, one size does not fit all, and dynamic slot creation is necessary. HTCondor supports this via pooled resources that dynamic sub-slots can be split off of. Finally, HTCondor provides knobs to control which conditions apply to the matching criteria, beyond hardware availability. Resource overbooking is a widely used approach in this regard, allowing the system to advertise more resources than are physically available to balance out jobs which request more resources than they use.

Most of the computing performed as part of HEP is based on HTCondor and similar batch systems to ensure optimal utilization of the available hardware. Therefore, considerations for throughput-optimized processing for HEP workloads should assume usage within a batch system by default.

4.2 The TOpAS Cluster

The TOpAS cluster is a compute cluster focused on high throughput for HEP end-user analysis workflows, located near the Grid Computing Centre Karlsruhe (GridKa) compute center [108]. It consists of multiple CPU-only and GPU worker nodes, totaling 56 GPUs (8 NVIDIA V100 32GB, 24 NVIDIA V100S 32GB, and 24 NVIDIA A100 40GB [99, 109]). The cluster is set up to function as part of a local HTCondor batch system, while also allowing jobs from the WLCG to run opportunistically.

The benchmarks in this thesis ran on one of the TOpAS machines, which was taken out of service to prevent additional workloads from skewing the results. The node contains two AMD EPYC 7662 64-Core CPUs with Simultaneous Multithreading (SMT), resulting in a total of 128 real and 256 SMT threads. The EPYC CPUs support boost behavior, for which the clock frequency can be boosted from a base of 2 GHz to a maximum of 3.3 GHz. Whether the boost is active depends on the CPU's power draw, so during low multi-threaded utilization, single-threaded processes are accelerated. The node also provided 10 TB Nonvolatile Memory Express (NVME) storage, 1 TB Random-Access Memory (RAM), and a 100 Gb/s network connection. Crucially, the node possesses 8 NVIDIA A100 40GB GPUs connected via Peripheral Component Interconnect Express (PCIe). The addition of GPU resources to the baseline system increased the procurement cost of the compute node to 2.98× that of a comparable CPU-only node [110].

The Thermal Design Power (TDP) of the machine is dominated by the CPUs and GPUs with 2x225 W for the two EPYC 7662 CPUs and 8x250 W for the eight NVIDIA A100 GPUs. The H12DSG-O-CPU main board manufacturer does not provide a TDP value, but based on typical main board specifications, I estimate it requires at most 50 W. Other components with discreet TDP data included the disk and the network card, resulting in a combined total TDP of roughly 2560 W. This value is primarily intended for reference, as TDP is inconsistently defined and may vary across hardware manufacturers. While TDP does not adequately describe the power draw of a system under variable load, it does provide a reasonable estimate of the maximum thermal load an individual machine can handle.

4.3 Necessary Benchmark Assumptions

Experience has shown that the only fully representative benchmark is production itself, as no synthetic setup can fully reproduce every detail of real workloads and environments. Therefore, any benchmark that aims to extrapolate results from a controlled environment to a larger-scale setup must make assumptions.

4.3.1 Full System Utilization

For all benchmarks, the entire machine was filled with work. Instead of comparing single-process performance, I focused on the machine's throughput limit for each workload. This approach assumes that a throughput-focused cluster, such as TOpAS, would always receive sufficient work to keep it fully utilized. While this is a reasonable assumption in theory, real-world monitoring has shown it is not always the case. The assumption represents an optimistic upper bound: the performance achievable if resources are fully saturated at all times.

In a real-world scenario, the composition of jobs on our batch systems is unlikely to be completely homogeneous. In reality, a wide range of workloads runs on the worker machines, as the batch system's scheduling algorithm tries to maximize resource utilization. An example of this can be seen in the underutilization of CPU or GPU resources throughout the thesis: a setup focused on CPU-only performance will not utilize GPU resources at all, and GPU workloads tend to require less CPU compute capacity than is available.

4.3.2 Representation of a Batch System Job

While the benchmarks discussed in this thesis were mostly run directly on the machine, they are intended to represent work submitted as a batch job. As a result, some discrepancies between benchmark behavior and real-life applications might occur.

A benchmark running directly on a machine has access to the full breadth of the machine's compute resources, but batch systems may limit the resources a job can access. Large-scale jobs might fail to run on a machine that doesn't allow full use of its resources, but small-scale jobs can be impacted as well. Coarse resource granularity may result in jobs with low resource requirements occupying more hardware resources than they requested, thereby preventing other jobs from utilizing these unused resources. One prominent example of this is how HTCondor handles GPU resources by default: they are assigned to only one job at a time, regardless of their utilization.

The second way in which workloads outside a batch system can outperform batch jobs is through the safety margins users might add to their job requests. In the bare-metal scenario, the number of concurrent instances can be tuned to match available system resources closely. Batch jobs, on the other hand, typically add safety margins to the requested resources to guarantee their completion.

I considered the usability of all workloads used in this thesis with a batch system and believe they are generally suitable for such systems. The complete version of the *TauTransformer Training* workload from Chapter 5 was fully executed through the batch system as part of Chapter 3. The successful deployment of the necessary changes for the *MSSM Training* workload of Chapter 6 is detailed in Chapter 7. The *$gg \to t\bar{t}gg$ Generator* workload in Chapter 8 was not transferred to a batch system as part of this thesis, but other studies [111] of the underlying MadGraph4GPU software [112] report success.

4.3.3 Extrapolation of Throughput

Throughput is a key performance metric in benchmarks, but defining a single methodology for calculating average throughput across diverse scenarios is challenging. Typically, one would run a benchmark scenario, record the runtime and the amount of work performed, then compute the throughput as work divided by time. In the simplest scenario, concurrent instances of an identical workload would complete in the same runtime with negligible overhead. However, these assumptions rarely held across the experiments of this thesis.

Chapter 5 is the only chapter where this straightforward approach holds without caveat. The *TauTransformer Training* workload was both consistent and long-running, enabling reliable per-instance throughput estimates. In Chapter 6, I observed inconsistent runtimes due to SMT effects and adapted my throughput calculation accordingly. In Chapter 7, I studied a similar workload to Chapter 6, but in a real batch system and with a larger and less homogeneous job mix. Therefore, throughput is calculated for the entire batch of jobs, not just the runtime of a single job. Finally, in Chapter 8, throughput is determined directly from non-overhead runtime and the number of processed events. Nonetheless, some benchmark configurations attributed nearly 50% of the runtime to overhead, casting doubt on the accuracy of these results.

Given this collection of edge cases and special considerations, a single throughput metric across all benchmarks is not possible. Instead, throughput is defined for each chapter separately. I consider this the responsible option to ensure the throughput measured throughout these experiments is representative of a saturated batch system environment.

4.3.4 Reasonable Optimization

It is necessary to put the code character used for the workloads of this thesis into perspective. The code quality in HEP is imperfect, especially for user-submitted analysis jobs. Still, software expected to run at a global scale, such as production workflows, must demonstrate a high degree of optimization.

Instead of requiring perfection, I present the benchmarked workloads of this thesis as representative snapshots of different GPU usage regimes in HEP. Each benchmark workload originated from a real HEP analysis and therefore provides a realistic picture of GPU usage in the HEP field.

The benchmark workloads of Chapters 5 through 7 are all based on existing HEP end-user workflows. In contrast, the final benchmark workload presented in Chapter 8 is based on MadGraph4GPU [112], a software framework intended for large-scale production.

The `Docker` container that defines the MadGraph4GPU benchmark was provided by the HEPiX Benchmark Group repository [113], as a candidate for a future GPU HEPScore benchmark. As a result, the expectations regarding the quality of the involved code are higher. Minor modifications to the benchmark container were necessary to adjust it to the available hardware, and high-level parallelization was employed in some benchmark runs. However, the underlying MadGraph4GPU benchmark code was not changed.

4.4 Measurement Methods

The benchmarks in this thesis aim to quantify the relationships among wall-clock time, throughput, and power draw for CPU and GPU resources. To this end, I collected a time series of data throughout each run of the different setups for each benchmark, consisting of:

- Timestamps
- Whole-node power draw
- GPU-only power draw
- GPU utilization
- CPU utilization

Sampling occurred at intervals of up to five seconds. I validated my implementation against the HEP Benchmark Suite, achieving near-identical results, except for slight differences in the time-keeping methodology.

Time Timestamps were recorded with the `date` software [114] at each query. Unlike the equidistant timestamps in the HEP Benchmark Suite, my method allowed slight deviations from the intended interval when the query process took longer than anticipated. The sampling rate and the update rate of the other benchmark metrics, therefore, limited the time resolution. Additional timestamps were added to each record file to indicate the start and end times for each benchmark run, allowing distinction between idle and active time.

Power Draw Whole-system power draw was measured via instantaneous Intelligent Platform Management Interface (IPMI) readings [115], chosen for its wide availability across compute hardware. The selected approach captures total system consumption, whereas TDP estimates provide only a rough upper bound on power consumption. Limitations include coarse one-minute update windows and significant systematic uncertainties of ±2.5% [116]. The power usage effectiveness for cooling at GridKa is reported as 1.3 [117], though throughout my studies, this overhead is not included in the queried power draw data. In this thesis, I rely almost exclusively on the relative comparison of power efficiency between CPU and GPU, on which such a linear factor has no influence. Still, the cooling efficiency factor is included for the estimation of used resources throughout this thesis in Chapter 9.

GPU Metrics GPU utilization and power were recorded using the NVIDIA NVIDIA Data Center GPU Manager (DCGM) [118]. DCGM offers higher fidelity than System Management Interface (SMI) [119], with 100 ms sampling resolution and a more extensive suite of queryable metrics. Utilization was measured using the Streaming Multiprocessor Activity metric (Streaming Multiprocessor Activity (SMACT)), as justified in Appendix A.8. Reported systematic uncertainties were within ±5% [120]. Specific configurations with both a high query rate and a large number of inspected GPU devices were avoided, as they led to idle power draws of up to 100 W in some cases.

CPU Utilization CPU utilization was measured using the instantaneous reporting mode of `top` [121]. Unlike other approaches, this method provides per-sample snapshots rather than runtime averages, enabling accurate reporting of short-lived drops in utilization. The effective resolution was 100 ms, as with the GPU metrics.

Aggregation Following the HEPiX Benchmarking Group's practice, all utilization and power time series were aggregated using the 85% quantile. This reduces the influence of ramp-up and ramp-down effects while retaining near peak behavior. I differentiate between idle and active power draw and utilization measurements throughout the benchmarks, using the start and end time stamps. In this benchmark, idle values refer to the four-minute interval before the workload starts, and active values refer to the benchmark interval itself.

4.5 Measurements of the Idle State

The goal of an HTC like TOpAS is to operate at maximum capacity at all times. However, even with a batch system, it is not possible to achieve complete hardware saturation at all times, due to the composition and quality of the incoming jobs. It is therefore necessary to understand the benchmark machines' idle state, as it illustrates the costs that accumulate even when the hardware is not utilized. Here, *idle* describes the state without active workloads, while background processes required for readiness remain active.

Table 4.1 outlines the power draw metrics during the idle state, averaged over the different benchmark runs. While I did not rely on SMI for evaluating my benchmarks, I still want to point out that the power measurements from DCGM and SMI match within uncertainties.

Table 4.1: Idle power draw of the benchmark system, measured over all benchmark runs. IPMI was used to measure the system's total power draw, while DCGM and SMI measured only the GPUs' power draw.

Power metric	Mean value [W]
IPMI	648 ± 20
DCGM	287 ± 16
SMI	279 ± 14

The system's idle power consumption was approximately 650 watts, of which the eight NVIDIA A100 GPUs made up an average of 287 W, or roughly 45%. The measured uncertainties for both IPMI and DCGM are comparable to their respective systematic uncertainties of 2.5% and 5%. The primary source of idle power draw fluctuations between

Table 4.2: Comparison of TDP, idle, and maximum load power draw. GPU idle power draw is subtracted from total power draw in the CPU-only case. GPU-only refers to the GPU hardware exclusively. The given idle power draw is from the maximum-power-draw benchmark run. The fraction is between the idle and the maximum power draw of the run.

Hardware	Idle [W]	Max. load [W]	TDP [W]	$P_{\text{idle}}/P_{\text{max}}$ [%]
CPU-only	373	663	560	56.3
CPU + GPU	644	2600	2560	24.8
GPU-only	285	2014	2000	14.2

runs was likely software updates over the several-month benchmark period. As each series of benchmark runs was conducted over a comparably short period, fluctuations in the idle power draw across the individual scenarios remained minimal.

Notably, the system's idle power draw does not reflect its power draw under load. Table 4.2 shows the idle power draw, the maximum power draw across all benchmarks, and the combined TDP of the system, both with and without GPU.

The comparison with TDP illustrates the metric's debatable accuracy. While the theoretical maximum aligns well with the measured maximum in the GPU-only cases, the CPU-only and combined cases show loads that are significantly higher than the TDP estimates. The discrepancy allows for two explanations: either I underestimated TDP by omitting a component in my calculation in Section 4.2, or the specified TDP of the components does not accurately represent the real maximum power draw. In either case, additional contributions, such as ventilation, may play a role, as this power draw is included in the total draw but was not included in my TDP estimate.

The GPUs accounted for the majority of the active power draw, with a combined draw of 2 kW, while non-GPU components drew at most 663 W, even under full load. When idle, both GPU and non-GPU power draw are similar, but compared to their respective active values, the difference in the ratios is significant. The relative change in power draw without GPU resources was 78%, while the system with GPU resources increased its active power draw by a factor of 4.

These results highlight the greater energy proportionality of GPUs: their idle-to-load scaling is much larger than that of CPUs or other system components. The behavior, also known as energy proportional computing [122], is a relevant metric for most compute centers with variable usage patterns. It shows that the studied GPUs throttle their power draw significantly more than the rest of the hardware, relative to their delivered compute capacity.

CHAPTER 5

Heavyweight Machine-Learning Workload

Over the past decade, HEP has increasingly adopted ML to tackle major data challenges [123–127]. ML-based techniques are integral components in HEP tasks ranging from event classification to reconstruction and detector simulation. While small-scale ML models suffice for simpler tasks, more complex problems demand larger, more refined architectures to capture intricate relationships in the data. This chapter focuses on the computational costs of such heavyweight ML workloads, motivated by three factors: their proven successes in existing analyses, their necessity for the HL-LHC era, and their excellent scaling with training data.

Firstly, large-scale ML approaches have demonstrated their competence in complex research areas, such as jet tagging and event reconstruction. Examples of this progression include the Particle Transformer [65] and its successors, such as the Universal Particle Transformer [128], which inspired the TauT architecture.

Secondly, even analyses that currently do not require a large-scale NN model might be forced to explore this option for the processing of HL-LHC data. The pile-up per bunch crossing will rise from its current average of 60 to 140-200 proton-proton collisions, substantially increasing event complexity. Even with advanced pile-up mitigation (possibly using transformer NNs themselves [129]), more robust ML models will be necessary to handle the complex event data.

Finally, as shown in Section 3.5, increasing training data volume boosts NN performance, especially in regard to WP FPR. Notably, the physics performance gained in this way is not at the expense of increased inference time, as no changes were made to the existing NN architecture. The price to pay lies entirely in the raised computational requirements during the training process.

Large-scale ML training is among the most compute-intensive single processes in HEP. Less monolithic HEP workloads are typically spread across multiple worker machines and combine their results as a final step, as they are exceedingly parallelizable. Although the gradient descent used for ML can be parallelized to a degree, the overall training process often remains bottlenecked by sequential steps. While research into asynchronous training is ongoing, most ML training cannot be distributed arbitrarily.

Another critical factor is absolute training time. Scaling physics performance with large training datasets can stretch training times into days or even weeks, conflicting with runtime restrictions on many HEP compute resources.

Given these considerations, this benchmark focuses on a heavyweight ML task: the training of the TauT classifier. In this chapter, I compare CPU-only, single-GPU, and multi-GPU configurations in terms of runtime, throughput, and energy efficiency. Two central questions guide us:

- Is CPU-only training feasible?
- What are the tradeoffs between speed and efficiency in multi-GPU training?

5.1 The TauTransformer Training Workload

I selected the training of the TauT classifier introduced in Section 2.4.3 as my heavyweight benchmark workload. The workload presents a HEP-relevant ML computing task with sufficient computational intensity to utilize multiple datacenter GPUs fully. In addition, the computational cost associated with increasing or decreasing the scope of classifier training can be quantified. Increases in the number of epochs or the volume of training data linearly apply to the runtime of the training process as well.

For benchmarking, I used the most computationally demanding TauT configuration from Chapter 3, which also yielded the best-performing model. The training dataset consists of 96 million hadronic tau candidates, split equally across 499 files (450 for training, 49 for validation), with no token pruning. Each benchmark run trained the model for one epoch to balance runtime with workload representation.

With fewer than half a million parameters, the TauT is compact by modern ML standards but still large within the HEP context. It uses 36% the parameters of its predecessor, DeepTauV2.5 [58], and about 22% of of the ParticleTransformer [65].

The training was implemented in TensorFlow 2.10 [72, 130], using TensorFlow-specific features, such as TensorFlow TFRecords and TensorFlow data pipelining. The variable number of constituents in the four input collections per example (ranging from fewer than 10 to over 400) necessitated the use of ragged TensorFlow tensors, which allow for dimensions with a dynamic number of entries.

Hyperparameters were kept consistent with Appendix A.3, except for batch size, which was scaled with the number of assigned GPUs in multi-GPU setups. Depending on the scenario, between one and sixteen concurrent instances of the *TauTransformer Training* workload were executed.

5.2 Benchmark Metrics

Due to the indivisible nature of heavyweight ML workloads, they are among the HEP processes that shouldn't be judged solely by their total throughput. Instead, it is essential to consider how long a full training run takes, since analyses may prioritize reduced runtime over maximum throughput in some cases. For this benchmark, I focus on three metrics: epoch runtime, total throughput, and energy efficiency.

Runtime The runtime t is defined as the wall-clock time between the start and completion of an epoch. In practice, all concurrent instances finish within a negligible variance, so the mean epoch time $\bar{t}$ is reported.

Total Throughput Throughput λ is measured as the number of completed epochs per time unit. With n concurrent instances and a consistent epoch runtime $t_i = \bar{t}$, the total throughput is

$$\lambda = \frac{n}{\bar{t}}. \tag{5.1}$$

Energy Efficiency Energy efficiency Eff_W quantifies the energy required to train one epoch. It is defined in terms of the mean throughput λ and the benchmark's power draw P:

$$Eff_W = \frac{P}{\lambda}.$$

Here, P is taken as the 85% quantile of the active power draw timeseries during the benchmark run, as suggested by the HEPiX Benchmarking Working Group.

5.3 Benchmark Setups

The two main questions that the *TauTransformer Training* benchmark aims to answer are how feasible training on CPU resources would be for this workload, and how well a multi-GPU approach scales. To answer these questions, I selected six benchmark scenarios: one CPU-only baseline and five GPU-assisted ones. Outside of the baseline GPU run, each run was orchestrated to process multiple concurrent instances of the respective setup and utilize all compute resources of the benchmark machine. Although the software setup for each run was identical, the resources available to the individual instances varied.

5.3.1 CPU

To ensure a fair comparison between CPU-only and GPU performance, I originally intended to investigate multiple CPU-thread-per-instance configurations, as they offer differing throughput-to-runtime ratios. This plan had to be reduced in scope due to the exceptionally long runtime of a full-scale CPU training, potentially reaching several weeks. Instead, I performed a preliminary scan to identify the most promising CPU setup and gather a rough understanding of the CPU scaling behavior.

For this scan, in each setup, the workload instances ran a small portion of the whole training epoch. Each run consisted of 15 mini-batch steps, out of a total of $\mathcal{O}(10^6)$, though the first 5 were omitted from the time data to avoid initialization-related skewing. All 256 available CPU threads were pinned to individual workload instances using `taskset` [131], while other resources, such as RAM and training data on disk, were

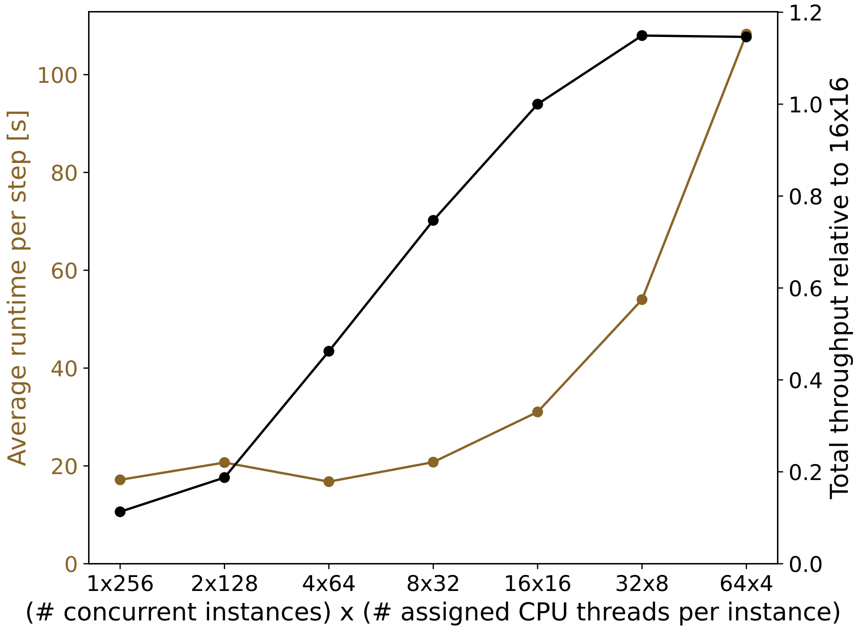


Figure 5.1: Runtime and total throughput of different CPU configurations for the *TauTransformer Training* preliminary CPU scan.

shared. The number of concurrent instances varied from 1 to 64, corresponding to 256 to 4 CPU threads per instance.

The motivation for this approach is that the methods provided by the TensorFlow framework can be insufficient to fully restrict an ML process from selecting CPU resources. This behavior may align with the conditions of a batch system, depending on the system configuration, though it competes with overbooking approaches. Instead of allowing automated load balancing across CPU threads, pinning imposes a specific workload on a particular set of resources. For this benchmark, I decided that load balancing was secondary as all concurrent workloads were identical.

Figure 5.1 shows the results of the preliminary scan, in terms of average runtime per step and the total step throughput across all concurrent instances. Between 256 and 32 threads per workload instance, the runtime remains mostly flat. With fewer CPU threads per instance, the runtime quickly deteriorates, increasing by more than a factor of 5 between 32 and 4 threads per instance. Total throughput increases nearly linearly from 256 to 16 CPU threads per instance, but then plateaus. These results suggest that NN training processes orchestrated with TensorFlow struggle to utilize more than 32 CPU threads efficiently during training. The expected performance boost from further parallelization is offset by overhead, leaving performance flat despite the additional compute resources. Other potential sources of performance bottlenecks could be external factors, such as limitations in the provision of input data or memory constraints. In practice, this is unlikely, as the total throughput rises when CPU resources are distributed across multiple instances of the *TauTransformer Training* workload.

While the per-instance performance reaches a ceiling with excessive CPU resources, there is also a throughput floor when resources become spread too thinly. Between the 4- and 8-thread-per-instance setups, the average runtime decreases linearly with the number of threads.

Based on this scan, I selected a compromise configuration between speed and throughput: The variation with 16 concurrent training instances with 16 CPU threads per instance offers 87% of the maximum throughput and 55% of the maximum speed. Alternative configurations (32 and 64 threads per instance) offer valid alternatives, with up to twice the speed, though at the cost of 50% reduced total throughput.

Even with only one CPU configuration, benchmarking an entire epoch on the complete dataset would require approximately 300 hours. I therefore decided to run the CPU-only baseline benchmark at a reduced scale, using 10% of the dataset during training and validation. To account for this scale-down in the training data, the CPU-only scenario runtime is scaled up by a factor of 10 during the benchmark evaluation. This approach allowed for a fair and meaningful comparison between CPU-only baseline and GPU-accelerated training runs, although with a reduced precision in the CPU-only results. The finalized CPU-only baseline setup was generally identical to the preliminary scan setup, except for the stopping condition of one epoch instead of 15 mini-batch steps.

5.3.2 GPU

Due to the significantly shorter runtime of benchmark runs involving GPU resources, I evaluated five GPU-based scenarios. The first scenario was an exception to the full-saturation approach and serves as a baseline for measuring the impact of non-GPU resource contention on training performance. In it, one instance of the *TauTransformer Training* workload is run with the assistance of one A100 GPU. The remainder of the benchmark machine was left unoccupied.

The other four GPU scenarios consider different distributions of GPU resources across concurrent instances, similar to the preliminary CPU resource scan. Instead of varying the number of CPU threads per training instance, I varied the number of GPUs assigned to each instance. The four scenarios consist of 1 to 8 concurrent instances of the workload, with 8 GPUs fully distributed across them.

There are two common multi-GPU training strategies: model parallelism, where a single NN model is distributed across multiple GPUs, and data parallelism, where the model is replicated and batches are processed in parallel across all copies. The former is mainly used when the model itself is too large to fit in a single GPU's memory, as seen with high-end large language models. In this case, this distribution is unnecessary, as the model is comparably small.

Data-distributed multi-GPU training splits a training batch into multiple sub-batches and distributes them across multiple GPUs. After the gradients of all sub-batches are computed, they are collected and processed by a central optimizer. In most regards, this approach efficiently extends the usable compute capacity for an NN training by processing more data concurrently. Still, while this method can improve single-training runtime, it does so at the cost of total throughput.

There are three main reasons why this method is expected to fall short of linear speedup in practice. Firstly, synchronous training requires that all sub-batches of a training batch be present before the optimizer can take the next gradient descent step. The runtime per batch can vary depending on the size and complexity of the training data. As a result, idle time occurs at the end of each batch, as the optimizer waits for the remaining sub-batches to arrive.

Secondly, some parts of the gradient descent loop remain sequential and do not profit from additional GPU hardware. Instead, these steps are slowed down as inter-device communication between GPUs has higher latency than intra-device communication. The rise in sequential compute time can be counteracted by increasing the total batch size across the GPUs, though this also risks degraded convergence with overly large batch sizes. Much of the sequential overhead scales non-linearly with batch size, and increasing both the number of GPUs and the total batch size typically results in comparable overhead.

Finally, as training accelerates, all other parts of the algorithm must keep pace. If bottlenecks shift to other system components, such as disk I/O or memory, the overall training speed may not improve, even with multiple GPUs. The input data pipeline must operate at higher speeds, and the connection between the CPU host and the GPU device experiences increased traffic. Hard constraints, like the speed of CPU memory, the connection bandwidth to the PCIe slots, or the I/O speeds, all have to be considered.

While it was not possible to reduce fluctuations in sub-batch runtime, I minimized the remaining performance limiters by scaling the batch size with the number of assigned GPUs. The risk to convergence remained minimal, as the number of steps per training epoch remained above $\mathcal{O}(10^6)$. I used TensorFlow's Mirrored Strategy API for synchronous multi-GPU training in my benchmark runs and took several steps to ensure minimal non-GPU bottlenecks:

- Adoption of the TensorFlow TFRecord binary data format to streamline I/O access. This binary data format is intended for large, complex training datasets and provides high read speeds and compression, while allowing for sharding across multiple files.

- Implementation of data pre-processing using TensorFlow's computational graphs. The training data requires significant pre-processing as part of the input pipeline. By relying on pre-compiled compute graphs, I parallelized the CPU-bound work and sped it up by up to an order of magnitude.

- Assigning of individual input pipelines for each device, with datasets partitioned by file. Data loading was highly parallelized by assigning each GPU a fixed subset of the 499 training files. Each pipeline operated concurrently, supplying data only to its respective GPU, and reducing the bottleneck of sequential operations in the input pipeline.

In contrast to the CPU-only baseline setup, each *TauTransformer Training* instance consistently used 16 pinned CPU threads and the GPUs were pinned to the respective *TauTransformer Training* instances via the `CUDA_VISIBLE_DEVICES` environmental variable.

5.4 Benchmark Results

This section presents the results of the CPU and GPU *TauTransformer Training* benchmark runs. Firstly, I examine the CPU-only baseline performance and contrast it with the baseline single GPU setup (8 × 1). Afterwards, I compare the GPU-assisted variants among themselves. Table 5.1 and Figure 5.2 summarize the benchmark metrics for the CPU-only baseline and GPU-assisted scenarios, including mean runtime, total throughput, idle and active power draw, energy efficiency, and GPU utilization.

CPU The runtime of the CPU-only baseline with 16 concurrent workload instances and 16 CPU threads per instance is 75.7× slower than that of the baseline GPU run. Although the CPU-only baseline run has more concurrent instances, total throughput is still 37.9× lower than the baseline GPU scenario. The CPU-only baseline scenario draws only 25.7% as much power as the baseline GPU run, though 56% of that power draw is also present without an active workload. At 8.82× the power per epoch, the CPU-only baseline configuration is also the least energy-efficient.

GPU The comparison between the GPU-assisted runs is more nuanced. The small runtime difference between the 1 × 1 and 8 × 1 configurations indicates minimal impact from non-GPU resource contention. At a runtime difference of only 10%, resource contention has only a negligible effect on the performance of the concurrent instances.

The comparison among the multi-GPU scenarios shows a steady decrease in runtime as more GPU resources were added. Increasing the number of GPUs per workload instance from one to eight reduced runtime by 72.8%. However, this also reduced the number of concurrent instances, resulting in a 53.9% drop in total throughput.

The idle power draw of this benchmark fluctuated between 616 W and 703 W. While the cause of this fluctuation is unclear, its impact on active power draw appears minimal, as the 4 × 2 and 2 × 4 configurations exhibit behavior that is well aligned with the 8 × 1 and 1 × 8 variants.

In terms of total active power draw, all GPU variants showed significantly higher values than the CPU-only baseline scenario, though power draw correlated with total throughput. While the 8 × 1 variant delivered 116.9% more throughput than the 1 × 8 configuration, it also required 40% more power to do so. As a result, power efficiency was similar, with the maximally concurrent variant requiring 35% less energy per epoch than the maximally parallel configuration.

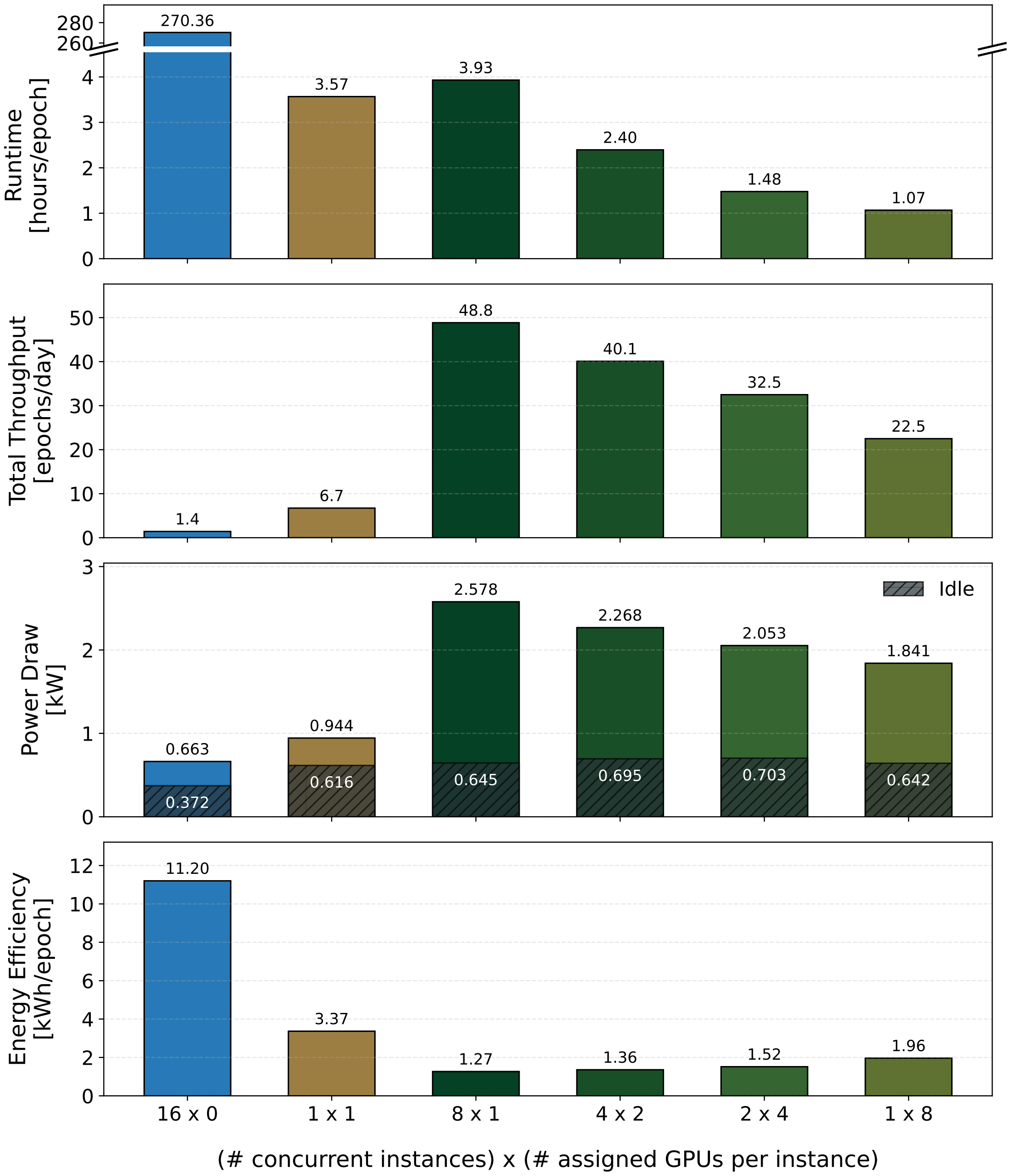


Figure 5.2: Summary of the CPU-only baseline and GPU-assisted benchmark results. The blue 16×0 setup corresponds to the CPU-only baseline, while all others show GPU-assisted results. The 1×1 setup does not fully utilize the available hardware's computational capacity and is intended for runtime comparisons. The CPU-only power draw and efficiency results do not include the GPUs' idle power draw, as they simulate a CPU-only machine.

Table 5.1: Summary of the CPU-only baseline and GPU-assisted benchmark results. The 16×0 setup refers to the CPU-only baseline scenario, while all others show GPU-assisted results. The 1×1 setup does not fully utilize the available hardware's computational capacity and is intended for runtime comparisons. The CPU-only power draw and efficiency results do not include the GPUs' idle power draw, as they simulate a CPU-only machine.

Configuration name	16×0	1×1	8×1	4×2	2×4	1×8
# GPU / process	0	1	1	2	4	8
# Concurrent processes	16	1	8	4	2	1
Mean runtime t [h / epoch]	270.36	3.57	3.93	2.40	1.45	1.07
Total throughput λ [epochs / day]	1.4	6.7	48.8	40.1	32.5	22.5
Idle power draw P_i [W]	373	616	645	695	703	642
Active power draw P_a [W]	663	944	2578	2268	2053	1841
Efficiency Eff_W [kWh / epoch]	11.20	3.37	1.27	1.36	1.52	1.96
GPU utilization [%]	—	72	68	55	46	36

5.5 Results Discussion

After gathering data from the benchmark runs, the core questions of this chapter can be addressed: is CPU-only NN training feasible, and what are the trade-offs between runtime speed and efficiency for multi-GPU training?

CPU Except for absolute power draw, the CPU-only baseline training underperforms in all benchmark metrics compared to the GPU variants. Especially the nearly two orders of magnitude slower runtime compared to the slowest GPU variants is damning. Even considering the potential 50% increase in single-training speed with 64 CPU threads per training instance, the gap barely shrinks. Furthermore, although I selected a throughput-oriented scenario for the whole benchmark run, total CPU utilization reached only 46.8%. This shows how unsuited the benchmark workload is to the CPU, as TensorFlow struggled to utilize the available resources efficiently. At a runtime of more than 11 days per epoch, factors beyond runtime must be considered. The stability of computing resources is not perfect, and interruptions of training processes can occur. Even if the trained model was saved via checkpoints after every epoch, an interruption could result in a setback of several days.

While it's possible to train on a reduced dataset, Section 3.5 shows that this incurs a measurable performance loss. Even with only 10% of the total dataset, a relatively short training time of 10 epochs, and 64 CPU threads per training, the training would still take around 180 hours. This alone illustrates why GPU resources are necessary for any large-scale ML effort.

Past the runtime, neither throughput nor power efficiency comes close to the performance of the GPU-assisted runs. Even considering the cost difference between a CPU-only and a CPU+GPU compute node, the 8×1 setup still provides an 11.9× better cost-efficiency. So, while technically feasible, CPU-only NN trainings should only be considered as an option of last resort for heavyweight ML.

GPU Among GPU setups, the tradeoff is primarily between per-instance speed and overall throughput. Based on the runtime of the 1 × 1 run, there is little non-GPU resource contention for concurrent instances of the *TauTransformer Training* workload. It is still possible that resources shared for this benchmark, such as disk performance, could act as bottlenecks for real batch system jobs.

Runtime decreases consistently as parallelism increases, though not in proportion to the drop in concurrency, mirroring the trends in GPU utilization and active power draw. Given that less work is being performed per second, the observed reduction in power draw is not unexpected. As a result, power efficiency between the extremes in concurrency and parallelism is similar.

The final trade-off for multi-GPU considerations, therefore, is that a 3.67× speedup comes at the cost of 53.9% lower total throughput and 54.3% higher energy usage per training epoch. This tradeoff may be justifiable when runtime concerns outweigh throughput and efficiency concerns.

Still, if such a scenario is considered in the context of an HEP batch system, then the likelihood of a job with such high resource requirements successfully matching becomes slim. The amount of resources required at once is high, and most computing centers would rather run other jobs on free resources than wait for a single monolithic job to start. Therefore, such a multi-GPU job is more in line with whole-node scheduling, commonly associated with high-performance computing, than with high-throughput computing.

5.6 Summary

This chapter evaluates the computational performance of the hadronic tau classifier TauT introduced in Section 2.4.3. The benchmark runs included a CPU-only scenario and multiple single- and multi-GPU configurations. The benchmark found that CPU resources are highly inefficient for this workload, lagging behind GPU setups by orders of magnitude in runtime, throughput, and energy efficiency. When considering the cost of procurement, GPU-supported training provided an 11.7× higher compute-per-cost ratio than CPU-only training, while also achieving an 8.8× improvement in energy efficiency. Multi-GPU configurations offered significant training acceleration with a reasonable tradeoff between throughput and energy usage. Parallelizing training across eight A100 GPUs achieved a 3.67× speedup in runtime at the cost of a 53.9% reduction in total throughput and a 54.3% increase in energy usage per training epoch.

CHAPTER 6

Lightweight GPU Workload

The field of HEP encompasses a diverse range of workflows, resulting in wide variations of resource requirements. While Chapter 5 examined the heavyweight *TauTransformer Training* workload, which fully occupied one or more GPUs, this chapter focuses on workloads at the opposite end of the spectrum: tasks that utilize only a fraction of a modern datacenter GPU's capacity.

While GPUs thrive on massive parallel workloads, offering high efficiency in terms of both cost per compute and compute per Watt, not every task can be expressed as a highly parallel process. Writing GPU-capable code is often challenging, and even tasks that could in principle benefit from GPU acceleration may fail to do so. A typical example from HEP is ML tasks with small-scale architectures, trained on high-level observables. Such NN trainings are typically composed of comparably shallow feed-forward networks with few layers [1]. Training data for these architectures is commonly characterized by a low number of information-rich features and a comparably small volume. As gradient descent is fundamentally sequential, the limits of parallelism are eventually reached for these workloads, and GPU utilization is limited.

Regardless of their origin, low-utilization GPU jobs are a significant part of current HEP workloads. Figure 6.1 shows an exemplary day in which one TOpAS worker machine was predominantly occupied by GPU jobs with limited GPU utilization. Such scenarios raise an important question: should these workloads run on GPUs, or are CPU resources still a viable alternative? Porting code to the GPU requires significant effort, and efficient gains are not guaranteed, especially for partially sequential workloads. Evaluating these trade-offs is crucial to maximizing the value of GPU infrastructure and determining the viability of GPU technology across the broader landscape of HEP software.

This chapter first introduces the *MSSM Training* workload, followed by the initial CPU and GPU benchmark scenarios. It goes on to motivate and explore three additional multi-tenant GPU setups and their respective technologies. Finally, results are re-evaluated in this broader context and summarized in the conclusion.

[1] Originally, any network with more than one hidden layer was considered *deep learning*, but since then usage has shifted, making the term less precise across the spectrum of network sizes.

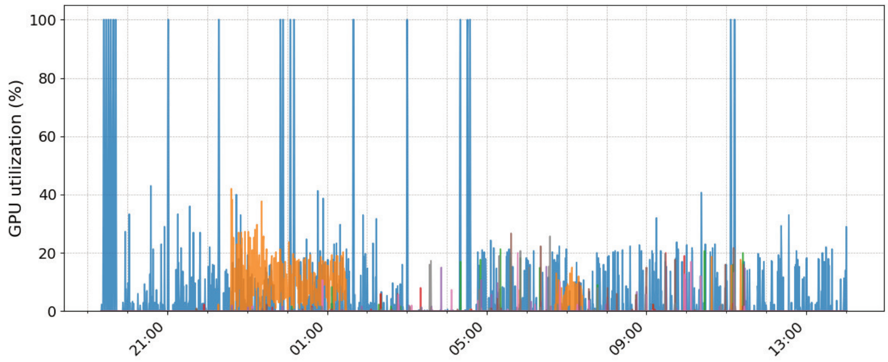


Figure 6.1: Example of a time period with a high percentage of low GPU utilization jobs on one TOpAS worker machine. Each color represents one of eight NVIDIA V100S GPUs.

6.1 Low-Complexity HEP ML Training

The *MSSM Training* workload consists of a multi-classification NN training originally developed as part of an H→ $\tau\tau$ Minimal Supersymmetric extension of the Standard Model (MSSM) analysis [132]. The architecture is a fully connected feed-forward NN with three hidden layers and six output classes. Compared to the architecture of the *TauTransformer Training* workload, it is considerably simpler. This workload represents an everyday, lightweight use of NN architectures relevant to HEP. It runs efficiently on GPUs and is still trainable in a reasonable time frame on CPU resources, making it well-suited for comparative benchmarks.

The original analysis relied on two identical networks for a k-fold cross-validation [133], but for this benchmark, only one of them is trained. The dataset consists of 1.48 million examples with 14 high-level features, split into training (75%) and validation (25%) sets. Each epoch consists of 1000 steps, with mini-batches generated using a *balanced batch* approach. With it, each output class label contributes 100 examples to a mini-batch, for a total of 600 examples per batch. While the original implementation relied on early stopping, I used a fixed number of 100 epochs to ensure stable runtime, roughly matching the average training length observed in the analysis. Other hyperparameters and physics performance details are documented in the original work and are not central to this benchmark.

The training was implemented in TensorFlow 2.10 [72, 130] and configured to use a GPU when available, requiring roughly 3.3 GB of VRAM. To guarantee consistent hardware utilization, `taskset` [131] was used to pin CPU threads to instances of the *MSSM Training* workload. GPUs were assigned via the environment variable `CUDA_VISIBLE_DEVICES`, as is standard operation in an HTCondor batch system with GPU access. Initial high-level profiling with TensorBoard [130] confirmed that the workload is neither I/O nor CPU bound.

Each benchmark run consisted of multiple concurrent NN training instances, scaled to saturate the available 256 CPU threads and eight GPUs under the respective scenarios.

6.2 Benchmark Metrics

HTC centers aim to maximize throughput, while maintaining energy efficiency. Accordingly, my evaluation of the *MSSM Training* benchmark focuses on three key metrics: runtime, throughput, and power draw.

Non-Standard Circumstances The benchmark workload consisted of an ML process, replicated across the machine until all available resources were saturated. Ideally, all instances would complete in the same time $t_i = \overline{t}$, yielding a mean throughput of

$$\lambda = \frac{n}{\overline{t}}, \tag{6.1}$$

with n denoting the number of concurrent instances.

In practice, some scenarios produced uneven runtimes despite seemingly identical resource allocation. Appendix A.10 goes into detail on the origin of the observed discrepancies, and the situation can be narrowed down to SMT effects. The irregular distribution of runtimes in the affected scenario is caused by CPU contention, and the behavior is reproducible. Therefore, I treated the mixed runtimes analogously to running on mixed hardware in a batch system.

Runtime and Throughput In a batch system with multiple hardware types, differences in runtime are expected, even for an otherwise identical workload. When runtimes differ, throughput must account for the relative contribution of fast and slow jobs. The harmonic mean captures this:

$$\lambda = \sum_{i=1}^{n} \frac{1}{t_i}, \tag{6.2}$$

$$\overline{t} = \frac{n}{\lambda} = \frac{n}{\sum_{i=1}^{n} \frac{1}{t_i}}. \tag{6.3}$$

This approach is valid as long as runtimes are stable and not influenced by external conditions. This was the case for all configurations discussed in this benchmark, including the ones affected by SMT-based CPU contention.

Power Draw and Utilization The power consumption and utilization metrics are equally affected by the inconsistent runtimes. The metrics are only meaningful while all workload instances remain active and the machine is fully saturated. As soon as any workload instance finishes, the total power draw and resource utilization decrease. I therefore compute the 85% quantiles only for periods in which all concurrent instances of the configuration were still active. This approach guarantees comparability across setups and consistency with the throughput metric. GPU-focused configurations of this benchmark rely on varying numbers of CPU threads as the number of concurrent instances changes. CPU utilization is therefore provided relative to the total number of assigned threads.

6.3 Initial CPU and GPU Benchmark Scenarios

The primary goal of the *MSSM Training* benchmark was to compare CPU-only and GPU-assisted configurations in terms of runtime, throughput, and energy efficiency. This section presents the initial benchmark scenarios, chosen to reflect conditions typical of an HTCondor batch system without any non-standard customization.

6.3.1 CPU Scan

In the CPU-only setup, the number of CPU threads assigned to each workload instance was varied to determine an optimal resource distribution. This scan follows the same principle as the preliminary CPU scan of the *TauTransformer Training* benchmark in Chapter 5. However, thanks to the reduced computational complexity of the *MSSM Training* workload, complete benchmark runs of every relevant CPU configuration were feasible.

Thread counts per instance of 2, 4, 8, and 16 were tested, with the number of concurrent instances adjusted to saturate the 256 available CPU threads ($n_{\text{instances}} = 256/n_{\text{threads}}$). Table 6.1 summarizes the tested configurations.

6.3.2 Exclusive GPU

In typical HTCondor configurations, GPUs are treated as discrete, exclusive resources: each GPU is allocated to at most a single job at a time. I therefore assigned one workload instance per GPU. Each GPU-assisted workload instance was paired with two pinned CPU threads.

6.4 Initial Benchmark Results

This section presents results from the initial *MSSM Training* benchmark configurations beginning with the CPU-only scenarios, and then contrasting them with the Exclusive GPU setup. Comparisons of runtime, throughput, and energy efficiency motivate the need for further benchmark scenarios that better utilize GPU resources.

6.4.1 CPU

Table 6.1 lists the results of the four CPU-based setups, while Figure 6.2 summarizes the critical metrics relative to one another.

Runtime The 16 × 16 configuration achieved the fastest mean runtime of 34.77 min. Runtime then rose steadily with increasing concurrency, reaching 164.61 min for the 128 × 2 configuration (4.7× slower).

Throughput Throughput rose with more concurrent instances, peaking at 47.4 instances per hour in the 64 × 4 setup, a 72% increase over the lowest-performing configuration. The 128 × 2 setup underperformed slightly at 46.3 instances per hour.

Table 6.1: Benchmark results of the CPU-only scenarios. The contributions of idle GPUs to idle and active power draw were removed to simulate a CPU-only machine.

Configuration name	16 × 16	32 × 8	64 × 4	128 × 2
# CPU / instance	16	8	4	2
# Concurrent instances	16	32	64	128
Mean runtime [min]	34.77	51.03	79.83	164.61
Throughput [inst. / h]	27.6	37.6	47.4	46.3
CPU utilization [%]	62.3	71.2	83.5	95.4
Idle power draw [W]	361	367	360	361
Active power draw [W]	650	659	671	682
Efficiency [Wh / inst.]	23.54	17.51	14.17	14.73

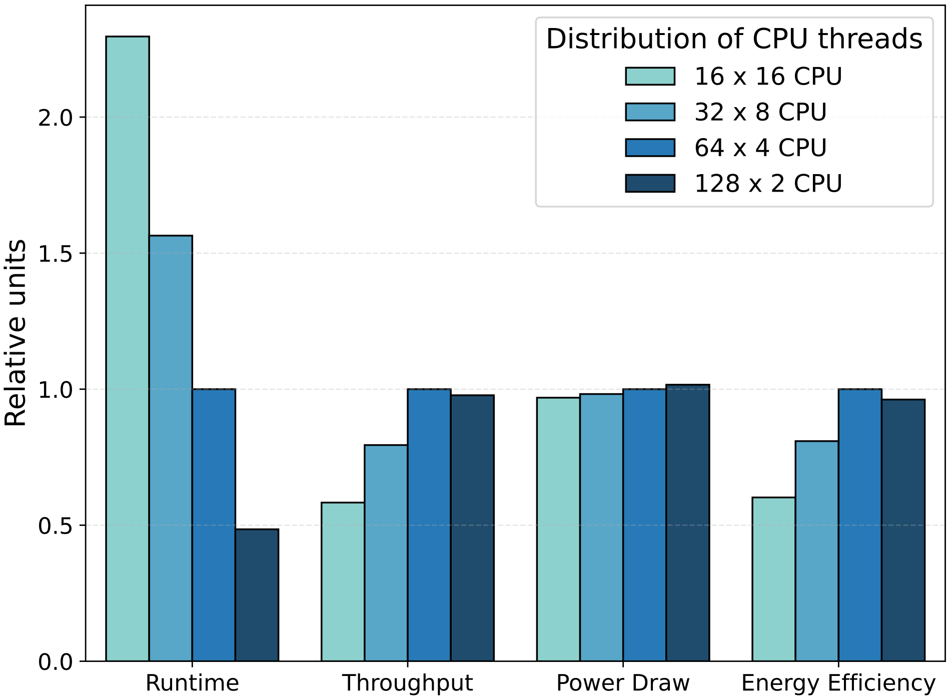


Figure 6.2: Relative performance metrics of the CPU-only scenarios. All values are normalized to the 64 × 4 configuration results. Power draw values exclude idle GPU contributions to simulate a CPU-only machine.

Table 6.2: Benchmark results of the Exclusive GPU scenario compared to the baseline CPU configuration. Idle GPU power draw was removed from the CPU values to simulate a CPU-only machine.

Configuration name	64 × 4 CPU	Exclusive GPU
# CPU / instance	4	2
# instances / GPU	—	1
# Concurrent instances	64	8
Mean runtime [min]	79.83	7.75
Throughput [inst. / h]	47.4	61.9
CPU utilization [%]	83.5	65.2
Idle power draw [W]	360	640
Active power draw [W]	671	764
GPU utilization [%]	—	6.7
Efficiency [Wh / inst.]	14.17	12.34

CPU utilization CPU utilization increased monotonically with the number of concurrent instances, from 62.3% (16 × 16) to 95.4% (128 × 2). Unlike throughput, CPU utilization did not decrease for the 128 × 2 scenario.

Power draw Power draw directly related to the idling GPU resources was subtracted from the power draw values to simulate a CPU-only machine. The metric rose from 650 W to 682 W with rising concurrency (a 5% increase). Idle power draw was stable at 364 ± 3W across all runs.

Energy efficiency Energy efficiency, measured as the amount of energy required per workload instance, mirrored throughput trends. The 64 × 4 configuration offered the best result with 14.17 W h per instance, nearly 40% better than the 16 × 16 configuration. The drop in throughput for the 128 × 2 configuration directly translates to a diminished energy efficiency.

Baseline choice Based on these results, the 64 × 4 CPU configuration was selected as the baseline, primarily for its superior throughput and energy efficiency.

6.4.2 Exclusive GPU

Table 6.2 summarizes the Exclusive GPU scenario alongside the baseline 64 × 4 CPU configuration.

Runtime The Exclusive GPU setup achieved a mean runtime of 7.75 min, making it 10.3× faster than the CPU baseline at 79.83 min.

Throughput Throughput increased less dramatically. The GPU configuration delivered 61.9 instances per hour, 30.5% more than the CPU baseline's 47.4 instances per hour.

Utilization CPU utilization decreases from 83.5% in the baseline CPU scenario to 65.2% in the GPU scenario, reflecting that CPUs primarily handled data preprocessing and transfer. GPU utilization reached only 6.7%, suggesting that the hardware was severely underutilized.

Power Draw Power consumption was notably higher for the GPU scenario. Idle power draw increased from 360 W to 640 W due to the inclusion of the GPU idle power draw. Active power draw rose from 671 W to 764 W, a 14% increase.

Energy Efficiency Despite the higher power requirements, the Exclusive GPU setup reached slightly better energy efficiency. At 12.34 W h per instance, it required 12.9% less energy per instance than the CPU baseline.

6.5 Initial Results Discussion

My analysis of CPU and GPU performance for lightweight GPU workloads highlights critical tradeoffs between runtime speed and energy efficiency. It also shows the potentially extreme underutilization of GPU resources in an exclusive-allocation setup.

6.5.1 CPU

When comparing different CPU threads-per-instance configurations, the most efficient setup is located in the middle of the spectrum. While configurations with more threads per instance achieve faster runtimes, this comes at the expense of the total number of concurrent instances. Consequently, even as the runtime increases with fewer threads per instance, total throughput continues to improve up to 64 concurrent instances.

The difference in performance between the 64 × 4 and 128 × 2 CPU setups illustrates the limitations of this trend. At very high concurrency, performance begins to deteriorate. I conclude that this behavior arises from external bottlenecks, though a definitive cause could not be identified. Potential contributors include memory bandwidth, I/O throughput, CPU overhead per thread, and the CPU boost frequency behavior. A likely explanation for the drop only at maximum concurrency is that a tipping point between rising CPU utilization and overhead is reached. So, while all configurations experience some overhead, in setups with fewer workload instances, the performance gains outweigh the costs, whereas at very high concurrency, the overhead dominates.

Power draw across CPU configurations was nearly identical, though a slight increase was observed with higher CPU utilization. This is expected, as performing more work per second naturally requires more power. The relatively small differences may be due to the boost behavior of AMD EPYC CPUs, which dynamically increase the frequency of active CPU cores during periods of low power draw, resulting in faster, but less energy-efficient, computation. As a result, even at moderate utilization, power draw remains mostly constant. Notably, power draw continues to rise with concurrency, even as throughput declines for the 128×2 setup.

Energy efficiency closely tracks throughput trends because the power draw is nearly constant across all CPU-only scenarios. This reinforces the concept that, due to low energy proportionality, maximizing throughput is the dominant factor in energy efficiency for CPU-only workloads.

The CPU results highlight two key concepts:

- High parallelization can speed up individual instances, but tends to reduce total throughput.
- This behavior has practical limits, as overhead eventually outweighs performance gains at high degrees of concurrency.

6.5.2 Exclusive GPU

In many ways, the GPU setup represents an extreme case of the CPU configurations: while runtime is accelerated, the number of concurrent instances is drastically reduced. The limited GPU concurrency is a complex topic in itself, but its impact on overall performance in this comparison is clear.

Adding GPU hardware reduced runtime, as GPUs are generally better suited for ML workloads and contribute vast quantities of processing power to the setup. The *MSSM Training* workload achieved a roughly tenfold speedup, which is a significant improvement, but far less than the nearly two orders of magnitude observed with the *TauTransformer Training* workload in Chapter 5. This discrepancy can be attributed to the higher fraction of sequential code in the *MSSM Training* workload, as described by Amdahl's Law [134]. Further acceleration beyond the observed improvement is unlikely through parallelism alone, given the measurable under-utilization of the assigned GPUs.

Despite the significant reduction in runtime, total throughput only increased by 30.5% relative to the CPU baseline. This is essentially a consequence of the low concurrency in the Exclusive GPU setup, which severely limits the number of simultaneous instances that can be processed.

The economic costs of GPUs further complicate the picture. GPUs impose a significant up-front cost, often exceeding the price of the entire remaining machine. For the TOpAS benchmark machine, adding the eight NVIDIA A100 GPUs nearly tripled the procurement cost. It is therefore expected that the machine with GPU resources provides at least three times the throughput of the machine without the GPUs to reach the same cost-per-compute. This hurdle is missed in this benchmark, with only a 1.3× increase in throughput.

These results naturally raise the question: should lightweight, GPU-capable workloads instead rely on CPU resources, reserving GPUs for workloads that benefit more from them, such as the *TauTransformer Training* workload? Based entirely on this initial comparison, the answer depends on context.

If energy efficiency is the primary metric, then relegating lightweight GPU applications to CPU might indeed be preferable. However, if direct turnaround is critical for end-user analyses, a tenfold speedup may justify the higher resource cost.

In addition, declining specific workloads risk prolonged idle time for GPU resources. Running the same workload on CPU resources is significantly less efficient if the same machine already includes GPUs that would then remain idle. As an example, the CPU-only scenario with idle GPUs would require 62.7% more energy per instance than the Exclusive GPU scenario.

Finally, and most importantly, the Exclusive GPU setup is clearly suboptimal: with only 6.7% SMACT, most of the GPU resources remain unused. This observation motivates exploring multi-tenant GPU usage to better utilize available hardware.

6.6 Multi-Tenant GPU Technologies

Following the initial Exclusive GPU scenario, it became evident that a more involved set of configurations was necessary to provide a complete picture. While exclusive GPU access remains the default in many batch systems, including HTCondor, it is often not optimal for all workloads. The low hardware utilization observed with the *MSSM Training* workload suggests that placing multiple concurrent workload instances on the same GPU could improve efficiency.

However, sharing a GPU among multiple processes introduces new challenges. While modern GPUs have made strides in process isolation, specific gaps remain. Before discussing the available GPU-sharing technologies, it is crucial to motivate multi-tenant GPU usage in the broader context and highlight the two main concerns of such usage: Out-Of-Memory (OOM) behavior and interference between concurrent processes.

Motivation for multi-tenant GPUs On the one hand, physicists who rely on cutting-edge ML techniques in their research require access to high-end hardware. Otherwise, turnaround times threaten to become a crucial bottleneck, as seen in Chapter 5. On the other hand, lightweight GPU-dependent workloads, such as the small-scale ML task in this benchmark, require only a fraction of that capacity. For HEP HTC centers, the challenge is to efficiently provide resources for both small- and large-scale workloads without provisioning multiple GPU types. A practical compromise is to procure high-end GPU resources and partition them among multiple tenants.

Past the field of scientific computing, cloud computing also has good reason to support the development of multi-tenant GPU technologies. Similar to HEP computing, they aim to market hardware access with maximum efficiency. While some of their clients require cutting-edge hardware to run their businesses, others only need a fraction of it. As a result, both scientific computing and the commercial cloud computing industry share a common interest in advancing the field of efficient GPU sharing. The HEP field can benefit from this common interest and the technologies that arise from it.

VRAM limitations GPU memory, often called VRAM, is a critical resource, as all data used by a GPU process must reside on the device first. Much like CPU RAM, VRAM imposes a hard limit: if a process attempts to allocate more memory than is available, the process fails. GPU OOM errors are typically abrupt and can leave all active processes in an undefined state. Strict VRAM management is therefore essential for multi-tenant approaches, as most GPU processes do not impose any intrinsic VRAM limits beyond the hardware's overall limits. Unlike CPU RAM, proactive mechanisms to effectively enforce per-process VRAM limits are sparse. Most GPU vendors do not provide tools to set rigid VRAM quotas through standard programming frameworks. CUDA offers some support, discussed further in Section 6.6.3, but in general, GPUs were historically designed for exclusive access by a single process.

Reactive approaches, such as terminating processes after they exceed VRAM limits, are also challenging. With the TOpAS benchmark machine, for example, each GPU had only 40 GB of VRAM, compared to 1 TB of shared system RAM. Many frameworks pre-allocate large portions of VRAM during initialization, assuming exclusive access. In addition, VRAM access tends to be more volatile than RAM access, as a GPU's high throughput enables rapid change of the data stored in VRAM. As a result, available VRAM can be exhausted within seconds, faster than a periodic supervisor can reliably respond to, making reactive management impractical.

Process interference While less catastrophic than VRAM OOM failures, interference between GPU contexts (structures analogous to CPU processes) can significantly impact performance. In the past, concurrent contexts on a GPU could directly influence each other's memory states, though modern GPUs place guardrails to prevent such overreach. While modern GPUs support multiple contexts concurrently, context management remains less mature than CPU process management and can still degrade performance.

Scope of this work In this thesis, I examine three multi-tenant GPU strategies under the *MSSM Training* benchmark: default Time-Slicing, Multi-Process Service (MPS), and Multi Instance GPU (MIG). All are part of the CUDA ecosystem, although other GPU manufacturers also employ concepts similar to Time-Slicing. My selection focused on technology actively supported by NVIDIA due to compatibility with the available hardware in the TOpAS cluster.

6.6.1 Default Time-Slicing

The most basic approach to multi-tenant GPU usage on NVIDIA hardware is the default Time-Slicing mechanism, introduced with the Volta generation. As illustrated in Figure 6.3, multiple processes may create GPU contexts and offload work to the same device. However, only one context can execute kernels at a time. While kernels belonging to a single context may run concurrently, inter-context concurrency is not supported. In practice, CUDA GPUs are therefore limited to *one active context per GPU.*

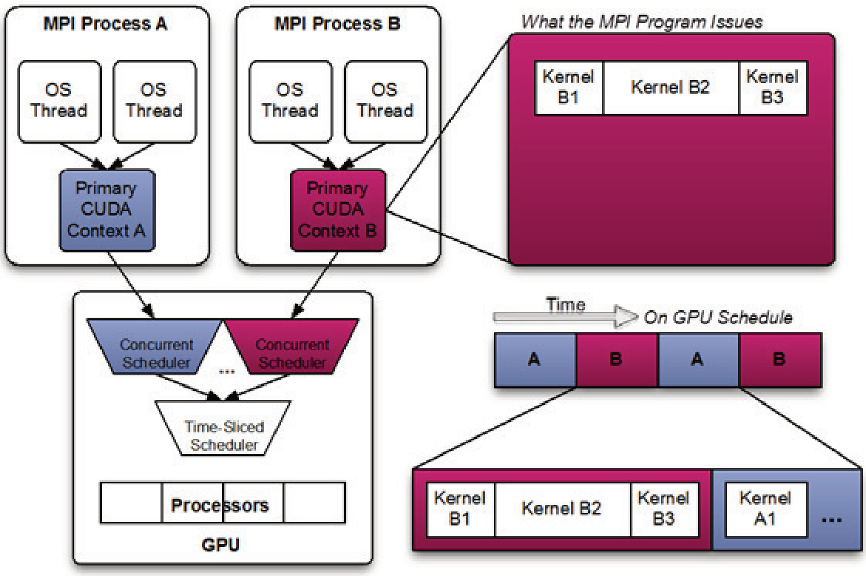


Figure 6.3: Sketch of CUDA Time-Slicing behavior [135]. When multiple processes create GPU contexts, only one can launch kernels at a time. Internal scheduling interleaves access in a round-robin fashion, making the effective runtime comparable to sequential execution.

Access time is divided among active contexts in a round-robin manner, ensuring that no process must wait for all others to complete before resuming. This gives the appearance of parallel execution, although at a significantly reduced per-process speed. From the perspective of total kernel throughput, Time-Slicing offers only small advantages over sequential execution. In fact, sequential processing may be preferable in some cases, as Time-Slicing still requires all active contexts to remain in memory simultaneously, placing pressure on GPU VRAM.

To probe the effectiveness of Time-Slicing, I tested the *MSSM Training* workload on a single NVIDIA A100 GPU while varying the number of concurrent instances from 1 to 12. The individual instances were identical to the setup described in Section 6.3.2, except that all instances were assigned to the same GPU. The resulting runtime and utilization are shown in Figure 6.4.

Three main features are visible:

- All runs exhibit a roughly three-minute constant runtime offset, which I attribute to workload overhead unrelated to the GPU.
- Runtime increases steadily with additional processes, reaching 5.7× the baseline at 12 concurrent instances.
- GPU utilization remains consistently below 10%.

These results show that Time-Slicing reduces effective performance to that of sequential execution if the kernel queue of the respective contexts is saturated. However,

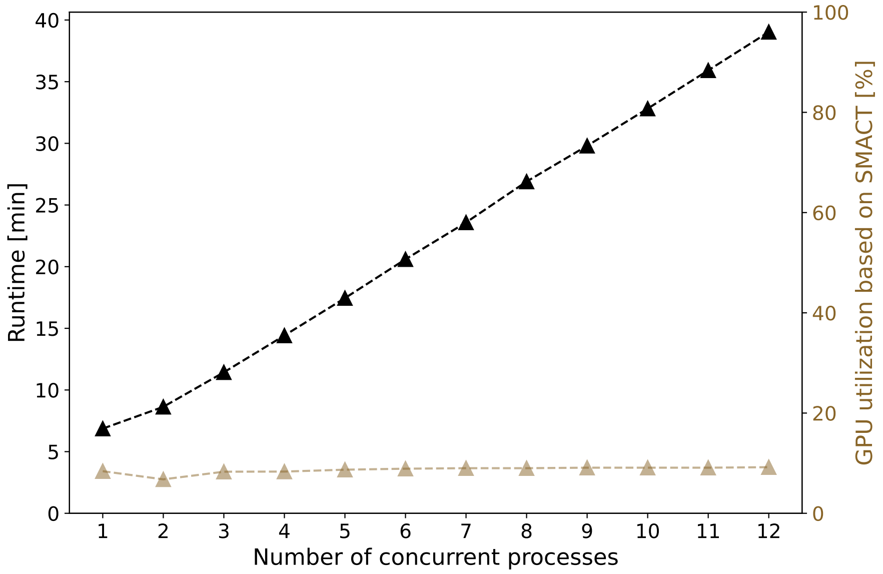


Figure 6.4: Preliminary scaling study of the Time-Slicing approach. All processes shared a single GPU concurrently. GPU utilization was determined via SMACT.

lightweight tasks such as the *MSSM Training* workloads do not continuously submit kernels to the GPU. During CPU-only gaps, other processes can temporarily use the device, which explains why concurrency can still provide benefits.

Compared to exclusive GPU access, monitoring Time-Sliced workloads poses additional challenges. While per-context VRAM usage can be tracked, metrics such as power draw and GPU utilization are only reported for the GPU as a whole. Even aggregate utilization values can be misleading: the commonly used Graphic Activity (GRACT) metric, reported by both SMI [119] and DCGM [118], tends to overestimate true device occupation during multi-tenant use. The SMACT, also reported by DCGM, provides a more representative measure and is used consistently throughout this thesis. Appendix A.8 compares GRACT, SMACT, and Streaming Multiprocessor Occupancy (SMOCC) using the *MSSM Training* workload as an example to illustrate this point.

6.6.2 Multi-Instance GPU

The MIG device configuration is a hardware feature that enables the safe partitioning of certain NVIDIA GPUs, including the NVIDIA A100 [99]. With MIG, a single GPU can be split into multiple independent units, which I refer to as *pieces* in this chapter to avoid confusion with concurrent workload instances. Each piece operates as a logically separate GPU, thereby circumventing the *one context per GPU* restriction encountered with the Time-Slicing scenario. Portions of the GPU that are not allocated to any piece remain inaccessible while MIG mode is active.

Each MIG piece includes robust isolation, including dedicated schedulers, error handling, and monitoring. Because MIG partitioning occurs at the hardware level rather

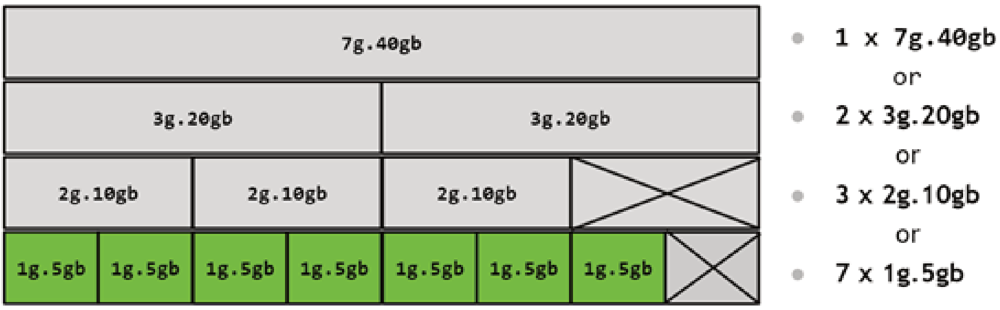


Figure 6.5: Sketch of the possible MIG configurations for an NVIDIA A100 GPU [136]. The first number in a piece name, multiplied by 14, denotes the number of assigned streaming multiprocessors, and the second number gives the assigned VRAM in GB. Pieces of different sizes may be deployed simultaneously, provided there is no horizontal overlap. For example, a valid configuration could include two `1g.5gb`, one `2g.10gb`, and one `3g.20gb` piece at once.

than through the CUDA runtime, it is compatible with non-CUDA libraries. The high degree of isolation ensures that OOM errors are confined to individual pieces and do not affect other pieces, even if they share a host GPU. Processes running on one piece cannot interfere with or even communicate directly with those on another, unlike traditional multi-GPU setups. This provides strong security and robustness, though at the cost of flexibility.

Configurations are limited to those explicitly provided in the official documentation [136]. They allow for little customization, and switching between configurations requires tearing down the existing setup. All involved pieces must be at complete rest for a configuration change, which makes MIG impractical for dynamic deployment scenarios. In this work, I therefore restrict the setups to static configurations that remain unchanged throughout a benchmark.

On NVIDIA A100 GPUs, each device with 40 GB of VRAM and 108 streaming multiprocessors can be split in various ways. Figure 6.5 illustrates the available piece sizes and allowed combinations. As long as there is no horizontal overlap, pieces of different sizes can coexist. Notably, some fraction of the total hardware capacity is lost depending on the chosen configuration.

For performance monitoring, MIG offers distinct advantages over Time-Slicing. While SMI provides only limited access to monitoring metrics, DCGM fully supports MIG. When enabled, DCGM reports GPU-wide metrics and metrics for each MIG piece, allowing more granular monitoring of resource usage.

6.6.3 Multi-Process Service

The MPS software is an alternative, binary-compatible implementation of the CUDA API [135]. It is supported in its current form on all post-Volta NVIDIA architectures, including the A100 and V100 GPUs used throughout this thesis. The central idea of MPS is to merge multiple client GPU contexts into a single unified context, thereby overcoming the *one context per GPU* limitation. Figure 6.6 illustrates how MPS differs from the default Time-Slicing behavior shown in Figure 6.3.

A high-level MPS control daemon manages the creation of MPS processes, with typically one per GPU. Each MPS process can fuse up to 48 client contexts, allowing the GPU kernel scheduler to parallelize workloads across them. If a non-MPS client accesses the GPU, or if multiple MPS processes share one device, these contexts revert to Time-Slicing behavior relative to each other.

MPS provides fully isolated GPU address spaces, ensuring VRAM protection and limited error containment. In practice, however, global errors such as collective VRAM exhaustion still affect all clients operating on a device. A useful feature is that such errors are consistently propagated to all client processes, preventing corrupted results. Isolation is, aided by additional context management options, stronger than in pure Time-Slicing scenarios, but remains weaker than with MIG.

The MPS daemons can be customized via daemon arguments or per-process environment variables. Useful options include:

- `CUDA_VISIBLE_DEVICES`: restricts devices visible to a daemon.
- `CUDA_MPS_PIPE_DIRECTORY`: enables multiple independent daemons, allowing for multiple daemons to operate on the same machine.
- `CUDA_MPS_PINNED_DEVICE_MEM_LIMIT`: set hard VRAM limits for client contexts.
- `CUDA_MPS_ENABLE_PER_CTX_DEVICE_MULTIPROCESSOR_PARTITIONING` and `CUDA_MPS_ACTIVE_THREAD_PERCENTAGE`: constrain the fraction of streaming multiprocessor threads available to a client.

Of these, the `CUDA_MPS_PINNED_DEVICE_MEM_LIMIT` is the most effective for isolation, as it allows for a fine-grained distribution of GPU VRAM across client processes. Processes that exceed their allocated limits are terminated with an OOM error, without disturbing other clients. The mechanism does not guarantee that the sum of all VRAM limits is below the hardware maximum, so careful configuration is necessary.

Using MPS is straightforward, as it only requires the MPS control daemon to run before launching the CUDA processes. Any process with matching `CUDA_VISIBLE_DEVICES` and `CUDA_MPS_PIPE_DIRECTORY` values is then fused with the corresponding MPS context. Earlier implementations restricted MPS to a single user at a time, but since driver version `r555`, multiple users may share a common MPS daemon.

In multi-GPU environments, the documented limit of 48 fused contexts per GPU proved misleading. My measurements indicate that this limit applies to the per-MPS daemon rather than per-GPU. With one MPS daemon managing eight GPUs, performance dropped sharply beyond 48 concurrent contexts despite the spread across multiple devices. Assigning one MPS daemon per GPU restored expected performance and allowed scaling up to 96 concurrent contexts.

In preparation for the whole benchmark, I repeated the scan over the number of concurrent processes on one GPU from Section 6.6.1. Outside of enabling MPS, the setups were identical. The results of both the scan with and without MPS are illustrated in Figure 6.7.

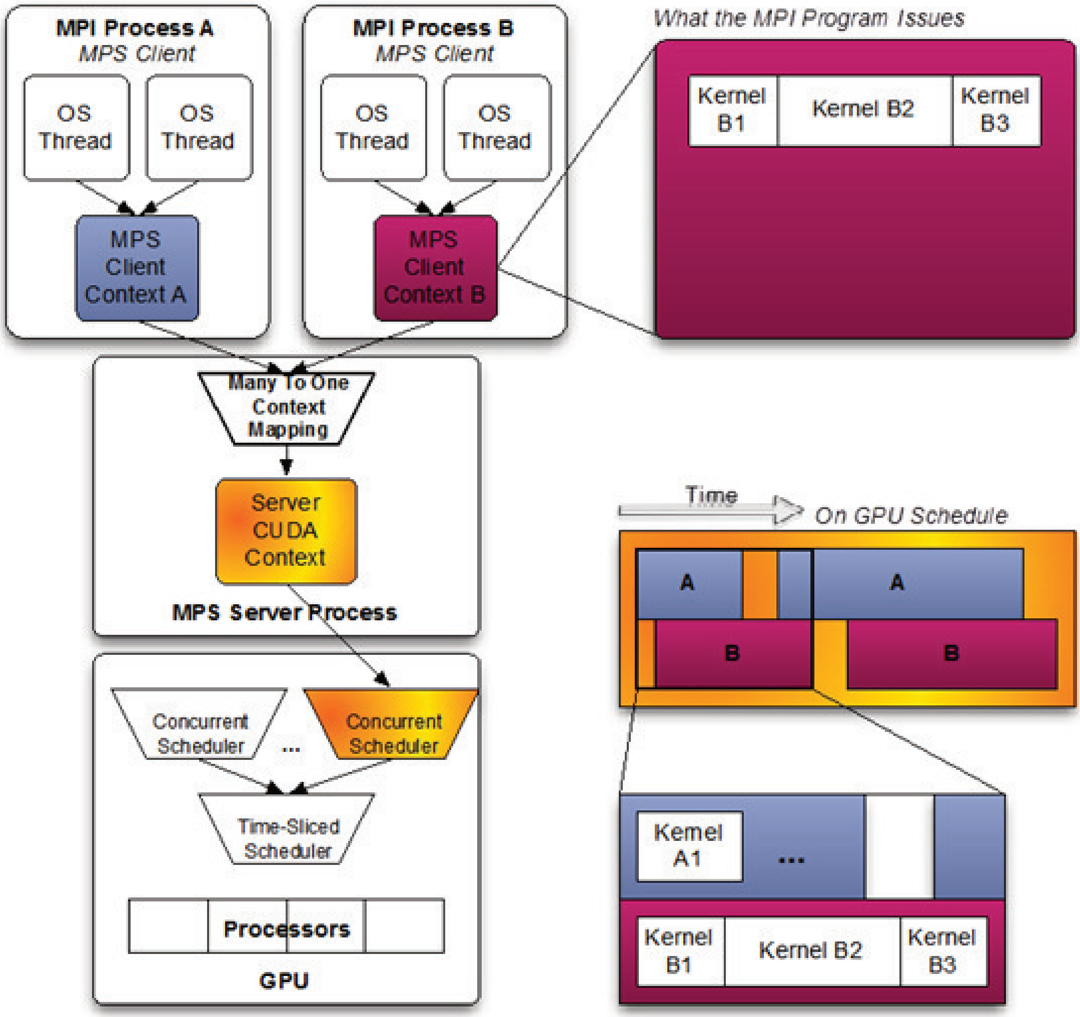


Figure 6.6: Comparison of default Time-Slicing and MPS behavior [135]. In contrast to Time-Slicing, MPS fuses multiple client contexts into a single context, which the GPU then schedules. Within this MPS context, all clients can concurrently submit and execute kernels. Multiple MPS contexts can access the same GPU, though they are again subject to Time-Slicing relative to one another.

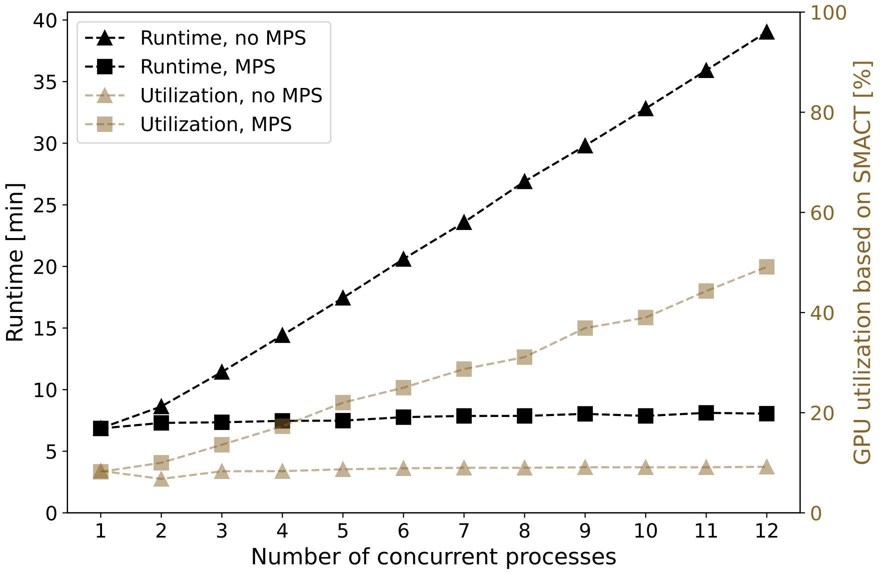


Figure 6.7: Scaling behavior of MPS. All processes were placed on a single GPU, with utilization measured via SMACT. Data from the Time-Slicing scan is included for comparison.

From the plot, I conclude four features:

- The runtime between both setups is identical for a single instance.
- With MPS, runtime remains constant regardless of the number of concurrent instances.
- For 12 processes, MPS achieves a speedup factor of 4.8 compared to Time-Slicing.
- GPU utilization increases linearly with instance count.

These findings demonstrate that MPS removes the overhead of Time-Slicing and does not introduce additional baseline overhead, enabling efficient parallelization up to the hardware scaling limits. The primary constraints remaining are the GPU's VRAM capacity and the per-daemon limit of 48 concurrent processes.

While MPS is compatible with all CUDA libraries, it cannot fuse non-CUDA contexts. Instead, these fall back to Time-Slicing. Monitoring capabilities with MPS are similarly limited as with Time-Slicing: SMI and DCGM cannot distinguish between fused clients, so only aggregate statistics are available. The MPS software cannot provide isolation at the level of MIG as client processes interact directly with one another. Nevertheless, the ability to enforce hard per-process VRAM caps provides stronger isolation than in the Time-Slicing case.

6.6.4 Multi-Tenant Options Summary

By default, multi-tenant GPU usage is implemented via Time-Slicing, which serializes kernel submissions across multiple contexts and therefore scales poorly with large num-

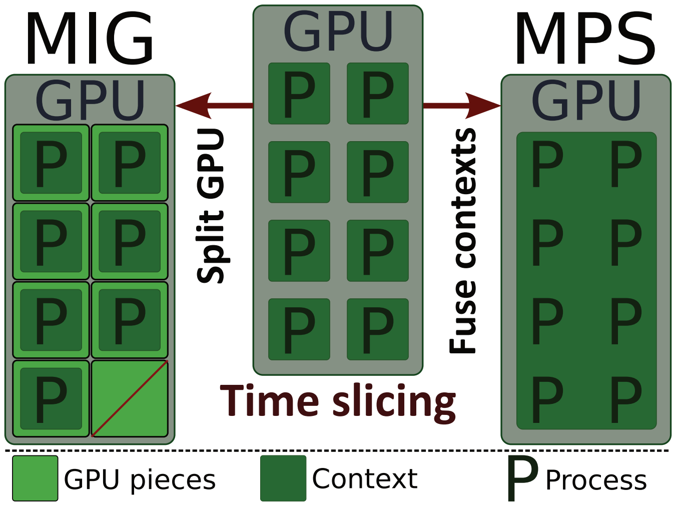


Figure 6.8: Illustration of how multiple processes interact with a GPU under different management approaches. Without MPS or MIG, processes are subject to inefficient Time-Slicing. MIG divides the GPU into separate hardware-isolated pieces, while MPS fuses them into a single monolithic context. Both approaches circumvent the *one context per GPU* restrictions for their managed processes.

bers of lightweight workloads. The alternative technologies, MIG and MPS, both remove the fundamental bottleneck of *one context per GPU*, but do so in opposite ways, as illustrated in Figure 6.8. MIG partitions the GPU into hardware-isolated pieces, each with its own context, whereas MPS consolidates multiple clients into a single MPS context.

The trade-offs among these approaches are summarized in Table 6.3. In short, MPS provides the highest concurrency and flexible resource assignment, at the cost of weaker isolation and less precise monitoring. MIG, in contrast, emphasizes strong hardware isolation and detailed monitoring, but requires static partitioning, a more complex setup, and is only available for a more restricted selection of GPUs. The default Time-Slicing approach remains the simplest option, but performs poorly with multiple concurrent lightweight workloads.

The technologies are not mutually exclusive. For instance, one could run an MPS daemon inside a MIG-created GPU piece while a non-CUDA process Time-Slices with the daemon. In practice, however, such stacked configurations tend to amplify overhead and complexity.

Table 6.3: Comparison of default Time-Slicing, MIG, and MPS in multi-tenant GPU scenarios. Performance and ease of use are largely equivalent across all approaches for single-tenant workloads.

Aspect	Default	MPS	MIG
Concurrency	Low	Highest	High
Resource Partitioning	Dynamic	Dynamic	Static
Isolation	None	Software-level	Hardware-level
Memory Isolation	No	Limited	Yes
Fault Isolation	No	Limited	Yes
Setup Complexity	Low	Moderate	High
Monitoring Granularity	Low	Low	High
Hardware Support	All	Post Volta	Select post Ampere

6.7 Extended Multi-Tenant GPU Benchmark Scenarios

The second set of benchmark scenarios was designed based on the insights from the initial *MSSM Training* runs. Whereas the baseline Exclusive GPU setup restricted the scope to a single workload instance per device, here I explored multi-tenant configurations that allow multiple workloads to share a GPU concurrently. Specifically, I implemented three configurations corresponding to the technologies introduced in Section 6.6, Time-slicing, MIG, and MPS. The overarching goal was to evaluate whether multi-tenant usage could improve overall throughput compared to the Exclusive GPU scenario and to determine how secure such approaches were.

I want to emphasize that the setups described here were constructed for benchmarking in a controlled environment. As such, they are not secure in a real batch system. In particular, Time-Slicing and MPS do not provide complete isolation between processes and rely on external bookkeeping to enforce resource limits. Chapter 7 covers these issues, along with possible mitigation strategies.

6.7.1 Default Time-Slicing

The first multi-tenant scenario employed the default GPU-sharing mechanism: Time-Slicing. Here, multiple instances of the *MSSM Training* workload were placed on the same GPU without additional optimization. This setup used the same hardware and benchmark workload as the Exclusive GPU scenario, but each GPU hosted as many concurrent workload instances as possible.

For the *MSSM Training* workload, this limit was reached at 12 instances per GPU, saturating the available VRAM. With eight GPUs, this resulted in 96 concurrent workload instances, each pinned to two CPU threads. GPU assignment was enforced by setting the `CUDA_VISIBLE_DEVICES` variable to the Universally Unique Identifier (UUID) of the target GPU, with multiple instances receiving the same UUID

6.7.2 Multi-Instance GPU

The second extended scenario leveraged NVIDIA's MIG feature to partition each GPU into smaller, hardware-isolated pieces. One instance of the *MSSM Training* workload was then assigned to each piece, thereby eliminating Time-Slicing limitations and providing strong isolation.

Each of the eight NVIDIA A100 GPUs was divided into seven `1g.5gb` pieces, with 5 GB of VRAM and 14 streaming multiprocessors each, yielding a total of 56 GPU pieces. This configuration incurs a resource overhead: 12.5% of the total VRAM and 9% of the streaming multiprocessors are unavailable due to partitioning [136]. Moreover, the per-piece VRAM allocation exceeded the workload requirement of 3.3 GB, resulting in a mismatch in granularity between hardware partitioning and workload needs.

Consequently, concurrency was reduced to 7 workload instances per GPU, for a total of 56. Compared to the Time-Slicing setup, nearly half of the concurrent instances were lost in this manner. As before, workload instances were pinned using the `CUDA_VISIBLE_DEVICES` variable, though here, each piece is assigned a distinct MIG piece UUID.

6.7.3 Multi-Process Service

The final extended scenario utilized NVIDIA's MPS software layer. Like the Time-Slicing setup, multiple instances of the *MSSM Training* workload were placed on each GPU, though here, their contexts were merged into a single MPS context, thereby avoiding the inefficiencies of Time-Slicing.

The distribution of workloads was identical to the Time-Slicing setup: 12 concurrent instances per GPU, for a total of 96 across eight GPUs, each pinned to two CPU threads. In addition, one MPS daemon was launched per GPU, with process isolation ensured by setting unique values for the `CUDA_VISIBLE_DEVICES` and `CUDA_MPS_PIPE_DIRECTORY` variables at startup. This prevented context fusion across GPU devices and ensured that each daemon only managed the workloads assigned to its GPU.

6.8 Multi-Tenant GPU Results

This section presents the results of the extended multi-tenant GPU scenarios for the *MSSM Training* benchmark. The main objective was to evaluate whether the full compute capacity of modern datacenter GPUs could be exploited even for lightweight GPU workloads. When comparing the three new setups (Time-Slicing, MIG, and MPS) against the initial Exclusive GPU scenario and the CPU baseline, I focus on runtime, total throughput, and energy efficiency.

Table 6.4 summarizes the key metrics for all five scenarios, while Figure 6.9 visualizes runtime, throughput, power draw, and energy efficiency.

Runtime and Throughput The multi-tenant GPU setups consistently outperform both the CPU baseline and the Exclusive GPU scenario in terms of throughput, although

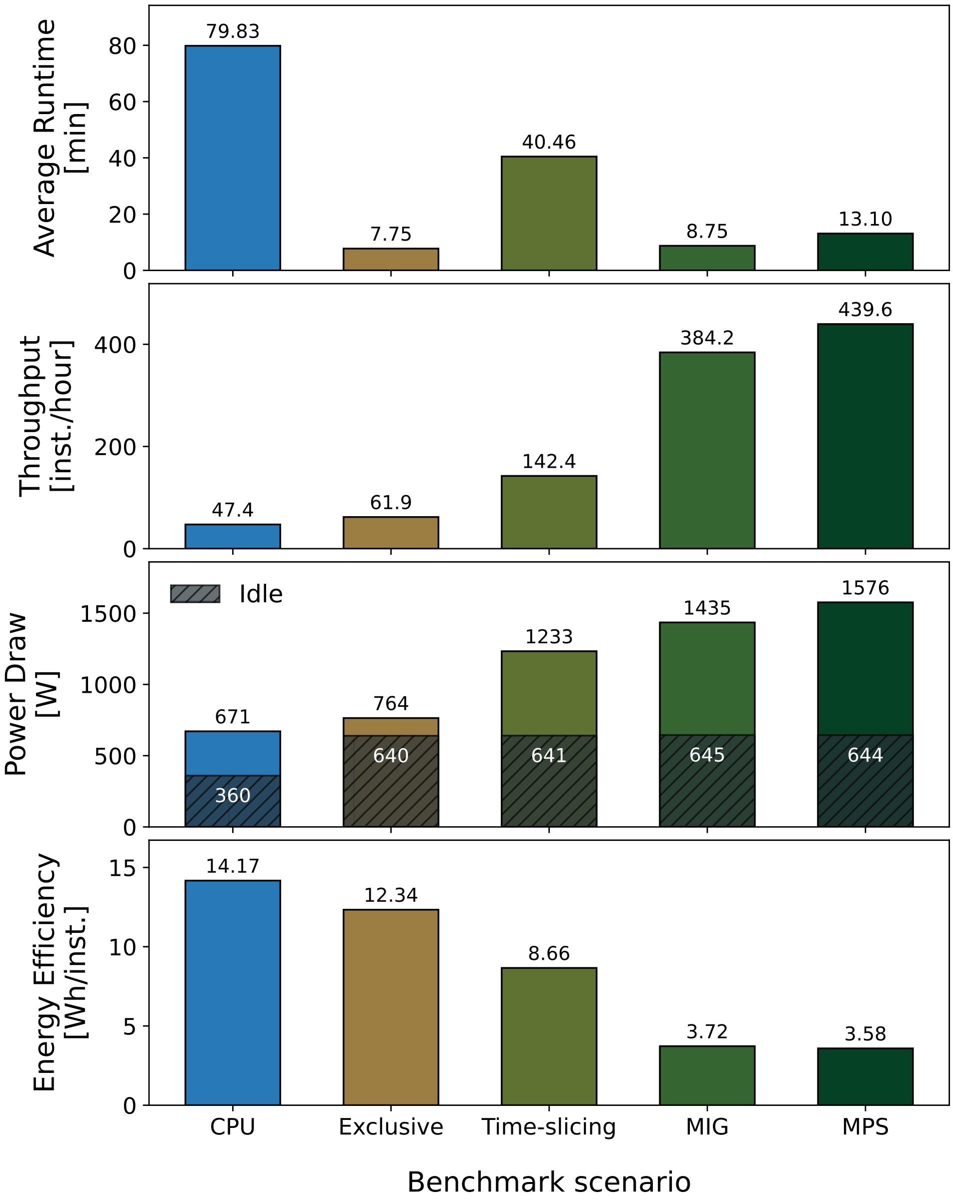


Figure 6.9: Runtime, total throughput, power draw, and energy efficiency for the CPU baseline and the various GPU configurations. Idle GPU power draw is excluded from the CPU baseline to simulate a CPU-only machine.

Table 6.4: Summary of average runtime, total throughput, CPU and GPU utilization, power draw, and energy efficiency for the CPU baseline and the various GPU configurations. Idle GPU power draw is excluded from the CPU baseline to simulate a CPU-only machine.

Configuration	64 × 4 CPU	Exclusive	Time-Slicing	MIG	MPS
# Instances / GPU	—	1	12	7	12
# Concurrent instances	64	8	96	56	96
Mean runtime [min]	79.83	7.75	40.46	8.75	13.10
Throughput [inst. / h]	47.4	61.9	142.4	384.2	439.6
CPU utilization [%]	83.5	65.2	29.0	64.8	66.6
Idle power draw [W]	360	640	641	645	644
Active power draw [W]	671	764	1233	1435	1576
GPU utilization [%]	—	6.7	9.2	15.3	11.5
Efficiency [Wh / inst.]	14.17	12.34	8.66	3.72	3.58

runtime patterns varied. Exclusive GPU, MIG, and MPS scenarios achieved mean runtimes between 7.75 min and 13.10 min, while the Time-Slicing scenario was significantly slower at 40.46 min. Compared to the CPU baseline, even the Time-Slicing scenario achieved a 2× speedup, whereas MIG and MPS setups improved speed by factors of 9.1 and 6.1, respectively. The Exclusive GPU scenario still delivers the shortest mean runtime, 11.4% faster than the MIG setup.

All multi-tenant configurations showed clear throughput gains due to the elevated concurrency. The Time-Slicing setup increased performance by 2.3× over the Exclusive GPU scenario, while the MIG and MPS setups achieved 6.2× and 7.1× increases, respectively.

Resource Utilization CPU utilization across non-Time-Slicing GPU setups was clustered around 65%, while Time-Slicing reached only 29%, reflecting a low concurrency efficiency. Idle power draw remained stable at 640-645 W across all GPU scenarios. Active power draw scaled with throughput, peaking at 1576 W in the MPS scenario, twice that of the Exclusive GPU scenario. GPU utilization rose from 6.7% in the Exclusive GPU setup to 9.2%, 11.5%, and 15.3% for Time-Slicing, MPS, and MIG, respectively.

Energy Efficiency All multi-tenant GPU configurations significantly improved energy efficiency relative to the CPU baseline and Exclusive GPU setups: Time-Slicing reduced the energy per instance by 30% compared to the Exclusive GPU and by 39% compared to the Exclusive GPU and CPU baseline setups. MIG and MPS achieve significantly better results than all other setups, cutting energy usage per instance by over 70% compared to the Exclusive GPU and CPU baseline setups.

6.9 Multi-Tenant GPU Discussion

My extended benchmark scenarios for the *MSSM Training* benchmark reveal substantial performance improvements enabled by multi-tenant GPU technologies. They show both the inherent limitations of CUDA's default Time-Slicing and the advantages offered by MPS and MIG in overcoming them. Most importantly, the results confirm that a GPU can deliver up to 4 times the energy efficiency of a CPU baseline, even for lightweight GPU workloads.

6.9.1 Time-Slicing

The default Time-Slicing approach provided a surprising improvement over the Exclusive GPU scenario. While it resulted in significantly longer runtimes for each instance, the increase in concurrency translated into a notable throughput gain. This demonstrated that even without explicit multi-tenant support, performance can be improved by exploiting gaps in GPU utilization for lightweight workloads.

Runtime and Throughput Mean runtime increased more than fivefold compared to the Exclusive GPU setup, in line with my preliminary scan in Section 6.6.1. Despite this slowdown, the Time-Slicing configuration still outperformed the CPU baseline by a factor of two in runtime, demonstrating the advantage of GPUs even under suboptimal sharing. Still, the prolonged runtime should be considered a major downside compared to other GPU configurations.

With up to twelve concurrent workload instances, the setup achieved 2.3× higher throughput despite the slower runtime. So, while performance was limited by the sequential kernel execution imposed by Time-Slicing, some optimization still occurred. I interpret this as underutilization of the GPU's internal work queue with a single workload, and Time-Slicing helps mask those gaps, leading to steadier, though not higher, utilization.

Utilization The drawbacks of Time-Slicing became apparent in the efficiency metrics. CPU utilization per assigned CPU thread dropped to less than half that of the other GPU scenarios, indicating that GPU constraints, not sequential code, were the main bottleneck. Supporting metrics from DCGM showed the GRACT quickly saturating at 100%, even as SMACT and SMOCC remained low. A detailed comparison of these utilization metrics is provided in Appendix A.8 at the hands of the *MSSM Training* benchmark. Together, these results indicate that while additional concurrent instances raise average utilization, a substantial performance ceiling prevents much of the hoped-for benefits.

Energy Efficiency Power draw increased noticeably compared to the Exclusive GPU setup, though less than for the MIG and MPS scenarios. As a result, gains in throughput were partially offset by the consideration of energy efficiency. Time-Slicing required

29.8% less energy per instance than the Exclusive GPU scenario, but still lagged well behind the MIG and MPS results.

Additional Remarks Time-Slicing represents a compromise between exclusive GPU usage and proper multi-tenant technologies. It offers higher throughput by masking idle gaps in the GPU scheduling, but at the cost of much longer runtimes and modest efficiency gains. In practice, its usefulness depends on whether maximizing throughput or minimizing turnaround time is the dominant priority. These limitations highlight why dedicated multi-tenant mechanisms such as MIG and MPS are necessary to fully unlock GPU performance for lightweight GPU workloads.

6.9.2 MIG

The MIG setup achieved the second-best performance across all metrics, ranking just behind the MPS scenario. It combined fast runtimes with high concurrency, resulting in strong throughput and efficiency gains while maintaining tight isolation between workload instances.

Runtime and Throughput Throughput increased by a factor of 6.2 over the Exclusive GPU scenario, thanks to concurrency levels comparable to the CPU baseline and runtimes close to those of the Exclusive GPU scenario. This combination makes the MIG approach attractive for workloads requiring both swift turnaround and significant throughput.

Utilization CPU utilization mirrored the Exclusive GPU scenario at roughly 65%, indicating that per-instance work was performed at a similar rate, and aligning well with the observed runtimes. The slight slowdown can be explained by CPU boost behavior as the MIG scenario's higher degree of concurrency required a higher aggregate load. Then again, this would imply slight CPU contention, which does not align with less than 100% CPU utilization.

GPU metrics reported by DCGM also raised questions. While SMACT increased compared to the Exclusive GPU and Time-Slicing setups, it reached only 15.3% despite throughput rising more than sixfold. Based on the Exclusive GPU configuration with 6.7% utilization, I would have expected SMACT closer to 40%. I suspect this discrepancy reflects limitations in DCGM's monitoring rather than true underutilization, as throughput and CPU utilization both showed consistent scaling.

Energy Efficiency The power draw of the MIG scenario was higher than that of the Exclusive GPU and baseline CPU scenarios. Yet, throughput gains significantly outweighed this increase in consumption: energy efficiency improved by 3.3× and 3.8× relative to the Exclusive GPU and CPU baseline scenarios, respectively.

Additional Remarks These results confirm my expectation that with proper multi-tenant technologies, the *MSSM Training* workload requires only a fraction of an NVIDIA A100's compute capacity for peak performance. The fact that 56 MIG pieces achieved nearly the same performance as 56 full GPUs used exclusively suggests that each piece is sufficient to support the entire workload. In summary, MIG met or exceeded all expectations: a runtime nearly as short as with the Exclusive GPU setup, resource granularity on par with CPU setups, reasonable power draw allowing for energy efficiency gains, and strong isolation between MIG pieces. Opposed to these boons are the two key limitations: partitioning must be pre-defined, as dynamic reallocation is highly complex, and only a limited subset of GPU hardware supports the technology.

6.9.3 MPS

Among all tested approaches, the MPS setup delivered the strongest overall performance in terms of throughput, efficiency, and flexibility. It also distinguished itself as the only GPU scenario where the eventual bottlenecks originated from outside the GPU itself, demonstrating the extraordinary capacity of MPS to scale concurrent workloads.

Runtime and Throughput The mean runtime under MPS was a third that of the Time-Slicing setup, despite utilizing the same number of concurrent instances. However, it still lagged behind both the Exclusive GPU and MIG setups by a factor of two. This was surprising given the results of my preliminary investigation in Section 6.6.1, which suggested nearly flat runtime scaling under MPS. In that earlier scan, the runtime was mainly independent of the number of concurrent workload instances and only 4% slower than the Exclusive GPU scenario. By contrast, with 96 concurrent instances in the complete benchmark, the runtime increased by 69%. I attribute this degradation not to the GPU, but to CPU contention. Specifically, SMT and the sharing of physical CPU cores appear to have limited throughput. As shown in the extended throughput scan in Appendix A.10, the MPS setup encountered a CPU-originating performance ceiling with the maximum number of concurrent workload instances. This indicates that MPS was capable of driving the GPUs harder than the CPU resources could sustain. I therefore expect that if MIG or another method enabled an equivalent degree of concurrency, they too would encounter similar CPU-induced slowdowns.

Utilization Reported utilization metrics paint a nuanced picture. CPU utilization per instance was nearly identical to that in the Exclusive GPU scenario, suggesting that each instance transferred a similar amount of data to the GPU. However, runtime degradation caused by SMT effects showed that the measured CPU utilization did not fully capture all relevant CPU resource constraints.

GPU utilization scaled poorly relative to throughput. Although the throughput of the MPS setup exceeded that of the Exclusive GPU setup by more than a factor of seven, GPU utilization rose only from 9.2% to 11.5%. Even more puzzling, the MPS scenario outperformed the MIG setup by 14% in throughput despite reporting a lower utilization (11.5% vs. 15.3%). This inconsistency suggests that DCGM-based utilization metrics lose accuracy at high levels of concurrency. While useful as qualitative indicators, they failed to scale reliably with actual throughput in multi-tenant workloads.

Energy Efficiency The power draw of the MPS scenario was higher than that of the Exclusive GPU and baseline CPU scenarios, as well as the MIG setup. Still, as with the MIG setup, throughput gains outweighed this increase in consumption: energy efficiency improved by 3.4× and 4× over the Exclusive GPU and CPU baseline scenarios, respectively. In addition, the MPS setup improves on the MIG setup by reducing the energy requirement per workload instance by 3.8%.

Additional Remarks Despite the many benefits, MPS is not free of limitations. It is tightly bound to both NVIDIA hardware and the CUDA libraries. Unlike MIG, which is integrated into the GPU architecture itself, MPS depends entirely on the CUDA software stack. This restricts portability to other accelerators or programming models and could complicate long-term hardware decisions in heterogeneous environments. Whether comparable solutions will emerge for non-NVIDIA GPUs and non-CUDA libraries remains to be seen.

MPS exceeded expectations by achieving the highest throughput and energy efficiency across all scenarios. It did this so well, in fact, that the benchmark machine ran into GPU-independent hardware limitations. Its performance clearly demonstrated the benefits of fine-grained GPU sharing over Time-Slicing, and to a degree MIG, though its reliance on NVIDIA's ecosystem poses challenges for broader adoption.

6.10 Summary

This chapter covered my benchmark of a lightweight GPU workload, focusing on throughput and energy efficiency. I initially introduced two common scenarios: the CPU-only and the Exclusive GPU approaches.

The CPU-only setup represented the historical approach, as it involved no GPU. It resulted in the longest runtime but achieved acceptable throughput due to a high degree of concurrency. The main upside of this scenario was that it foregoes the costly GPU hardware.

The Exclusive GPU approach is the default setting in current HTCondor batch systems, limiting each GPU to one job. It provided the lowest possible runtime, but at the cost of a minuscule degree of concurrency. As a result, the approach delivered low throughput compared to the theoretical raw compute capacity of the available GPUs. When the cost of procurement is considered, the Exclusive GPU scenario delivers 56% less compute per cost than the CPU baseline. The throughput downside is only partially offset by improved energy efficiency, with 13% lower energy usage per compute instance than the CPU baseline. The setup is generally unsuited for lightweight GPU workloads and leaves most of a GPU's compute capacity underutilized. Power draw was high compared to throughput, as the idle power draw was non-negligible.

Following the initial set of scenarios, I further investigated the possibilities provided through multi-tenant GPU technologies. I introduced three additional configurations that leverage Time-Slicing, MIG, and MPS, respectively, and compared them with the initial approaches.

The Time-Slicing scenario represents the naive approach to multi-tenant GPU use. While the mean runtime increased, the number of concurrent instances also increased. I did not expect any improvement in total throughput for this setup, so I was positively surprised to measure more than twice the throughput compared to the Exclusive GPU scenario. The total power draw also increased, though at a slower rate than throughput. After accounting for procurement costs, the Time-Slicing GPU scenario achieves identical cost per compute to the CPU baseline. The main benefit, therefore, lies in the 39% reduction in energy usage compared to the CPU baseline. The approach has significant organizational downsides, primarily insufficient error protection and instance isolation. I do not recommend this approach if either the MPS or the MIG setups are an option.

The MIG scenario is the most robust and secure of the setups I tested. It boasted outstanding performance, maintaining most of the speed from the Exclusive GPU scenario while achieving up to 7 times as many concurrent instances. Similar to the Time-Slicing scenario, power draw rose compared to the Exclusive GPU scenario, though at a much lower pace than throughput, resulting in significantly reduced energy use per instance. With procurement costs considered, the MPS scenario achieves 2.7× lower cost per compute and a 74% decrease in energy usage compared to the CPU baseline. The main downsides lie in the configuration's rigid setup. In a batch setup, this approach would require frequent reconfiguration, making it unwieldy in its current form. The fixed sizes of the MIG pieces are also relatively coarse and cannot be adjusted. I recommend this approach if available GPUs support it and a mix of GPU piece sizes, appropriate to the local job mix, can be ensured.

The true extent to which a GPU supported by MPS can be pushed remains unclear, as non-GPU factors limited total performance in my benchmark scenario. The results I measured paint a flattering picture, though, as the MPS approach achieved the best performance across flexibility, throughput, and energy efficiency. It exceeded the degree of concurrency achieved by MIG while maintaining most of its speed. While it also drew more power compared to the MIG scenario, it narrowly takes first place in terms of energy efficiency due to its higher throughput. With procurement costs considered, the MPS scenario achieves 3.1× lower cost per compute and reduces energy usage by 75% compared to the CPU baseline. Isolation of MPS-assisted instances is better than with the Time-Slicing approach, but not as good as with MIG. The main downside of MPS lies in its inability to work with non-CUDA libraries. I recommend MPS for systems that either contain GPUs that do not support MIG, or that experience job-mixes too volatile to work with static MIG pieces.

CHAPTER 7

Swarm of Lightweight GPU Applications on a Batch System

Chapter 6 showcased how much of an impact hardware and software configurations have on the throughput of large groups of lightweight GPU workloads. Unlike bare-metal tests, batch systems introduce scheduling effects, security concerns, and competing workloads. Understanding performance in this context is critical, as it is the environment where real production jobs run.

The *MSSM Training* benchmark of Chapter 6 relied on optimistic assumptions to extrapolate throughput from a comparably small number of jobs. This chapter remedies this with a more lifelike setting and with a larger, more realistic set of lightweight HEP tasks. The main goal of this chapter is to verify the performance of the three multi-tenant GPU technologies introduced in Section 6.6 in a batch system. As part of this, I outline the steps required to deploy each approach in a real-life batch system environment. Additional concepts, such as scheduling efficiency, are introduced and the implementation details of the setips are described.

I first discuss the differences between the *MSSM Training* and *NMSSM Training* workloads before introducing the implementations of the four GPU setups (Exclusive, Time-Slicing, MIG, and MPS) in the context of a private benchmark batch system. Afterwards, I discuss the results of the *NMSSM Training* benchmark and compare them with those of *MSSM Training*. Finally, I introduce the MPS-based multi-tenant GPU setup that we use in production as of writing.

7.1 NMSSM ML Training

The workload for a batch system benchmark must satisfy two properties: It should be sufficiently varied and contain enough tasks to occupy the hardware for multiple sequential jobs. Another goal was to keep the overall workload similar to the benchmark workload in Chapter 6 to assess how well the implementations translated to the batch system. The *NMSSM Training* workload I selected for this benchmark fulfills all of these criteria. It consisted of 414 ML tasks, each with a workload similar to that of the *MSSM Training* workload. This specific workload is an integral part of an Next-to-

Minimal Supersymmetric extension of the Standard Model (NMSSM) analysis conducted by Janeck Bechtel [137]. While I do not discuss the NMSSM theory in detail here, comprehensive details, including relevant theoretical background, can be found in the cited thesis.

The NMSSM theory introduces two additional neutral scalar Higgs bosons. If they were to exist, then they would enable the decay of a heavy Higgs boson (H) into the SM Higgs boson ($\mathrm{h_{SM}}$) and a light Higgs boson ($\mathrm{h_S}$), $\mathrm{H} \rightarrow \mathrm{h_{SM}h_S}$. Bechtel's analysis investigates the theory across a two-dimensional grid of mass hypotheses for the two additional scalar Higgs bosons, resulting in 69 groups of mass hypotheses for H and $\mathrm{h_S}$[1]. Furthermore, the analysis focuses on the three most common leptonic decay channels for a $\mathrm{H} \rightarrow \tau\tau$ decay: $e\tau_h$, $\mu\tau_h$, and $\tau\tau_h$ where τ_h denotes a hadronic tau decay.

To perform a two-fold multi-classification for each scenario, two lightweight NN were trained for each combination of mass-hypothesis grouping and decay channel. The total number of necessary training tasks for this workload was therefore:

$$\begin{aligned} &\textbf{69 hypothesis groups} \times \textbf{3 channels} \times \textbf{2 folds} \\ = &\textbf{ 414 neural network trainings} \end{aligned}$$

Each of those 414 NN training sets relied on the same NN architecture as the *MSSM Training* workload: three fully connected feed-forward layers with 512 nodes each. Each training had five output classes, though the number of input features varied between channels. For the $e\tau_h$ and $\mu\tau_h$ channels, 23 features were used, while the $\tau\tau_h$ channel utilized 28. The amount of training data also varied across the 414 training runs, as they relied on both simulated data for the individual mass hypothesis and channel-specific data. Figure 7.1 showcases the distribution of the training data among all three channels, with an average of roughly 590000 events per training. The training data was split between training (75%) and validation (25%) datasets.

Each training batch contained 384 events per input class, for a total of 1920 per batch. Unlike the *MSSM Training* benchmark, the *NMSSM Training* trainings did not use a fixed number of epochs and instead relied on an early stopping condition. Each epoch consisted of 300 training steps, and an early stopping patience of 50 epochs. All training runs were performed with a common seed, resulting in a fixed number of epochs for each specific training. The epoch distribution across all trainings is shown in Figure 7.1, with an average of 164.1 epochs.

As with the *MSSM Training* workload, the focus here was on the computational throughput of the *NMSSM Training* workload, not the physics performance of the trained NNs. I therefore omit a detailed listing of the hyperparameters and refer to the original thesis for additional details.

[1] A note on the exact number of mass hypothesis groups: at some point during development the grouping for the heaviest H mass hypothesis groups was shifted, resulting in an additional group compared to those outlined in the thesis itself. My benchmark utilized the version with 69 groups, not 68.

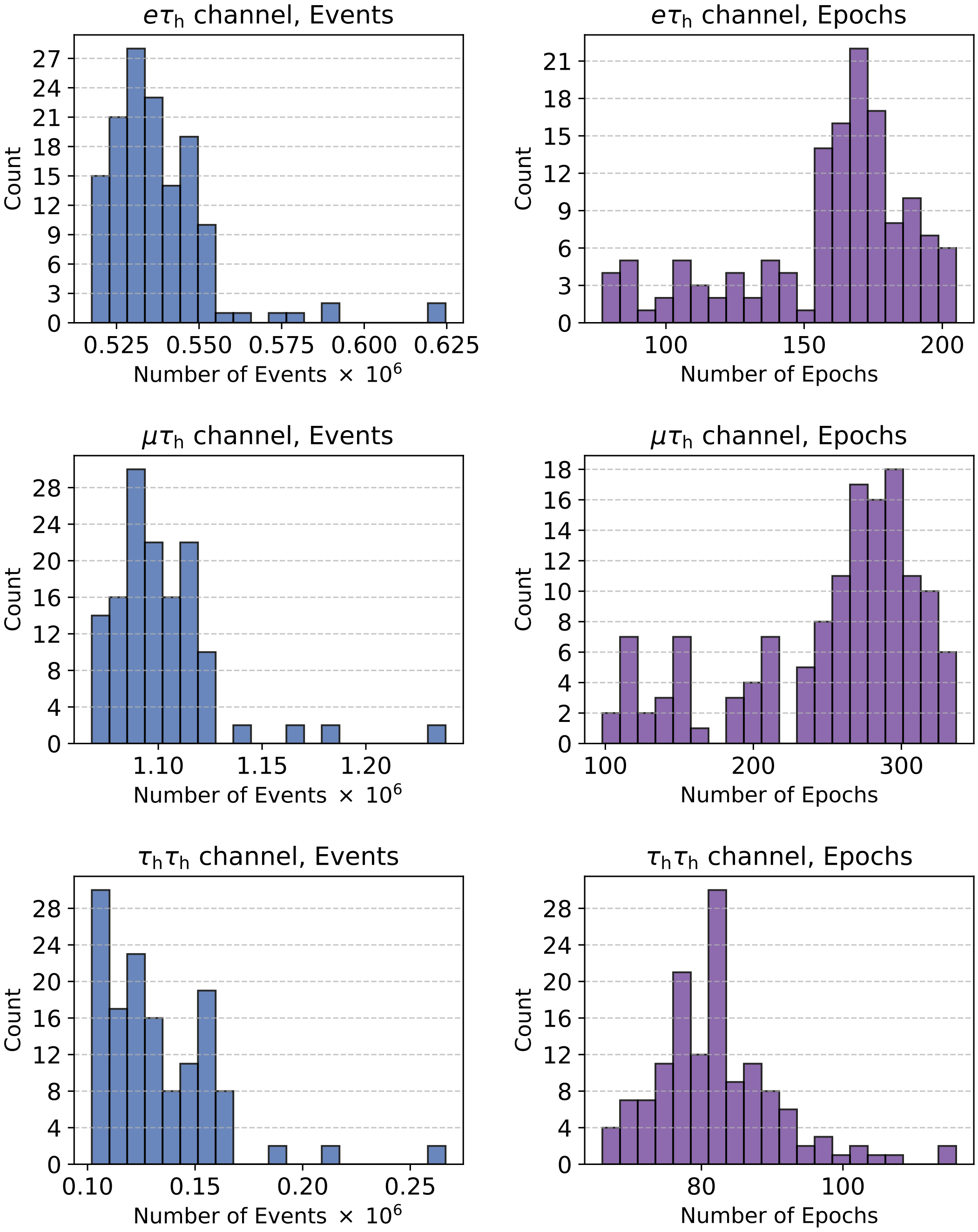


Figure 7.1: Distribution of the number of events and number of epochs for the 414 trainings in the *NMSSM Training* workload. The number of events includes both training and validation data. All training runs used an early stopping criterion that terminated after 50 epochs without improvement in the validation loss.

7.2 Benchmark Metrics

This chapter provides a realistic comparison of the four setups introduced in Chapter 6 in the context of a batch system. The main metrics of the benchmark are the average runtime $\bar{t}$, the scheduling efficiency E_S, and the throughput of the saturated system λ. Combined, they provide great insight into the speed at which work was processed, the degree of concurrency reached, and the processing power lost due to scheduling inefficiencies.

To calculate the metrics, I recorded the start and end times of all $N_J = 414$ jobs. From these timestamps, I derived the number of actively running jobs at any given time, denoted as $n(t)$. The maximum concurrency n_M for each setup was known from my setups outside of a batch system in Chapter 6. I assume that the ramp-up time for the benchmark is negligible, which leaves the following timestamps of note:

- t_0: the time when the benchmark started
- t_L: the time when the last job of the benchmark started
- t_E: the time when the last job of the benchmark ended

The interval $[t_0, t_L]$ is representative of the fully saturated system, as jobs were able to start whenever a job slot was free. In contrast, during the ramp-down interval $[t_L, t_E]$, the number of running jobs continuously decreased, as no further jobs started.

I assume that the potential bias in the job runtimes that still ran during this ramp-down phase had a negligible impact on the average runtime. If the release of resources accelerated job completion, then the finishing jobs would have been expected to cluster towards the end of the ramp-down phase, forming a visually identifiable concave shape. As shown in Figure 7.2, the decrease in active jobs was near-linear, with a normalized area under the ramp-down curve consistently around 0.45-0.5 for all benchmark runs.

With the assumption that runtimes were independent of machine saturation, the mean runtime can be expressed as the average over all job runtimes t_i:

$$\bar{t} = \frac{1}{N_J} \sum_i t_i = \frac{1}{N_J} \cdot \int_0^{t_E} n(t) dt \tag{7.1}$$

The maximum job time at 100% scheduling efficiency is given by the wall time of the saturated period t_L multiplied by the maximum degree of concurrency n_M. The actual observed job time was given by the integral over the number of running jobs $n(t)$ during the saturation time frame, which leads to the scheduling efficiency during saturation:

$$E_S = \frac{1}{t_L \cdot n_M} \cdot \int_0^{t_L} n(t) dt \tag{7.2}$$

The effective average number of concurrent jobs during saturation is therefore given by:

$$\overline{n} = n_M \cdot E_S = \frac{1}{t_L} \cdot \int_0^{t_L} n(t) dt \tag{7.3}$$

Finally, by applying Little's Law [138], the average throughput λ for the *NMSSM Training* workload is calculated, using Equations 7.1 and 7.3:

$$\lambda = \frac{\overline{n}}{\overline{t}} = \frac{N_J}{t_L} \cdot \frac{\int_0^{t_L} n(t)dt}{\int_0^{t_E} n(t)dt} \tag{7.4}$$

To summarize, $\overline{t}$ measures how long individual jobs run on average, E_S measures how efficiently HTCondor was able to assign work to the hardware resources, and λ is the effective throughput (jobs per hour) that combines both aspects.

Compared to the throughput metric used in Chapter 6 (see Equation 6.2), this approach accounts for scheduling inefficiency and a variable, but fixed mix of jobs. Both throughput models converge for an infinite job queue with a job mix representative of the benchmark workload and perfect scheduling efficiency.

7.3 Benchmark Setups

As this benchmark was an extension of the *MSSM Training* benchmark, I refer to Chapter 6 for motivation, theory, and general considerations, and to Sections 6.3 and 6.7 specifically for the four GPU setups outside of a batch system. The focus of this section is to describe how the already proven setups were transferred to work in conjunction with an HTCondor batch system.

The software environment for the *NMSSM Training* workload was based on TensorFlow version 2.8 [130, 139]. The hardware requirements for each of the 414 trainings were similar to the training workload of the *MSSM Training* workload, though I did not examine every single training in detail. Across all trainings, the maximum reported VRAM usage was 3.38 GB, while the average was around 2.35 GB. Based on this metric, I concluded that all individual trainings for the *NMSSM Training* workload could run under the same resource limits as the *MSSM Training* workload.

The hardware in each setup remained the same as in the *MSSM Training* benchmark. One compute node of the TOpAS cluster, described in Section 4.2, with access to eight NVIDIA A100 40GB GPUs. The machine is part of our local HTCondor batch system, allowing us to apply changes to the HTCondor setup as necessary. All benchmark jobs were restricted to run only on that specific machine, and the machine accepted only the benchmark jobs for the duration of the benchmark. Therefore, while the benchmark was conducted as part of a real batch system, it was on an isolated worker node.

To use GPUs as part of an HTCondor batch system, the option `use feature:GPUs` is required in the HTCondor configurations. With it, HTCondor detects the available GPUs and assigns them to their respective jobs by setting the `CUDA_VISIBLE_DEVICES` environmental variable. From a security perspective, this approach is delicate as there is nothing to prevent a user from simply changing the variable to any other value.

The key distinctions of the four setups are:

- **Exclusive**: allows for exactly one job per GPU.

- **Time-Slicing**: allows for multiple jobs to be assigned to the same GPU, but does not optimize usage any further.
- **MIG**: utilizes MIG to split the available GPUs into pieces and assign up to one job to each of the pieces.
- **MPS**: similar to Time-Slicing in that it allows for multiple jobs to be assigned to the same GPU. Further optimizes performance with MPS.

The job setup for all setups was identical. Each of the 414 trainings of the *NMSSM Training* workload was described in an individual job submission file with identical resource requirements. An upper limit to the VRAM usage was added to the submission files in the form of a `request_GPUMemoryMB` value, though this setting only affected the Time-Slicing and MPS setups.

All jobs of a specific setup were submitted at once. During the benchmark evaluation, I found that simultaneous submission had potential performance downsides, as scheduling efficiency was affected by the large number of jobs starting or stopping simultaneously. It is difficult to state whether such behavior is normal for a batch system, as mass job submission is common in HTC batch systems, though mass startup is less so.

7.3.1 Exclusive GPU

The first GPU setup was based on the GPU's default behavior in an HTCondor batch system. Each GPU was assigned to at most one job. In this scenario, all GPUs were part of a single partitionable slot, from which GPU slots with at least one requested GPU were split into dynamic job slots. Since the benchmark machine hosts eight GPUs, this setup supports up to eight concurrent jobs. The Exclusive setup required no additional configuration beyond the `use feature:GPUs` setting in the HTCondor configuration.

7.3.2 Time-Slicing GPU

The second GPU setup was based on the Time-Slicing setup, in which multiple GPU jobs were assigned to the same GPU with no additional optimization. Time-Slicing itself allowed for multiple processes to access a GPU device at the same time, but performance was limited as only one of the processes was allowed to submit work to the GPU at a time. I based my target degree of concurrency on the results of the *MSSM Training* benchmark: with 3.3 GB of VRAM usage, up to 12 instances of the workload were able to fit onto a single GPU with 40 GB of VRAM. For the *NMSSM Training* benchmark, this translates to a maximum of 12 concurrent jobs per physical GPU.

To allow HTCondor to assign multiple GPU jobs to the same GPU without overassigning VRAM, I made four modifications to the HTCondor configuration:

- Each GPU was logically replicated 12 times during the GPU discovery process
- Each physical GPU was assigned a private partitionable slot, from which job slots with access to the logical GPUs were split off into dynamic slots

- Jobs were matched to these partitionable slots based on their VRAM requests
- Job submissions were transformed on the scheduler side to adhere to the new configuration

Each of the eight GPUs was therefore mapped to an individual partitionable slot, each with two types of associated resources: GPUs and GPU VRAM. The GPUs were provided as a discrete resource, replicated up to the maximum number of jobs allowed on a single GPU (12 for this benchmark). This replication allowed HTCondor to assign the same discrete resource to multiple jobs. However, if there were no additional constraints, HTCondor would over-subscribe the GPU without regard to VRAM usage. I therefore introduced an additional resource for the partitionable job slot: the total amount of available VRAM for the GPU mapped to the slot. This resource was not discrete but rather a pooled resource, such as CPU, memory, or disk space. In the case of the NVIDIA A100 GPU, the VRAM limit for each partitionable slot was 40 GB. All other non-GPU resources were handled by the standard HTCondor distribution algorithm, which assigns resources from a partitionable slot to its dynamic slots.

The HTCondor batch system implementation supports exactly one level of partitioning. A partitionable slot can be split into multiple dynamic slots, but it is not possible to split a partitionable slot into multiple smaller partitionable slots. Consequently, this means that multi-GPU jobs also aren't able to function alongside multi-tenant jobs, as it would require nested partitioning. These limitations directly motivated the restriction that machines with multi-tenant GPU slots only accept jobs that request exactly one GPU. In this context, a *GPU* can refer to either a full physical GPU or a logical GPU that grants access to a physical GPU, but not the whole VRAM associated with it. Multiple logical GPUs can map to the same physical GPU, as long as the collective VRAM demands of the active jobs do not exceed the GPU's total resource pool. The limit to single GPU jobs is necessary, as jobs that request multiple GPUs could instead receive the same GPU multiple times.

Jobs that wished to take advantage of the partitionable multi-tenant GPU slots needed to provide information on how much VRAM (`GPUMemoryMB`) they intended to take up. If a job required less than the available VRAM of a GPU mapped to a partitionable slot, it could utilize the setup. Each job started in a dynamic sub-slot of the main slot and had the requested amount of VRAM assigned to it. In this way, the limited VRAM pool across all jobs running on the same GPU prevented the GPU from running OOM.

The main downside of this approach lies in the limited isolation: While a job might have requested a particular volume of VRAM, there was no way to ensure the limit was adhered to. Ramp-up of VRAM usage is often more volatile than CPU memory usage, and total volumes of VRAM are generally much lower than CPU RAM. Fencing techniques like `cgroups` [140] can significantly aid in isolating system resources, but these tools do not exist in the same capacity for GPU resources. If a user misjudges the amount of VRAM required for a job, it could lead to job failure, not just for that job but also for all other jobs assigned to that same GPU. There is currently no mechanism to protect HTCondor jobs in a Time-Slicing setup from one another.

Finally, I considered the default behavior of jobs that do not request any `GPUMemoryMB` but do request a GPU. These jobs needed to be modified to function with the multi-tenant GPU slots properly. Each job that requested a GPU but did not specify how much of the VRAM it would occupy was assigned the total available VRAM on the GPU.

With this setup, multiple GPU jobs were able to run on the same GPU, though job isolation was lacking. If any job understated the VRAM it required, the combination of jobs placed on the GPU could exceed the available VRAM. In such a case, some jobs would be aborted, although there would be no guarantee that the job with the incorrect VRAM request would be among them.

7.3.3 MIG

The second multi-tenant approach was based on the MIG device configurations. I consider the MIG-setup the most secure approach, though in exchange, it was also much less dynamic than the Time-Slicing and MPS setups.

HTCondor is capable of detecting MIG pieces in the same manner as it detects GPUs as a whole. So, once the MIG pieces were set up, this approach was identical to the Exclusive GPU scenario. The main challenge with this setup was the configuration of the MIG pieces. Any amount of reconfiguration required all involved MIG pieces to be at complete rest, including job-independent scripts. Both monitoring software and HTCondor itself include such scripts to provide information on the status and resource utilization of the hardware. All of these processes have to be stopped for the duration of the reconfiguration. In practice, reconfiguration has also frequently failed due to some miscellaneous process still accessing an existing MIG piece. A dynamic setup for MIG would have had to address this issue through a reconfiguration script that ensures two main requirements: MIG pieces must be at complete rest during reconfiguration, and only specific combinations and orders of MIG piece sizes are allowed.

What specific piece sizes and which combination of pieces are allowed is hard-coded by NVIDIA for each supported GPU [136]. Some configurations can also be deployed only in a specific order. For example, if I simplify the NVIDIA A100's MIG configurations to pieces of sizes 1 to 7 with a total capacity of 7, then the setup $1 + 1 + 1 + 1 + 3$ would be valid. At the same time, the combination $3 + 1 + 1 + 1 + 1$ would be forbidden. Such restrictions on the available configuration sets hinder their use in a partitionable slot. Additional logic would be necessary to determine the optimal configuration for the current job requirements, with complex logic to handle the creation of pieces.

Instead of facing this task, it's also possible to only provide static MIG pieces of fixed sizes. This way, the setup is simplified to a one-time configuration, and afterward, the machine is treated identical to a machine with many smaller GPUs rather than a few large ones. The main downside of such usage is that jobs with requirements that exceed the resources provided by the static configuration cannot run on the MIG pieces at all[2].

[2] It is not possible to recombine multiple MIG pieces in the manner of multi-GPU approaches as the MIG pieces are invisible towards each other once set up.

As a whole, the MIG approach is straightforward when used with a static setup, but becomes immensely complex when dynamic resource provisioning were to be involved. For the *NMSSM Training* benchmark, I used the static approach, as it suited all jobs in the workload. Each GPU was split into seven pieces, for a total of 56 MIG pieces, and each piece was detected as an individual GPU by HTCondor, resulting in 56 GPU job slots.

7.3.4 MPS

The MPS setup was closely related to the Time-Slicing setup, but with additional features. In terms of configuration, everything outlined in the Time-Slicing section also applied to the MPS setup. In addition, two more modifications were added to the setup:

- An MPS daemon was set up for each of the GPUs
- VRAM limits were set for each job through the mechanisms of the MPS software

The setup to use MPS was generally straightforward: As MPS fuses all CUDA processes it can detect running on a GPU by default without additional configuration. The maximum number of processes per MPS daemon is 48, so to account for the up to 96 concurrent jobs, multiple MPS daemons were necessary. I set up one MPS daemon per GPU by assigning unique values to the `CUDA_VISIBLE_DEVICES` and `CUDA_MPS_PIPE_DIRECTORY` environment variables. Jobs that were supposed to take advantage of the MPS server need to have their `CUDA_MPS_PIPE_DIRECTORY` environmental variable set to the same values as the respective MPS daemon. Otherwise, the jobs would default to not using any of the MPS daemons and instead rely on Time-Slicing. The `CUDA_MPS_PIPE_DIRECTORY` and `CUDA_VISIBLE_DEVICES` environmental variable were added to the jobs at startup.

This change was sufficient to use MPS as a tool for multi-tenant GPU usage, but MPS also provided additional benefits useful for this benchmark. The `CUDA_MPS_PINNED_DEVICE_MEM_LIMIT` environmental variable allowed for a hard limit to the amount of VRAM that a process, managed by MPS, was able to access. Notably, this limit applies on a per-process basis, not per job. If a job spawned more than one GPU process, then this limit applied to each of them separately. Therefore, the total used VRAM might exceed the requested VRAM. Still, I considered this a significant upgrade over the hazardous handling of VRAM in the Time-Slicing approach, where no limits of any kind were enforced.

To provide a meaningful value to the environmental variable, the setup script read the value of the requested `GPUMemoryMB` from the classadd file of the job[3]. Just as with the other environmental variables, `CUDA_MPS_PINNED_DEVICE_MEM_LIMIT` was then set as part of the job startup. As an example, if the request for `GPUMemoryMB` was set to 3300, then the resulting value for the environmental variable was `${CUDA_VISIBLE_DEVICES}\=3300MB`. The first part of this value denotes the GPU associated with the

[3] The HTCondor setting `JOB_EXECDIR_PERMISSIONS = world` was necessary to access the job information at runtime.

limit, while the second part, after the escaped equal sign, denotes the limit of resources that should be set for the job. This limit can be provided in either `MB` or `GB`.

The setup provided all the benefits of the Time-Slicing approach with additional upsides. As long as users did not change the provided environmental variables, the setup guaranteed sound isolation between jobs and improved performance. The only limitation in this regard was that MPS was bound to CUDA, meaning that jobs that did not rely on the CUDA libraries also did not gain any of the benefits and instead defaulted to Time-Slicing behavior.

7.4 Benchmark Results

This section presents the results of the *NMSSM Training* benchmark, outlined in Table 7.1 and visualized in Figure 7.2. Both the raw numbers and the time series provide valuable insight into the tested multi-tenant approaches. I rely on the Exclusive GPU setup as my baseline in this section, as it describes the default use case with HTCondor.

Runtime With an average runtime of 6.71 minutes, the Exclusive baseline scenario was the setup with the fastest individual job runtime. The MIG and MPS variants had slower average runtimes, by 20.7% and 47.4%, respectively. The Time-Slicing approach stood out from the others, with an average runtime 3.56× that of the Exclusive baseline.

Throughput Due to the significant difference in the number of concurrent jobs across the setups, the total throughput differed widely from the average runtime. The Exclusive setup placed last, with 70.1 completed jobs per hour. The second worst result corresponded to the Time-Slicing approach, which achieved a 3.36× higher throughput. Both variants of multi-tenant optimization improved further on the Time-Slicing setup. The MIG setup took second place with a 5.54× increase in throughput compared to the baseline, and the MPS setup achieved the best results with an 8.11× rise. The direct comparison between the Time-Slicing and the MPS approach showed that enabling the MPS software increased total throughput by a factor of 2.42.

Scheduling Efficiency In terms of scheduling efficiency, the Exclusive Variant lead with 98.7%, tightly followed by the Time-Slicing approach with 98.3% and the MPS

Table 7.1: Summary of the *NMSSM Training* benchmark results. Throughput is calculated as described in Section 7.2.

Setup name	Exclusive	Time-Slicing	MIG	MPS
Max. concurrent inst. n_M	8	96	56	96
Mean runtime $\bar{t}$ [min]	6.71	24.06	8.10	9.89
Scheduling efficiency E_S [%]	98.7	98.3	93.7	97.7
Throughput λ [Jobs/h]	70.1	235.4	388.6	568.7

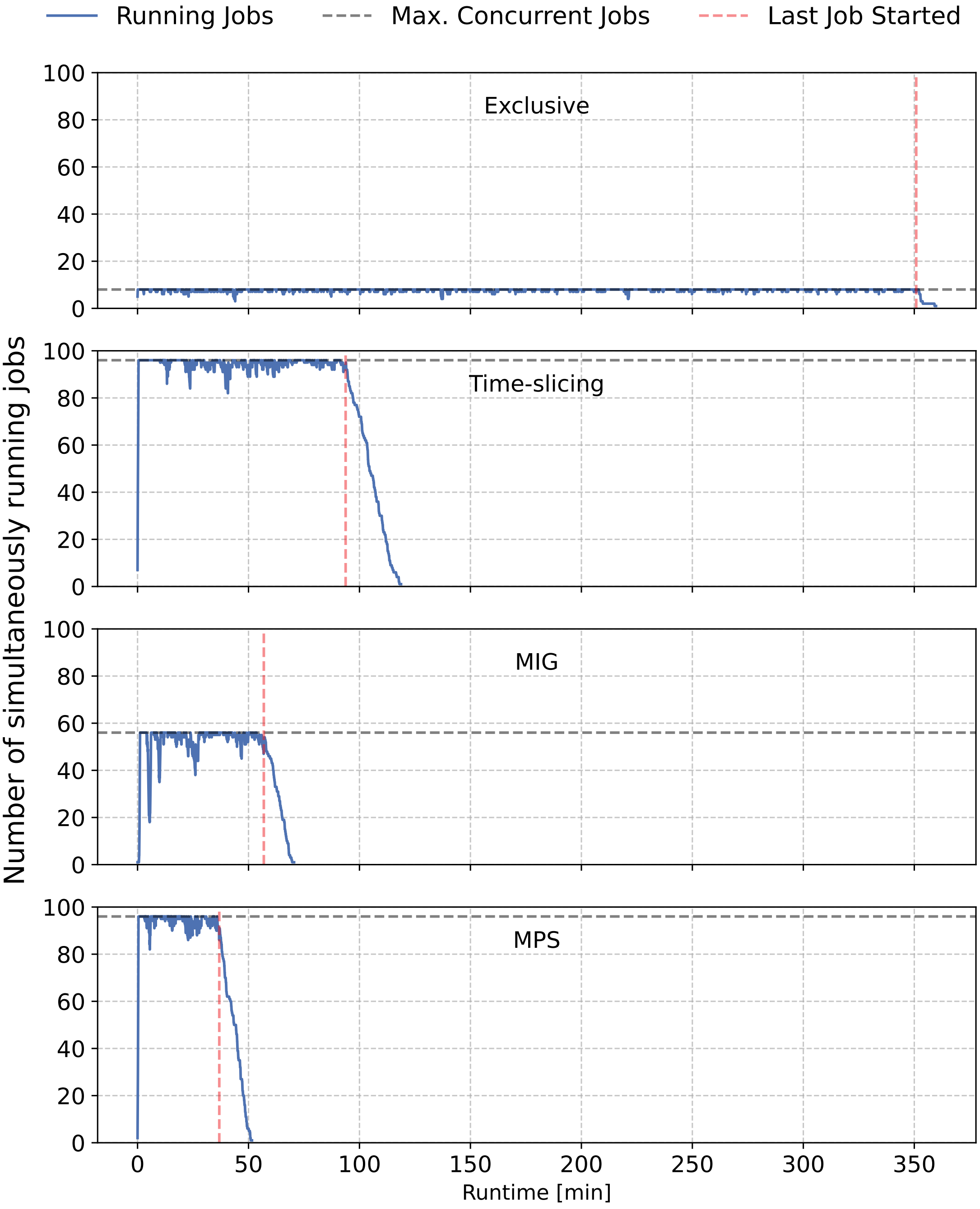


Figure 7.2: Number of concurrent jobs throughout the *NMSSM Training* benchmark scenarios with a total of 414 jobs. The red line denotes the starting time of the final job t_L, and the grey line denotes the maximum number of concurrent jobs possible with the setup n_M.

approach with 97.7%. Further removed was the MIG variant, with 93.7% scheduling efficiency.

7.5 Results Discussion

The *NMSSM Training* benchmark revealed that all the approaches tested on bare metal in Chapter 6 also transferred to the batch system. The results highlight not only raw throughput differences but also how scheduling and resource contention affect each multi-tenant strategy.

Runtime and Resource Contention The average job runtime was correlated with resource contention. The Exclusive GPU setup achieved the fastest average job runtime, as each job had an entire GPU to itself. In contrast, the Time-Slicing setup suffered heavy slowdown, with runtimes more than three times longer than in the Exclusive GPU case, supporting my results from Chapter 6. The MIG and MPS setups fell in the middle for average job runtime: both had to manage some resource contention due to the high parallelism they enabled, resulting in longer runtimes compared to the Exclusive GPU setup. The difference in resource contention is also visually represented in Figure 7.2 as the area under the curve. The greater the area, the slower the average processing time per job.

Scheduling Efficiency Scheduling efficiency was generally high across all setups, with the Exclusive, Time-Slicing, and MPS variants all exceeding 97%, while the MIG setup performed worse at 93.7%. A lower number of concurrent jobs, longer job times, and less clustered job start and end times were preferable for optimal scheduling efficiency. Each setup, except for the MIG setup, had at least one of these properties, which explained the high measured utilization. The Exclusive setup consisted of only a few concurrent jobs, and the Time-Slicing jobs were generally slow. The MPS jobs were less influenced by scheduling inefficiencies because they were slowed down due to SMT effects described in Appendix A.10. This same effect likely also contributed to a greater spread in job runtimes for the MPS setup. The MIG setup was therefore unique in that it offered a high degree of concurrency and a fast individual job runtime, resulting in reduced scheduling efficiency. Still, this should not be interpreted as a downside of MIG, as it indicates only a limitation due to external factors, similar to the SMT limitations the MPS setup experienced.

Throughput Ultimately, total throughput (the rate of completed jobs per hour) is the most critical metric for production use. With a maximum of 8 concurrent jobs, the baseline Exclusive setup finished last despite achieving the fastest average job runtimes. Time-Slicing improved on the baseline setup by a factor of 3.4, but still lagged behind the MIG and MPS approaches. The MIG setup delivered a further improvement of 5.5× the baseline throughput, while MPS achieved the highest overall throughput, exceeding 8× that of the Exclusive baseline. Despite slower individual job runtimes, the optimized

MIG and MPS setups compensated by running at a significantly higher concurrency on the same hardware resources.

Comparison with Previous Workload Compared with the *MSSM Training* benchmark of Chapter 6, I observed generally higher throughput in this benchmark. In the Exclusive scenario, which has both the highest scheduling efficiency and no GPU usage bottleneck, throughput was 13.2% higher in this benchmark. I conclude from this that the difference in sequential runtime between the average *NMSSM Training* job and the *MSSM Training* workload was approximately 13%.

Curiously, other setups experienced much different changes in throughput. The Time-Slicing variant gained 65.3% in throughput compared to the same setup in Chapter 6. My explanation for this lies in the orchestration of the individual processes: in Chapter 6, all processes were started simultaneously for each benchmark run. The part of the workload that sets up and tears down the main ML processes takes up some time but requires few GPU resources. In the benchmark of this chapter, jobs were started mostly at random due to the batch system's scheduling, and non-GPU portions of jobs could overlap with the GPU portions of other jobs. Because the Time-Slicing setup specifically suffered from GPU contention, and performance therefore increased with greater job-level parallelism. While this effect was less pronounced with the SMT-limited MPS scenario, the same effect led to a 29.4% increase in throughput compared to the bare metal benchmark.

The main outlier in this regard is the MIG scenario, whose throughput changed little. GPU resource contention did not exist for this setup in the first place, as individual MIG jobs were isolated entirely on the GPU side. Similar change in throughput as for the Exclusive scenario were expected, but instead, the shift was just 1.1%. While a lower scheduling efficiency can explain part of this discrepancy, the difference is still smaller than expected.

7.6 Summary and Use in Production

In the *NMSSM Training* benchmark of this chapter, I investigated the throughput of various GPU configurations in a batch system when running many small-scale GPU jobs. The jobs contained a set of unique NN training tasks, similar to the *MSSM Training* workload, but with a range of runtimes. The goal of the benchmark was to point out batch-system-specific details that were not bserved in Chapter 6.

All tested batch system variations achieved the maximum degree of concurrency observed in the bare-metal setup. Some of the setups proved more challenging to manage than others. Especially the Time-Slicing variant suffers from a severe lack of resource isolation, which requires users to know exactly how much of the GPU's resources a submitted job would require. Otherwise, one or more jobs may fail if any of them oversteps their non-enforced boundaries.

The fundamentally similar MPS approach partially addressed this issue by introducing process-specific hard VRAM limitations for the jobs. This addition provided a significant degree of isolation between jobs, though a determined user would still be able circumvent these limitations.

The MIG setup was entirely secure on the GPU side, as each MIG piece was registered as an individual GPU with no interaction between the GPU pieces. In exchange, this approach was hindered by its rigidity. While a dynamic use of MIG was theoretically possible, it would have to involve much more management as the other methods due to the complex reconfiguration process. The more straightforward setup, involving fixed-size pieces, had the severe downside of not allowing for jobs that required more than the resources of one MIG piece.

Regarding throughput, all multi-tenant variants improved on the default Exclusive GPU usage. Time-Slicing improved the throughput by a factor of 3.36, while MIG reached an improvement of factor 5.54, and MPS a factor of 8.11, on the same set of hardware resources.

While both the MIG and MPS setups provided excellent performance, the MIG setup was significantly constrained by static setup and hardware limitations. In the end, a slightly simplified version of the MPS setup introduced in Section 7.3.4 was selected for use with the TOpAS cluster and the associated institute batch system. This production setup is described in detail in Appendix A.11 and uses only one MPS daemon process, with a maximum of 8 jobs per GPU instead of 12. While we did deploy the system to all machines with NVIDIA V100S GPUs, we did not deploy it to any machines with NVIDIA A100 40GB GPUs.

Instead, the more capable NVIDIA A100 40GB GPUs are reserved for larger-scale whole and multi-GPU jobs. This decision was based on the fact that most large-scale ML jobs in our system specifically request NVIDIA A100 40GB GPUs due to their additional VRAM. As the name implies, an NVIDIA A100 40GB GPU provides 40 GB of VRAM while the V100S only provide 32 GB each. We considered it illogical to set up GPU machines capable of handling dozens of small-scale GPU jobs by reducing the number of most requested large-scale job slots.

With both MPS and MIG, if any of the multi-tenant GPU job slots were to accept a job, then that GPU would no longer be available for whole or multi-GPU jobs. The machine would only accept additional jobs that still fit within the GPU's remaining available resources instead. We therefore decided to only apply the setup to the machines with the less capable NVIDIA V100S GPUs.

While the benchmark NVIDIA A100 40GB GPUs supported both MPS and MIG, the older NVIDIA V100S GPUs only supported MPS. Even if all of the machines were capable of MIG configuration, the job mix in the institute batch system is not homogeneous enough to motivate a static set of MIG configurations.

The implemented setup has been in use since May of 2024 with no significant issues. During peak usage, up to 113 concurrent jobs were active across the 24 NVIDIA V100S GPUs. I consider this setup a great success, as I was able to transfer the throughput gains from my benchmark first to an isolated batch system and then to our real-life production setup.

CHAPTER 8

Madgraph4GPU, a Scalable HEP Benchmark Candidate

There are two main categories of tasks that occur throughout an HEP analysis: monolithic tasks operating on entire event distributions, and event-based tasks that process events individually. The first category includes workloads such as deriving cut criteria in a cut-based analysis or training NNs to use them as discriminants. These processes are generally monolithic, and cannot be split into smaller chunks without sacrificing accuracy. They also require a representative, but independent distribution of event data to guarantee unbiased physics performance for later application to the event data. As a result, it isn't easy to scale these processes efficiently or dynamically. Chapters 5 to 7 investigated NN training workloads at both large and small scales.

The second category includes all tasks that add data to individual events or filter events based on pre-determined cut criteria. Event reconstruction is one task in this group, but other workloads, such as event simulation and ML inference, also fall into this category. Unlike the monolithic tasks in the first category, these workloads can be split into arbitrarily sized pieces down to a single event. The only information necessary to process the event is either pre-defined or part of hat event. Because of this property, such a workload can run independently, whether sequentially, concurrently, or even across multiple machines.

While previous Chapters concentrated on the training of NNs, these use cases alone are too narrow for a general comparison of CPU and GPU in the context of HEP. The natural next step is therefore to include a benchmark with an event-based and scalable workload, relevant to the global scales of HEP operation. I took this opportunity to utilize the HEP Benchmark Suite [93] developed by the HEPiX Benchmarking Working Group. The Suite can orchestrate a variety of benchmarks, including HEPSpec06 [141] and HEPScore23 [95], to quantify CPU hardware capabilities, with the latter being particularly suited to HEP workloads. The Suite is designed to allow for reproducible benchmarks with rich metadata and optional benchmark metrics. It also supports direct publishing of benchmark results to centralized bookkeeping.

The available benchmark workloads themselves are provided as application containers to ensure consistent environments and exact reproducibility. A multitude of benchmarks are available in this manner, and they can be pulled directly from the gitlab registry associated with the Suite repository. In addition to the fully validated benchmarks, some in-development candidates also exist, including GPU-based benchmarks that are not yet part of any official HEPScore release.

A significant part of event-based software optimization involves exploring novel compute architectures, including vectorized CPU architectures and GPU resources. Resources of this kind do not rely on the single-instruction-single-data compute model like classical CPUs do, but instead exploit single-instruction-multiple-data programming. This approach allows them to operate on multiple events simultaneously, executing the same computational step for each of them, rather than performing the steps sequentially. Due to the measurable increase in both throughput and energy efficiency, vectorized technologies play a crucial role in the journey towards more efficient computing in HEP. During planning for future computing operations, it is therefore necessary to have credible data on which HEP workloads are likely to benefit from such vectorized implementations, be it through CPU or GPU resources.

A workload that explores the opportunities provided through modern hardware is the `mg5amc-madgraph4gpu-2022-bmk` benchmark. It provides a wide selection of CPU vectorization instruction sets and a GPU enabled implementation of the workload. It is based on the GPU implementation (MadGraph4GPU) of the MadGraph5_aMC@NLO event generator software [142], the CPU-only version of which is commonly used in HEP. This benchmark represents a scalable, HEP-relevant workload with both CPU and GPU implementations for direct comparison.

8.1 The gg → $t\bar{t}gg$ Generator

The *gg → $t\bar{t}gg$ Generator* workload consists of the matrix element calculation for a number of events following the default configurations of the `mg5amc-madgraph4gpu-2022-bmk` benchmark container [143]. The benchmark of this chapter utilized versions 1.5 of the HEP Benchmark Suite and version 0.7 of the benchmark container. To my knowledge, this version of the benchmark is not based on a specific release of the MadGraph4GPU package, but instead the most recent commit available at the time of creation [144]. Minor modifications were necessary to utilize the available hardware with the benchmark container fully. The specific code changes are outlined in Appendix A.12.

There are four different layers to the *gg → $t\bar{t}gg$ Generator* benchmark workload:

1. **Benchmark Run**: a complete benchmark with a specific benchmark configuration. Consists of multiple concurrent Copies and sequential Sub-Runs.

2. **Copy**: one of the concurrent Copies that execute the sequential Sub-Runs. Copies are synced between the sequential Sub-Runs and workload Variations. The number of Copies depends on the benchmark arguments and ranges from 8 for GPU to 255 for CPU runs.

3. **Sub-Run**: one pass through the complete set of workload Variations. The number of Sub-Runs per Copy depends on the number of requested repetitions (3 in my Benchmark Runs and by default).

4. **Variations**: one specific workload with fixed parameters. Typically, a Sub-Run consists of only one Variation, but in this benchmark, each Sub-Run encompassed multiple Variations to account for float and double data precision and varying vectorization instruction sets.

The results of the individual layers are aggregated as outlined in Section 8.2.

The number of matrix elements computed in a Benchmark Run depends on three variables, going from the innermost layer outwards: the number of threads per block, the number of blocks per grid, and the number of iterations. While the number of threads per block and the number of blocks per grid describe how many events are processed at once, the number of iterations states how many of these grids are processed sequentially. In my Benchmark Runs, I used the default values for the number of threads per block and the number of blocks per grid, though I varied the number of iterations: 100 for the CPU run and 1000 for the GPU run. The main goal was to achieve a minimum runtime per configuration of roughly 5 minutes to collect sufficient power draw data for a meaningful result.

Four aspects define the Variations: instruction set, target process, precision, and optimization level. For each combination of these aspects, there exists one Variation of the benchmark workload.

Firstly, which instruction sets can be used in the Variations depends on the hardware. There is only one GPU-capable implementation available, which is based on the CUDA libraries and supported by the NVIDIA A100 40GB GPU available on the TOpAS cluster. For CPU, there are multiple possible instruction sets:

- `none`: the non-vectorized, scalar instructions
- `sse4`: the *Streaming SMT Extensions 4* version, capable of 128-bit vectorization
- `avx2`: the *Advanced Vector Extensions 2* version, capable of 256-bit vectorization
- `512y`: the *AVX-512 Ymm* version, capable of 512-bit vectorization with partial vector register usage
- `512z`: the *AVX-512 Zmm* version, capable of 512-bit vectorization with full-width vector register usage

Of these instruction sets, the AMD EPYC 7662 CPUs of TOpAS only supported the `none`, `sse4`, and `avx2` versions.

Secondly, the benchmark uses the matrix-element calculation in the `SigmaKin` kernel for various physics processes. Potential options include `eemumu`, `ggtt`, `ggttg`, `ggttgg`, `ggttggg` and `heftggh`, each representing one process. The gg → $t\bar{t}$gg QCD process (`ggttgg`) is the default target process and the one used throughout all of the Benchmark Runs in this chapter. The process was chosen for its breadth of final-state particles and the significant degree of correlations between spin and color. These properties make the process a demanding computational task, well-suited for vectorization across both CPU and GPU.

Thirdly, the workload is available for both float and double precision, with double precision preferred for matrix element computation. The default on whether float Variations are included has changed multiple times during development. I elected to follow the defaults for the 0.7 version, which includes both float and double precision.

Finally, the Variations' optimization level supports bit, aggressive, and non-aggressive inlining for CUDA and C++ code. I followed the default and only included the Variations with non-aggressive inlining.

The outermost layer of the benchmark, the number of concurrent Copies, depends mainly on the volume of available compute resources. For the CPU setup, I used a total of 255 Copies of the workload, one for each available CPU thread of the benchmark hardware. In the GPU configurations, I limited the number of Copies to the number of available GPUs. For this approach, minor modifications to the benchmark container were necessary. Initially, the container allowed only one GPU to be used by the benchmark, preventing the other seven GPUs installed on the benchmark machine from being used. The necessary changes to the code are documented in Appendix A.12. With these changes, Copies of the GPU benchmark workload were pinned to the available GPUs in a round-robin manner.

8.2 Benchmark Metrics

The benchmark metrics employed in this chapter were similar to those of Chapters 5 and 6. However, some differences exist due to the pre-existing internal score computation.

The original intent was to rely solely on the data from the benchmark report files generated by the HEP Benchmark Suite. While those files contained a wide range of metadata and throughput-related information, some crucial details concerning the exact startup and end times of individual Variations were still missing.

Runtime was less important in this benchmark than in benchmarks with monolithic workloads, such as the *TauTransformer Training* workload in Chapter 5. The runtime of individual matrix element computations was on the order of seconds, meaning the latency per event was small. Instead, throughput and energy efficiency became the sole focus. As total runtime included multiple sources of overhead, this metric was disregarded.

The HEP Benchmark Suite generally reports the results of the benchmarks it orchestrates as a score. For this benchmark, I followed the HEPiX Benchmarking Working Group's standard approach to aggregate benchmark scores and related secondary metrics. In this benchmark, each score point represents a throughput of 10^6 matrix elements per second. A walk-through of the individual layers of the benchmark helps to explain how the sub-scores from a full Benchmark Run combine into a single main score.

For an individual Variation of the workload, throughput is calculated from the non-overhead portion of the total runtime ($t_{\texttt{SigKin}}$) and the number of processed events (N). The Variation score is then calculated by dividing the respective throughput for each sub-score, implementation, Sub-Run, and Copy ($S_{s,i,r,c}$) by 10^6 :

$$S_{s,i,r,c} = \frac{N}{t_{\texttt{SigKin}} \cdot 10^6 s^{-1}} \tag{8.1}$$

The *$gg \to t\bar{t}gg$ Generator* workload focuses on the `SigmaKin` step, meaning that only this step ($t_{\texttt{SigKin}}$) is counted towards the runtime per event. The justification for excluding overhead from the throughput calculation in this way is that all other contributions to runtime are non-essential. Either they represent initialization steps that vanish as the number of iterations increases, or they are part of benchmark-only logging and management.

The `SigmaKin` step accounts for more than 99% of the benchmark's total runtime across all tested CPU configurations. For the GPU implementation, it typically accounts for 93-95% of the total runtime, depending on the precision.

The scores of each Copy of the workload ($S_{s,i,r,c}$) are then aggregated as the sum of their respective scores, as all of them run concurrently:

$$S_{s,i,r} = \sum_c S_{s,i,r,c} \tag{8.2}$$

These scores ($S_{s,i,r}$) then represent the sub-scores for each Sub-Run of the workload, which are aggregated via a median to limit the impact of outlier results:

$$S_{s,i} = \text{med}_r \left(S_{s,i,r} \right) \tag{8.3}$$

At this point, each remaining score ($S_{s,i}$) is a stand-alone sub-score of one of the possible implementations of the benchmark workload. In this thesis, I only combined the scores up to this point to compare them. However, in many cases, a single comprehensive numerical result is needed to facilitate high-level comparisons involving multiple different benchmarks.

A distinction has to be made between the remaining sub-scores. Some of them compete for the best performance, while others represent independent results. In my case, the float and double-precision scores were independent, but the different implementations across hardware and instruction sets competed for best performance. Therefore, only the best result among each of the float and double precision Variations was kept:

$$S_s = \max_i \left(S_{s,i} \right) \tag{8.4}$$

As a final step, the remaining sub-scores (S_s) are aggregated into a single result using a geometric mean to limit the impact of outliers on the final score S.

$$S = \text{gmean}_s \left(S_s \right) \tag{8.5}$$

By combining (8.1) through (8.5) the complete aggregation chain is

$$S = \text{gmean}_s \left[\max_i \left[\text{med}_r \left[\sum_c \frac{N_e}{t_{\texttt{SigKin}} \cdot 10^6 s^{-1}} \right] \right] \right] . \tag{8.6}$$

Notably, the number of events (N) does not need to be the same across all implementations or sub-scores.

In short, each Copy's throughput is aggregated into a Sub-Run score. The best instruction-set Variation is then selected based on the median Sub-Run score. Finally, the float and double-precision results are combined via a geometric mean.

In the case of the three CPU implementations that ran on the hardware, the aggregated scores initially contained an error. While the best score was correctly determined from the implementations, the geometric mean was then applied to all eight sub-scores, the original three sub-scores of the implementation, and the best sub-score, for both float and double precision results. As a result, the aggregated score across all CPU Variations under-reported performance by 84.2% for the CPU results. This issue did not affect the results of this thesis, as I directly compared the sub-scores rather than the final aggregated score.

The second metric of interest in this benchmark was the power draw. Similar to the measurement method outlined in Section 4.4, total power draw was measured using the IPMI software [115] every 10 seconds through the Benchmark Suite's internal plugin system. Notably, while the plugin supports various aggregation methods, such as the mean or quantiles of the recorded values, it does so only across the entire timeline of Sub-Runs. As with all previous benchmarks, the power draw of the Benchmark Runs was determined by the 85% quantile of their power draw time series during the Benchmark Run.

Finally, both CPU and GPU utilization were measured using the same plugin system as for the power draw. The plugin reported the machine's total CPU and GPU utilization, not just the utilization associated with the benchmark workload. In practice, the offset in utilization was at most one active CPU thread, as no other work was performed on the machine during the Benchmark Runs and only standard background processes ran. The utilization metrics were collected over the entire runtime of the Benchmark Run, as with the power draw, and aggregated using the 85% quantile of the metric time series.

8.3 Direct CPU to GPU Comparison

The benchmark consisted of two main runs, one based on the default CPU implementation and one based on the default GPU implementation. As outlined in Section 8.1, each of the two runs contained three Sub-Runs, each with individual sub-workloads. On the benchmark hardware, the CPU defaulted to run a total of six Variations per Sub-Run:

- `ggttgg-sa-cuda-d-inl0`: no vectorization and double precision
- `ggttgg-sa-cuda-f-inl0`: no vectorization and float precision
- `ggttgg-sa-cuda-d-sse4`: streaming SMT Extensions 4 vectorization and double precision
- `ggttgg-sa-cuda-f-sse4`: streaming SMT Extensions 4 vectorization and float precision

- `ggttgg-sa-cuda-d-avx2`: advanced Vector Extensions 2 vectorization and double precision
- `ggttgg-sa-cuda-f-avx2`: advanced Vector Extensions 2 vectorization and float precision

All of these Variations shared the number of threads per block (256), the number of blocks per grid (64), and the number of iterations (100), for a total of 1638400 events per Variation.

The GPU defaulted to just two Variations instead, both of which relied on the CUDA implementation of the workload:

- `ggttgg-sa-cuda-d-inl0`: Double precision
- `ggttgg-sa-cuda-f-inl0`: Float precision

The default number of blocks per grid was increased to 2048, while the number of threads per block remained the same at 256. Because of the faster runtime per iteration, I raised the number of iterations from 100 to 1000 to achieve a comparable runtime to the quickest CPU Variations. The total number of events per Copy was 524288000, 32 times more than in the CPU scenario. The machine could accommodate 255 Copies of the CPU workload, the number of available GPUs limited the number of Copies to 8 for the GPU workload (1 per GPU). After combining the throughput of all concurrent Copies for the CPU and GPU runs, each iteration contained nearly the same number of events (off by 1/256). Each GPU Copy relied on exactly one CPU thread to transfer data and to instruct its respective GPU. While individual Copies in the GPU setup necessarily were pinned to the GPUs, they were not pinned to any CPU resources.

8.4 Benchmark Results

Table 8.1 shows the scores and secondary measurements of the four CPU- and GPU-based setups, and Figure 8.1 presents the critical metrics visually.

A clear trend in throughput is observed. Furthermore, a significant difference in power draw between the CPU and GPU configurations is visible.

CPU Throughput Focusing on the CPU results first, the scores for both float and double-precision Variations increased over the non-vectorized default. For both float and double precision, the increase was nearly linear with the degree of vectorization available through the respective implementations. Compared to the non-vectorized case, the 128-bit vectorized `sse4` Variations delivered a speedup of factor 3.87 (1.88) for float (double) precision. The 256-bit vectorized `avx2` instructions improved on this result with a 7.63 (4.07) times increase over the non-vectorized float (double) precision result.

CPU Energy Efficiency Because the power draw values were reported as identical across all CPU instruction sets, power efficiency followed the progression of throughput for each implementation.

Table 8.1: Benchmark results of the $gg \to t\bar{t}gg$ *Generator* benchmark. The first three columns describe the three CPU Variations with different degrees of vectorization. The 85% quantile is applied to the power draw and utilization metrics of the entire timeseries of the Benchmark Run. GPU idle power draw was removed from the CPU power draw values to simulate a CPU-only machine. CPU and GPU utilization describe the absolute utilization across all CPU threads (256) and GPUs (8).

Configuration	`none` CPU	`sse4` CPU	`avx2` CPU	GPU
Score [10^6 **Events / s**]				
Double precision	0.200	0.376	0.815	6.299
Float precision	0.215	0.831	1.641	14.219
Power Draw [**W**]				
Idle	357	357	357	649
Active	565	565	565	2482
Efficiency [**Score / W**]				
Double precision	0.355	0.667	1.444	2.538
Float precision	0.381	1.471	2.907	5.729
Utilization [%]				
CPU utilization	25601	25601	25601	863
GPU utilization	—	—	—	782

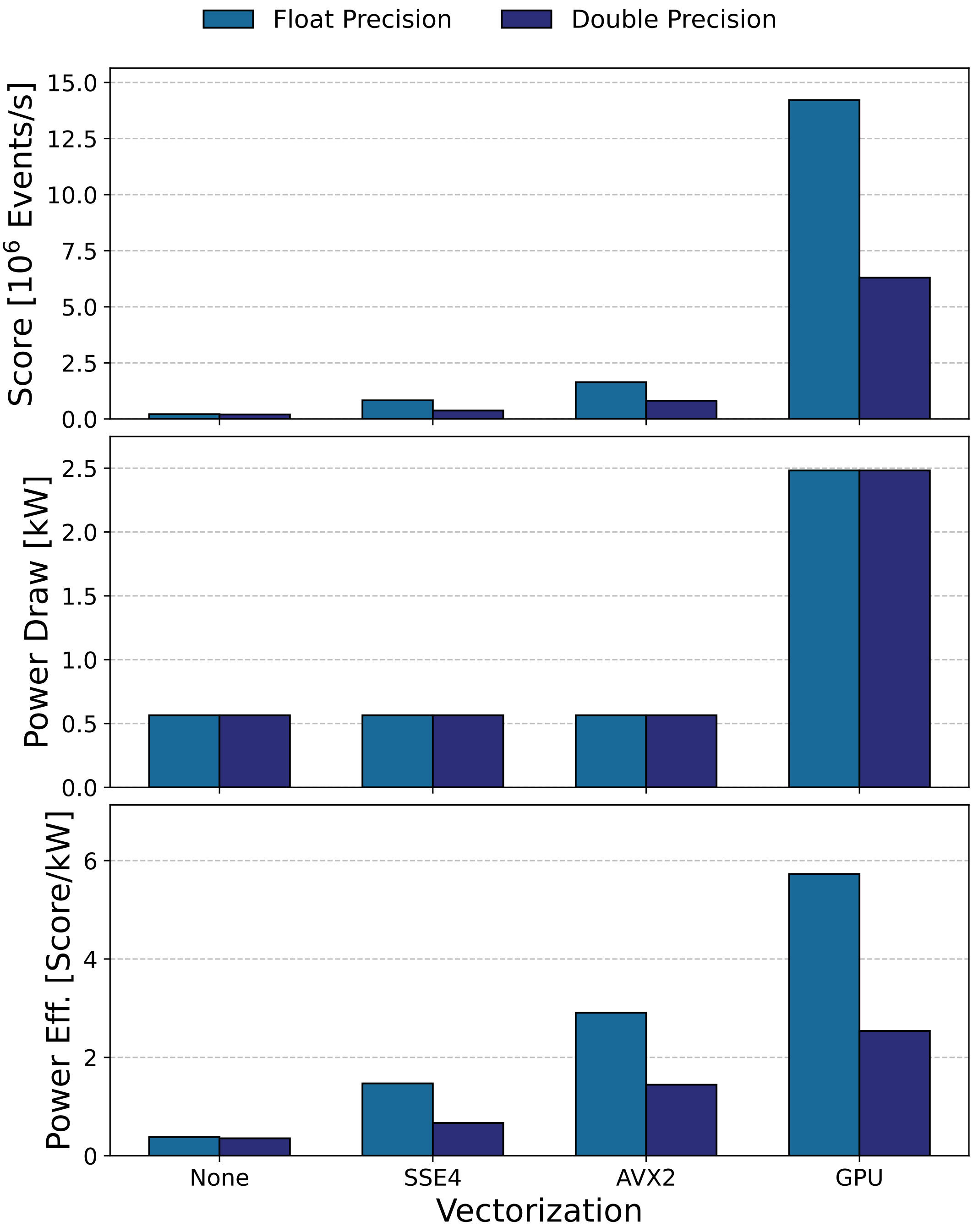


Figure 8.1: Throughput, Power draw, and Power efficiency of the $gg \to t\bar{t}gg$ *Generator* benchmark configurations. The three CPU configurations show the results for the non-vectorized and vectorized options described in Section 8.1. GPU idle power draw was removed from the CPU power draw values to simulate a CPU-only machine.

CPU Utilization Similar to the power draw, utilization metrics were reported as identical across all CPU Variations and indicate that the majority of the $gg \to t\bar{t}gg$ *Generator* workload fully utilized the available CPU resources, as the reported utilization was maxed out.

GPU Throughput Compared to the CPU implementation, the GPU Variations achieved significantly higher throughput despite a drastic reduction in concurrent Copies. The GPU run achieved an 8.66 (7.73) times improvement in throughput for float (double) precision, when compared to the `avx2` result.

GPU Energy Efficiency In terms of power consumption, the GPU-only run exceeded the `avx2` run by 4.39×. While the GPU implementation's throughput was far ahead of the CPU-only result, much of this lead was lost when energy efficiency was considered. The float (double) Variation of the GPU configuration still improved on the best CPU results by 97.1% (75.8%).

GPU Utilization The CPU utilization metrics showed behavior similar to that of the CPU-only configuration. CPU utilization exceeded the total number of Copies placed on the machine by less than 10%. The GPU utilization was high for this workload, as 97.8% of the available GPU capacity was active throughout the benchmark.

8.4.1 Extended Benchmark Results

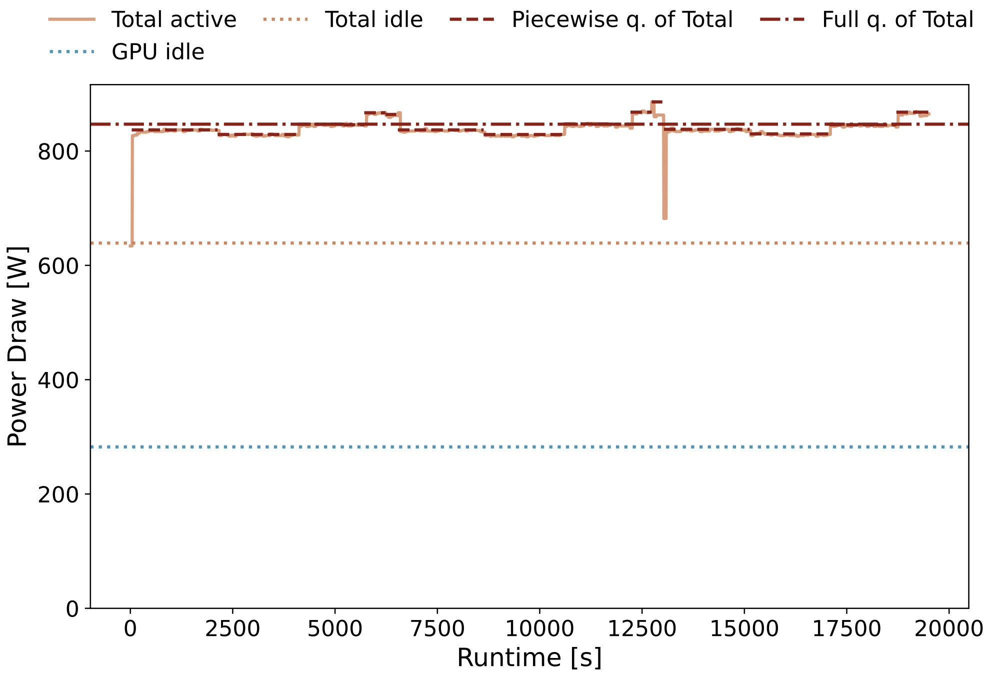
While the sub-scores of the individual Variations were reported on in great detail in the benchmark report file, secondary metrics like power draw and utilization posed an issue. The report stated the start and end of the Sub-Runs in a Benchmark Run, but it did not indicate when the Variations that led to the individual throughput scores occurred.

Motivated by discrepancies in the power draw time series observed for the CPU and GPU Benchmark Runs in Figure 8.2, I revisited the benchmark logs for a more detailed examination. It should be noted that examining low-level logging files is generally tedious and prone to data loss during cleanup. In that sense, this section aims to motivate the necessity for more detailed reporting on the workload Variations during the individual Sub-Runs.

In addition to the raw power draw time series data, Figure 8.2 also includes 85% quantiles across the whole data range, as well as piecewise quantiles based on the Variations of the Sub-Runs. The fact that the two sets of quantiles are not identical is most evident in the GPU run, where the difference between the full and piecewise quantiles can reach up to 300 W.

Table 8.2 and Figure 8.3 outline the results based on the more accurate power draw and utilization values and compare them to the results of Table 8.1.

By aligning power draw data with the timing of the individual workload Variations, the picture became more refined:



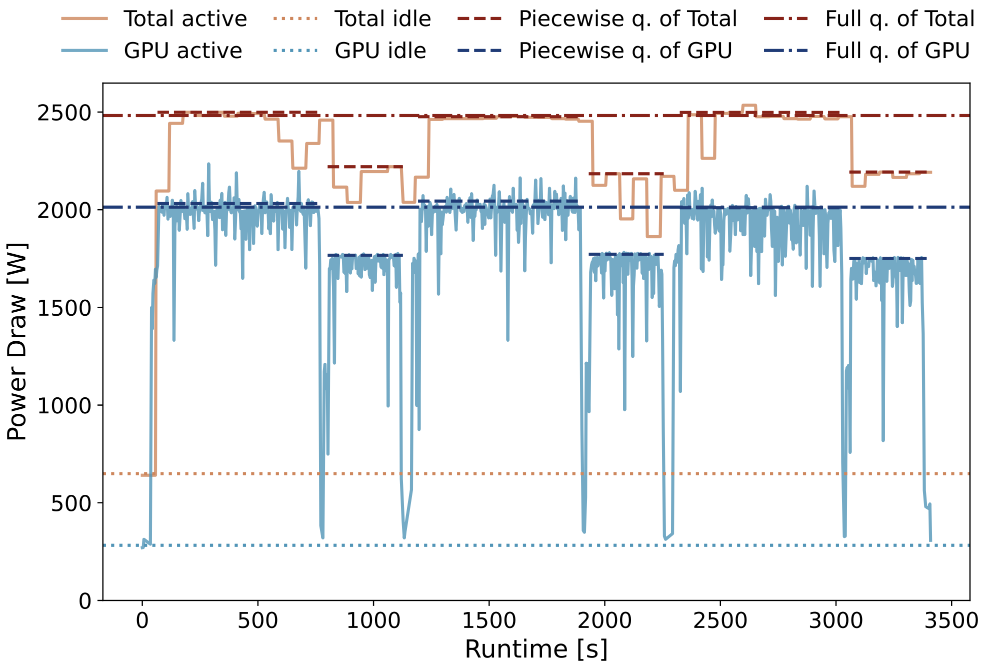


Figure 8.2: Recorded power draw data of the $gg \to t\bar{t}gg$ *Generator* workload in the CPU-only (upper) and GPU-accelerated (lower) Benchmark Runs, with idle power draw (dotted). The 85% quantile of the active power draw is shown for the entire duration of the runs (dot-dashed) and split by individual workloads and run repetitions (dashed). Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.

Table 8.2: Benchmark results of the $gg \to t\bar{t}gg$ *Generator* benchmark with piecewise quantile. The first three columns describe the three CPU Variations with different degrees of vectorization. The 85% quantile of the power draw and utilization metrics is based on the time frames of the specific Sub-Run Variations. GPU idle power draw was removed from the CPU power draw values to simulate a CPU-only machine. CPU and GPU utilization describe the absolute utilization across all CPU threads (256) and GPUs (8).

Configuration	`none` CPU	`sse4` CPU	`avx2` CPU	GPU
		Score [10^6 **Events** / s]		
Double precision	0.200	0.376	0.815	6.299
Float precision	0.215	0.831	1.641	14.219
		Active Power Draw [W]		
Double precision	555	565	585	2498
Float precision	547	564	586	2193
		Efficiency [Score / W]		
Double precision	0.361	0.667	1.392	2.521
Float precision	0.394	1.474	2.803	6.484
		CPU Utilization [%]		
Double precision	25577	25566	25551	867
Float precision	25568	25554	25518	847
		GPU Utilization [%]		
Double precision	—	—	—	786
Float precision	—	—	—	771

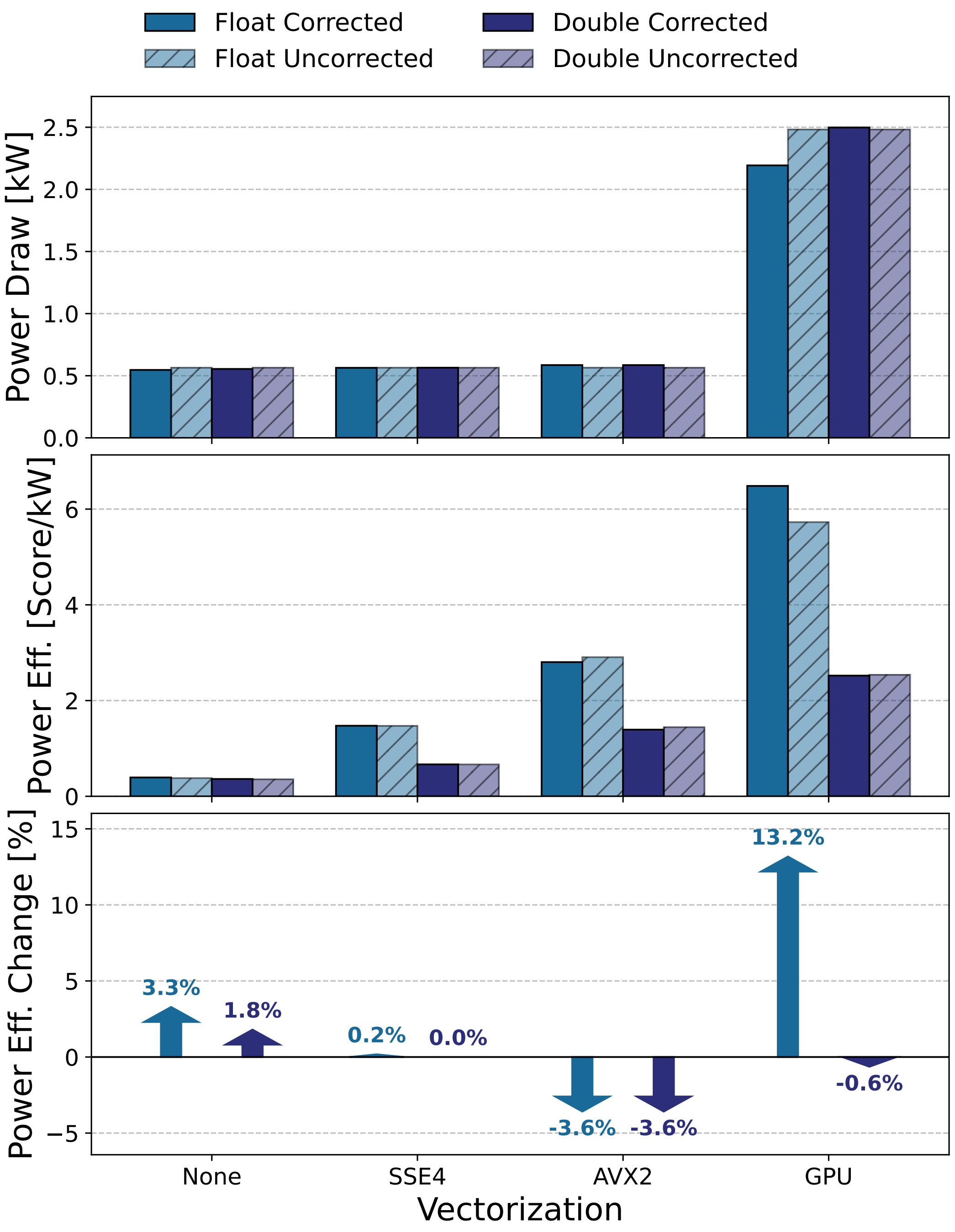


Figure 8.3: Corrected and uncorrected Power draw and Power efficiency of the $gg \rightarrow t\bar{t}gg$ *Generator* Benchmark Runs. Also shows the relative change in the power efficiency between the uncorrected and corrected values. The three CPU Variations show the results for the non-vectorized and vectorized options described in Section 8.1. GPU idle power draw was removed from the CPU power draw values to simulate a CPU-only machine.

- Observed fluctuations in the power draw time series were a direct result of the multiple benchmark Variations.
- A small, but consistent difference in power draw between the Variations exists for both CPU and GPU configurations.

In the CPU-only case, some of the six Variations were hard to distinguish because of their short runtimes and similar power draws. In the GPU case, double and float Variations were easily distinguishable by their offset in power draw.

CPU Power Draw All Variations with vectorization draw slightly more power. The difference between the `sse4` and non-vectorized power draw was minute, with a 3% (2%) difference for float (double) precision. For the `avx2` Variations, the differences reached a non-negligible 7% (5%) for the float (double) precision Variation. The most significant change in this regard occurred for the GPU Variations, where the relative difference between the double and float Variations was 14%.

The results shifted considerably compared to the initial global power draw results stated in Table 8.1. Previously, all CPU Variations shared a single power draw value, which was pulled down by the long, lower-intensity power draw of the non-vectorized Variations. Compared with the initial values, the `avx2` Variations shifted 3.6% downwards in efficiency, while the non-vectorized Variations shifted upwards by 3.4% (1.8%) for float (double) precision. The `sse4` variant remained nearly unchanged.

GPU Power Draw For the GPU Variations, results shifted in different ways, as in contrast to the CPU run, the GPU run contained only the two Variations with different precisions. The 85% quantile was initially chosen to cut off rising and falling edges in the benchmark metrics. In this run, it went much further and cut off all values associated with the float-precision Variation when applied across the whole time series. The power draw of the double-precision Variation mainly remained unchanged, with a less than 1% increase, but the float-precision Variation's power draw drops by over 13%.

Energy Efficiency The changes to the measured power draw also affected the power efficiency, as the scores of the individual Variations remained unchanged. The more accurate treatment of the individual workload, therefore, changed the direct comparison of the energy efficiency between the CPU and GPU results by a considerable margin. With the `avx2` results corrected downwards and the GPU results mostly corrected upwards, the GPU results improve on the best CPU results by 131.3% (81.1%) for the float (double) precision workload, up from the initial 97.1% (75.8%).

Utilization While power draw was not the only metric for which the piecewise evaluation should be used, it was the only one for which a large discrepancy occurred between the uncorrected and corrected values. Both CPU and GPU utilization, shown in Figure 8.4, shifted slightly, though the aggregated difference in the values was less than 1% in both cases.

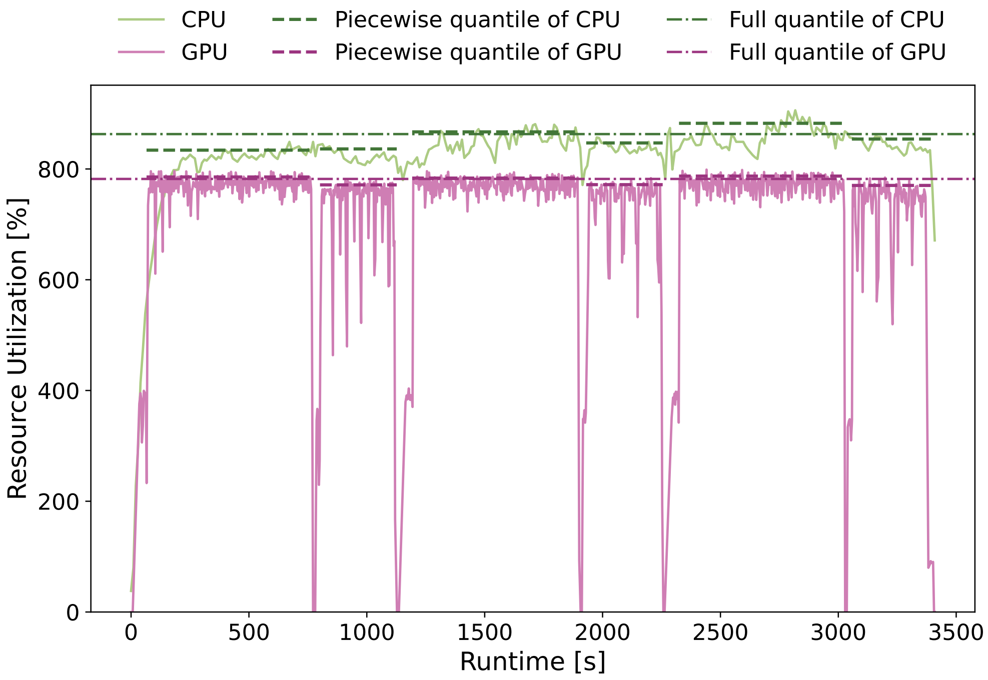


Figure 8.4: Recorded utilization data of the *gg* → *t̄tgg Generator* workload in the GPU-accelerated Benchmark Run. The 85% quantile of utilization is shown for the entire duration of the runs (dot-dashed) and split by individual workloads and run repetitions (dashed). Green lines represent the CPU utilization, while magenta lines represent the GPU utilization.

Final Remarks The correction to the data handling outlined in this section has led to a more representative result for the individual Variations of the *gg* → *t̄tgg Generator* workload. This should be understood as a motivation for more detailed time information on the Variations' start and stop times. An alternative solution would be to restrict each benchmark to a single Variation of a workload, resulting in a single benchmark score. In either case, the main benchmark scores and the sub-metrics collected by the plugin feature must be directly connectable with one another.

8.5 Results Discussion

The *gg* → *t̄tgg Generator* benchmark revealed impressive speedups from both the vectorized CPU and the GPU implementations of the workload. These increases were also associated with a significant increase in GPU power draw, resulting in a smaller but measurable improvement in energy efficiency.

CPU Vectorization Benefits The results of the CPU-only Benchmark Run highlight the importance of vectorization. Both `sse4` and `avx2` implementations improved over the non-vectorized baseline by factors of approximately 4× and 8× for float precision, and 2×

and 4× for double precision, respectively. These improvements align with expectations: `sse4` instructions with 128-bit vectorization process up to four 32-bit floats or two 64-bit doubles simultaneously, while `avx2` instructions with 256-bit vectorization process eight floats or four doubles. The observed throughput of the vectorized implementation, therefore, roughly matches the theoretical gains expected for the instruction sets.

Precision Effects on CPU Power and Efficiency The power draw for float and double-precision variations of the vectorized implementations displayed was nearly identical. This balance can be explained by the ratio of work per event nearly canceling out with the ratio of throughput (e.g., 1.641 vs. 0.831 for `sse4`, and 0.815 vs. 0.376 for `avx2`). As a result, both float and double precision Variations effectively executed a similar amount of work. The non-vectorized CPU Variations, however, showed equal throughput across precisions. At the same time, double-precision still incurred higher computational effort, leading to higher power draw. These findings confirm that the CPU implementations fully utilized the implemented vector instructions.

CPU Utilization and Power Characteristics Despite consistently high CPU utilization, close to 100% of available threads, the measured power draw differed slightly between Variations. This discrepancy suggests that while all CPU threads were active, they were not equally occupied across instruction sets. The `avx2` implementation in particular drew the highest power, reflecting its ability to saturate compute resources fully. The up to 7% increase in power draw was well invested for the up to 7.6× throughput gain achieved with vectorization.

GPU Throughput and Efficiency The GPU Variations achieved a mean 8.2× higher throughput compared to the `avx2` CPU implementation. The float precision GPU Variation delivered a 2.25× higher throughput than the double precision Variation, deviating from NVIDIA's expected 2× ratio for A100 GPUs [99]. The imbalance was compounded further for the energy efficiency metric. Due to increased power draw, the float Variation was 2.57× more energy efficient than the double Variation, compared to the near-2× ratio observed for the CPU Variations. These results suggest that performance was lost in the double-precision GPU Variation, though overall, the GPU setup still outperformed the CPU setup considerably in both throughput and energy efficiency.

GPU Utilization Behavior A closer inspection of the GPU configuration prompted further research. Both power draw, shown in Figure 8.3, and GPU utilization, shown in Figure 8.4, display downward spikes, indicating periods of lower GPU activity. I suspected that this was caused by the internally sequential structure of the $gg \to t\bar{t}gg$ *Generator* workload: While `SigmaKin` calculation were offloaded to the GPU, remaining sequential steps left gaps where the GPU was comparably idle.

Supporting evidence that the GPUs were not utilized to the degree implied by the quantile of their utilization is twofold. Firstly, the MadGraph4GPU CUDA code is limited to the default stream [145], preventing overlap of memory transfers and kernel execution. Secondly, the CPU utilization shown in Figure 8.4 showcases that CPU utilization remained stable, unlike the flickering GPU utilization. These factors suggest that a CPU-side bottleneck limits GPU performance, similar to the experiences in Chapter 6.

While I researched the possibility of performance gains through additional parallel GPU Copies, the results were inconclusive. Appendix A.13 describes the complete argumentation for and against the validity of the results observed in this manner.

8.6 Summary

This chapter presented the benchmark results for the *$gg \to t\bar{t}gg$ Generator* workload, in which I investigated the throughput and energy efficiency of both CPU- and GPU-based implementations for calculating matrix elements in Madgraph. Between non-vectorized and vectorized CPU instruction sets, GPU, and float and double-precision variations, a total of eight versions of the workload were compared. Both vectorized CPU and GPU implementations demonstrated immense leads over the non-vectorized CPU implementation.

Due to hardware limitations, I was unable to test the available 512-bit vectorization Variations. Among the CPU Variations I did test, the vectorized Variations that relied on the *Advanced Vector Extensions 2* instruction performed best. It achieved 7.63 (4.07) times higher throughput than the non-vectorized CPU Variation for float (double) precision. Due to a slightly higher power draw than the non-vectorized CPU Variations, energy efficiency was marginally lower. At 7.11 (3.86) times better power efficiency for float (double) precision, the results were still impressive.

I then compared this already improved result to the GPU-based implementation, which demonstrated another significant leap in throughput and efficiency. Across eight GPUs, an increase in throughput of 8.66 (7.73) for float (double) precision was measured compared to the best CPU result. In other words, just one NVIDIA A100 GPU, assisted by a single CPU thread, matched the machine's total throughput without the GPU. When the cost of procurement was considered, the GPU Variations achieved a 2.9 (2.6) times lower cost per compute than the `avx2` CPU-only results. In terms of power efficiency, the difference was less significant, due to the near 2 kW additional power draw from the active GPUs. Still, the GPU Variations achieved 2.31 (1.81) times better power efficiency for float (double) precision than the vectorized CPU Variations.

While I originally intended to rely solely on the benchmark result file for evaluation, I had to abandon this approach as critical time information was missing from it. Without information on when specific Variations of the workload occurred, I was unable to calculate accurate secondary metrics, such as power draw. In the most extreme case, this resulted in a 13% deviation between the initial and corrected values.

Finally, I attempted to improve GPU performance by assigning multiple Copies of the workload to each GPU, supported by the MPS software introduced in Chapter 6. While I did measure an increase in the score by nearly a factor of two, I could not be certain whether this result represents a fundamental change in throughput. Therefore, the result should not be included in the final GPU performance. Additional investigation is necessary to determine the true nature of these measurements.

All findings stated in this chapter have been communicated to the HEPiX Benchmarking Working Group responsible for large-scale benchmarks in HEP. These results motivated improvements to the *$gg \rightarrow t\bar{t}gg$ Generator* benchmark setup, and highlighted limitations of the HEP Benchmark Suite at the time.

CHAPTER 9

Estimation of Resource Usage

The importance of resource efficiency and the value of rigorous benchmarking have been central themes in this thesis. The insights gained guide our approach to code optimization, resource management, and future hardware acquisitions. At the same time, such studies incur a measurable computational cost. This chapter documents the computing resources utilized during both the NN trainings presented in Chapter 3 and the GPU benchmarks discussed in Chapters 5 through 8.

The computational cost of this thesis is presented in three ways: in terms of required CPU and GPU compute time, the CPU-to-GPU ratio, and the power draw associated with the studies. I introduce the CPU-to-GPU ratio here to illustrate not only the practical resource requirement but also the balance between supporting CPUs and utilized GPUs.

I separated my calculations into GPU benchmarks that ran on a private machine and the NN training runs, which were performed through the institute's batch system. My estimates of power consumption throughout the studies were based on the power draw values recorded during the benchmarks.

For the NN trainings of Chapter 3, which were run through the HTCondor batch system, resource attribution per job was straightforward. Estimating power draw, however, is more difficult, since the metric was not explicitly recorded during regular operation of the batch system. I estimated the power draw based on the *TauTransformer Training* benchmark results from Chapter 4, though these estimates are rough for several reasons.

Firstly, not all runs of the *TauTransformer Training* were equally demanding, as mentioned in Section 3.7. Some of the runs relied on modified training data, leading to lower GPU utilization, and I expect that this, in turn, would have lowered power draw. In addition, some runs also relied on more complex pre-processing than others, affecting the precise power draw of the training process.

Secondly, some of the training jobs ran on different hardware from the NVIDIA A100 GPUs utilized in Chapter 5. Both the NVIDIA V100 and V100S GPUs installed in the TOpAS cluster were used for training the various TauT models. While these GPUs have the same power-consumption limit, they are older models than the A100 GPUs and are likely to exhibit different power-to-compute behavior.

Finally, while the benchmark runs in Chapter 5 were conducted in isolation, the

Studies	CPU-to-GPU ratio	GPU time [h]	CPU time [h]	Energy usage [kWh]
TauT physics perf.	16	20000	320000	$\lesssim$ 8372
TauT computing perf.	16	550	8800	122
Lightweight GPU	2-24	430	6880	63
Lightweight Swarm	2-24	550	8800	81
Madgraph4GPU	1	470	7520	162
Total	—	22000	352000	$\lesssim$ 8800

Table 9.1: Estimated resource use throughout the thesis and respective CPU-to-GPU ratio of the workloads. GPU and CPU time, as well as Energy usage, are estimations and, in the case of the TauT physics performance study, an upper limit. The listed entries relate to the physics performance study of the TauT architecture, the computational performance study of the TauT training, the study regarding lightweight GPU processes inside and outside of a batch system, and the study of the MadGraph4GPU-based HEP benchmark candidate.

training jobs were part of regular batch system operation. The exact composition of the workload on the machine was therefore uncertain, and the number of concurrent jobs on a single machine was generally less than 8. So, even if power draw had been measured for the TOpAS worker nodes, proper attribution would remain difficult.

To account for these uncertainties, I state an upper limit on the energy required if the conditions of the benchmark scenario were met. I assumed each job drew 322 W, corresponding to one-eighth of the power draw measured for the 1 × 8 benchmark scenario in Chapter 5.

For benchmarks on a private machine, all 8 NVIDIA A100 GPUs and 128 physical CPU cores were unavailable to others during each benchmark. Even if a specific benchmark run did not rely on a particular hardware resource, that resource was therefore still considered occupied, as no other work could be performed on it. Because I did not measure the power draw throughout the *NMSSM Training* benchmark of Chapter 7, I estimate the power draw based on the performance recorded for a similar workload outside of a batch system in Chapter 6.

The cooling efficiency was provided as 1.3 [117] and included in my calculation for effective energy usage. Based on these assumptions, Table 9.1 shows the CPU-to-GPU ratio, and summarizes GPU and CPU usage, as well as the corresponding energy usage. These results highlight two main points: firstly, the balance between GPU and CPU resources is highly dependent on the application. Between ML at small and large scales and the event-based workloads of global-scale computing operations, the CPU-to-GPU ratio varied from 1 to 24. For ML-based applications, this ratio was a direct result of the limitations in preprocessing and data loading. For the non-ML MadGraph4GPU application, the CPU only delegated work, saturating the GPU resource with a single assigned CPU.

This ratio imposes significant constraints on HEP computing centers that aim to serve both sides of the physics applications landscape, as CPU and GPU saturation must be balanced. On the one hand, an HTC with insufficient CPU resources risks underutilization of the GPUs, which may exceed the rate at which the limited CPU resources can provide data to them. On the other hand, an HTC with significant CPU resources available risks underutilization. Still, an oversupply of CPU resources poses a lesser risk, as idling CPUs can be occupied by additional CPU-only batch jobs, which are not an option for GPU jobs.

Secondly, the price of an efficiency benchmark for a production-type application is small compared to its impact on total resource usage at scale. The GPU hours of the TauT physics performance study were 36 times higher than the GPU hours of the TauT computing performance study. At this ratio, even a 3% increase in efficiency is enough to amortize the resource investment.

All efficiency benchmarks have identified trade-offs and potential areas for improvement in the respective HEP applications. The *TauTransformer Training* benchmark directly impacted how I implemented my large-scale ML effort for the TauT physics performance study. The *MSSM Training* and *NMSSM Training* benchmarks led to modifications to our institute batch system, enabling more efficient job handling in the future. The results of the $gg \to t\bar{t}gg$ *Generator* benchmark were conveyed to the HEPiX Benchmark Working Group, and a new version of the benchmark has since been released.

Together, these findings show that benchmarking is not only an academic exercise but a necessity for strategic planning. The wide variation in CPU-to-GPU ratio underscores the need for heterogeneous computing centers in HEP. At the same time, the relatively low cost of efficiency benchmarks compared to production-scale workloads highlights the outsized impact of even modest efficiency improvements. These insights are essential for effective hardware procurement and for guiding code optimization in both current and future HEP computing environments.

CHAPTER 10

Conclusion

The focus of this thesis has been the expanding role of GPU hardware in High-Energy Physics (HEP), motivated by the jump in computational demands for the upcoming High-Luminosity Large Hadron Collider (HL-LHC). The studies presented here address two central questions: What HEP research has become possible due to emerging GPU technologies, and how impactful is GPU availability to energy and data efficiency across the broader HEP computing spectrum?

In the first half of the thesis, I developed an improved version of the TauTransformer (TauT) Neural Network (NN) architecture, proposed as a potential successor to the current DeepTau classifier for hadronically decaying tau leptons (τ_{h}). Performance gains were achieved by using an extended training dataset and a more generous convergence condition, enabled by the GPU resources of the local Throughput Optimized Analysis System (TOpAS) cluster. The same training configuration was found to be roughly two orders of magnitude slower on CPU hardware, underscoring that these studies would not have been computationally feasible without GPU acceleration.

The improved False Positive Rate (FPR) of Working Points (WPs) relevant for the Compact Muon Solenoid (CMS) detector was lowered by factors of 9 and 7 for electron and muon backgrounds, respectively, relative to the current standard algorithms DeepTau V2.5. The performance against the jet background was largely retained, with an at most 25% increase in FPR. Discrimination against all backgrounds was improved compared to previous studies of the TauT architecture. The TauT also demonstrated robustness and reproducibility, with a Receiver Operating Characteristic (ROC) variance below 6% beyond extreme True Positive Rate (TPR) regions. These results emphasize the potential of training workflows with GPU access at scale to push the performance frontier for tau identification.

To quantify changes in performance for WPs along ROC curves, I introduced a modified power-law model that links NN improvements to their physics impact. In this model, improvements to the underlying NN manifest as offsets along both TPR and FPR axes, and performance scaling depends on the explicit shape of the ROC curve. Consequently, even modest NN performance gains may yield substantial physics performance gains, depending on the steepness of the ROC curve in that region.

I further conducted a series of ablation studies to quantify the impact of training data volume, training patience, and token pruning. These parameters were selected to preserve the TauT architecture, ensuring that adjustments would not affect inference latency or throughput and making them highly valuable for achieving high performance at low computational cost. The results revealed several practical procedures. In isolation, training data scaling was more cost-effective than extending early-stopping patience, with scaling exponents as high as 0.6. Still, with the necessary preprocessing for the additional training data accounted for, extended patience was the computationally least expensive option. Modifications to the input particle data were found to risk introducing biases when cuts were feature-specific. In contrast, random token reduction degraded performance when thresholds were set low, while high thresholds retained performance, but allowed for only modest gains in computational performance. These findings illustrate a balance between requirements for optimal exploitation of the experimental data and motivate investment in modern computing resources.

In the second part of the thesis, I extended the scope to the broader landscape of HEP workloads to survey the throughput and energy-efficiency gains enabled by GPU hardware. This included several GPU application cases, as the training of the TauT model, as well as a representative selection of simulation and analysis applications. These provide a comprehensive view of GPU utilization across the diverse HEP computing landscape.

The studies compared CPU and GPU performance across single-, multi-, and fractional-GPU configurations, highlighting scaling characteristics across a broad range of workloads. After optimizing the data-loading scheme for the TauT training, multi-GPU execution achieved a 3.67× runtime speedup on eight GPUs. The remaining inefficiencies were primarily due to I/O bandwidth and synchronization overhead at larger scales. These measurements quantify the trade-offs between computational speed, throughput, and scaling efficiency that are critical to HEP computation at the global scale.

Particular attention was given to the challenges of small-scale GPU applications, which often fail to saturate available hardware thoroughly and therefore exhibit poor energy efficiency. To address this, I explored available options. I proposed a practical system for fractional GPU allocation within an HTCondor setup, enabling multiple small jobs to share GPU resources effectively (A.11, [146, 147]). Bare-metal benchmarks highlighted the performance, flexibility of the Multi-Process Service (MPS), and the security of Multi Instance GPU (MIG) configurations. With the fractional-GPU system, the cost per compute was improved by 3.1×, while energy consumption was lowered by 75%. Further benchmarks utilizing the TOpAS cluster and our local batch system supported the results. The system I developed has been actively deployed in production since mid-2024, with no significant disruptions.

Finally, efficiency at a global campaign scale was examined using an official HEPBenchmark candidate simulation workload that represents production-style event generation. GPU resources achieved a $2.6 - 2.9\times$ throughput per cost at $1.8 - 2.3\times$ lower energy consumption than vectorized CPU results. The benchmark results not only quantified GPU performance gains but also contributed to the evolution of GPU HEP Benchmarks under the HEPiX Benchmarking Working Group [93].

A central finding of this thesis is the consistent advantage of GPU resources across all evaluated applications, from small-scale neural-network training tasks to large-scale production HEP simulations. In every case, GPUs provided superior speed, throughput, and energy efficiency compared to CPU-based setups, demonstrating both the generality and scalability of GPU acceleration in HEP. In addition to the direct requirement for GPU resources in large-scale Machine Learning (ML) efforts, such as the TauT studies, this demonstrates the relevance and need for GPU hardware in HEP.

Looking ahead, several natural steps result from this work:

1. **TauTransformer for Run 3 and HL-LHC data**: The increasing complexity and volume of events will benefit from high-efficiency τ_h classification. GPUs will enable the generation and preprocessing of larger simulated training datasets, allowing the model to fully exploit data scaling effects and achieve improved performance.

2. **GPU benchmarking and resource planning**: A comprehensive suite of GPU benchmarks will be essential to inform hardware procurement, workload scheduling, and long-term strategy in HEP.

3. **Energy-aware computing**: As HEP computing scales for HL-LHC, energy-efficient integration of GPU resources into scheduling and resource allocation will be critical to ensure sustainable High Throughput Compute (HTC).

In summary, this thesis demonstrates that GPUs not only enable HEP research but also fundamentally reshape the landscape of throughput and energy efficiency. GPUs extend the methodological options available and provide operational advantages across a diverse set of applications.

APPENDIX A

Appendix

A.1 Sample Selection

Monte Carlo simulations are used to train the TauTransformer (TauT) model and to validate its performance. The event samples correspond to the 2018 data-taking periods and cover a range of Standard Model (SM) processes relevant for τ_{h} identification.

The Shuffle and Merge (S&M) and testing datasets are built from the following simulated processes:

- **Drell-Yan and W+jets**:
  Z/γ^*+jets and W+jets events, simulated with MADGRAPH5_AMC@NLO [148] at leading order (LO), using MLM jet matching and merging [149].

- **Top quark processes**:
  $t\bar{t}$ and single top production, simulated with POWHEG at NLO precision [150–154].

- **QCD multijet events**:
  Generated with MADGRAPH5_AMC@NLO and PYTHIA [155].

- **Higgs boson processes**:
  $H \to \tau\tau$ events produced through gluon-gluon fusion, vector-boson fusion, associated ZH, WH, and double-Higgs ($HH \to bb\tau\tau$) production, simulated with POWHEG.

- **$\mathbf{Z' \to e^+e^-}$**:
  Simulated at LO with PYTHIA and TAUOLA [156], with $m(Z')$ between 1 and 4 TeV, used to provide clean electron labels at high transverse momentum.

- **Tau-gun samples**:
  Single τ leptons with uniform η and p_{T} distributions ($-2.5 < \eta < 2.5$, $15 < p_{\mathrm{T}} < 3000$ GeV), including the tau decay but no proton-proton interaction. These are used to extend phase-space coverage into otherwise sparse regions.

All events are interfaced with PYTHIA to model parton showering, fragmentation, and tau decays, using the CP5 tune for underlying event simulation [157]. Detector effects are simulated with GEANT4 [158], and events include pileup interactions matched to the observed data distribution.

Table A.1 details the involved processes, their sample selections, as well as the attribution of τ_h candidate output classes derived from the samples. Dedicated samples, such as τ-gun, are included to extend the coverage in otherwise sparse regions of the sampling space.

The output class selection of the τ_h candidates from the samples is:

- **e**(μ):matching to a prompt $e(\mu)$ or τ decaying into a $e(\mu)$ at generator level within a cone radius of $\Delta R = 0.2$. The associated generated lepton has to pass $p_\mathrm{T}^\mathrm{vis}(l) > 8\,\mathrm{GeV}$.
- τ_h:matching to a hadronically decaying prompt τ at generator level within a cone radius of $\Delta R = 0.2$. The associated generated lepton has to pass $p_\mathrm{T}^\mathrm{vis}(l) > 15\,\mathrm{GeV}$.
- **jet**: absence of an associated generator level lepton and matching to a generator level jet within a cone of radius $\Delta R = 0.4$. In addition, selected procedure jets with a reconstructed $p_\mathrm{T}(\mathrm{jet}) > 80\,\mathrm{GeV}$ are randomly rejected with a probability of $p = 1 - \exp(-0.05 \cdot (80 - p_\mathrm{T}))$, to balance contributions from low- and high-p_T ranges.

While the adversarial dataset is composed of similar samples, it also includes real data samples that are selected separately. Real data selection and simulation details are detailed in [13].

Table A.1: S&M dataset sample selection and primary class attribution. H_T refers to the scalar sum of the transverse momenta. τ-Gun refers to samples with a simulated tau lepton, but no proton-proton interaction.

Process type	Sample selection	Output classes
Drell-Yang	inclusive	e, μ, τ_h, jet
	jet-binned	e, μ, τ_h, jet
	H_T-binned	e, μ, τ_h, jet
QCD	p_T-binned	jet
	H_T-binned	jet
$t\bar{t}$	leptonic	e, μ, τ_h, jet
	semi-leptonic	e, μ, τ_h, jet
	fully-hadronic	e, μ, jet
W+jets	jet-binned	e, μ, τ_h, jet
	H_T-binned	e, μ, τ_h, jet
Higgs	ZH→ $\tau\tau$	τ_h
	$\mathrm{W}^\pm$H→ $\tau\tau$	τ_h
	H→ $\tau\tau$ (vector-boson fusion)	τ_h
	HH→ $bb\tau\tau$ (gluon-gluon fusion)	τ_h
$Z' \to e\bar{e}$	$m(Z') \in [1000, 4000]$ GeV	e
τ-Gun	$p_\mathrm{T}(\tau) \in [15, 3000]$ GeV	τ_h

A.2 Hadronic Tau Classification Input Features

The input features of DeepTau and TauT are similar, though the latter expands on the feature set of the former. Past that, I added and removed multiple features from the feature set used in [55]. Tables A.2 to A.5 outline the breadth of features for each input collection type and note which have been added or removed throughout the different studies.

Table A.2: Global (τ_h candidate) input features of the TauT.

Category	**Features**
Particle properties	`particle_type`[1], `tau_pt`, `tau_eta`, `tau_mass`, `tau_E_over_pt`, `tau_charge`
Isolation	`tau_chargedIsoPtSum`, `tau_chargedIsoPtSumdR03_over_dR05`, `tau_footprintCorrection`, `tau_neutralIsoPtSum`, `tau_neutralIsoPtSumWeight_over_neutralIsoPtSum`, `tau_neutralIsoPtSumWeightdR03_over_neutralIsoPtSum`, `tau_neutralIsoPtSumdR03_over_dR05`, `tau_photonPtSumOutsideSignalCone`, `tau_puCorrPtSum`
Secondary vertex	`tau_hasSecondaryVertex`[1], `tau_sv_minus_pv_x`[1,2], `tau_sv_minus_pv_y`[1,2], `tau_sv_minus_pv_z`[1,2], `tau_flightLength_x`, `tau_flightLength_y`, `tau_flightLength_z`, `tau_flightLength_sig`
Impact parameters	`tau_dxy_valid`, `tau_dxy`, `tau_dxy_sig`, `tau_ip3d_valid`[2], `tau_ip3d`, `tau_ip3d_sig`, `tau_dz_sig_valid`, `tau_dz`, `tau_dz_sig`
Misc.	`rho`, `tau_n_charged_prongs`, `tau_n_neutral_prongs`, `tau_pt_weighted_deta_strip`, `tau_pt_weighted_dphi_strip`, `tau_pt_weighted_dr_signal`, `tau_pt_weighted_dr_iso`, `tau_e_ratio_valid`[2], `tau_e_ratio`, `tau_gj_angle_diff_valid`, `tau_gj_angle_diff`, `tau_n_photons`, `tau_emFraction`, `tau_inside_ecal_crack`, `tau_leadingTrackNormChi2`, `tau_leadChargedCand_etaAtEcalEntrance_minus_tau_eta`

Table A.3: PF candidate input features of the TauT.

Category	**Features**
Position (η–ϕ)	`r`[1], `theta`[1]
Particle	`rel_pt`, `particle_type`[1], `charge`
Vertex & IP	`pvAssociationQuality`, `fromPV`, `vertex_dx`, `vertex_dy`, `vertex_dz`, `vertex_z_valid`[3], `vertex_dx_tauFL`, `vertex_dy_tauFL`, `vertex_dz_tauFL`, `dxy`, `dxy_sig`, `dxy_sig_valid`[3], `dz`, `dz_valid`[3], `dz_sig`, `dz_sig_valid`[3]
Tracker info	`lostInnerHits`, `nPixelHits`, `hasTrackDetails`, `nHits`[1], `nPixelLayers`[1], `nStripLayers`[1], `track_ndof`, `track_ndof_valid`, `chi2_ndof`
Calorimeter	`hcalFraction`, `rawCaloFraction`, `rawHcalFraction`[1]
Miscellaneous	`tauLeadChargedHadrCand`, `puppiWeight`

Table A.4: RECO muon input features of the TauT.

Category	**Features**
Position (η–ϕ)	`r`[1], `theta`[1]
Particle	`rel_pt`, `particle_type`[1]
Calorimeter	`segmentCompatibility`, `caloCompatibility`, `pfEcalEnergy_valid`[2], `rel_pfEcalEnergy`
Track & IP	`dxy`, `dxy_sig`, `normalizedChi2_valid`, `normalizedChi2`, `numberOfValidHits`
Muon chambers	`n_matches_DT_{1,2,3,4}`, `n_matches_CSC_{1,2,3,4}`, `n_matches_RPC_{1,2,3,4}`, `n_hits_DT_{1,2,3,4}`, `n_hits_CSC_{1,2,3,4}`, `n_hits_RPC_{1,2,3,4}`

[1] Added with [55].
[2] Removed in this thesis.
[3] Added in this thesis.

Table A.5: RECO electron input features of the TauT.

Category	**Features**
Position (η–ϕ)	`r`[1], `theta`[1]
Particle	`rel_pt`, `particle_type`[1]
MVA variables	`mvaInput_earlyBrem`, `mvaInput_lateBrem`, `mvaInput_sigmaEtaEta`, `mvaInput_hadEnergy`, `mvaInput_deltaEta`, `mva_valid`[3]
Shower shape	`sigmaEtaEta`[1], `sigmaIetaIeta`[1], `sigmaIphiIphi`[1], `sigmaIetaIphi`[1], `e1x5`[1], `e2x5Max`[1], `e5x5`[1], `r9`[1], `hcalDepth1OverEcal`[1], `hcalDepth2OverEcal`[1], `hcalDepth1OverEcalBc`[1], `hcalDepth2OverEcalBc`[1], `eLeft`[1], `eRight`[1], `eBottom`[1], `eTop`[1], `full5x5_sigmaEtaEta`[1], `full5x5_sigmaIetaIeta`[1], `full5x5_sigmaIphiIphi`[1], `full5x5_sigmaIetaIphi`[1], `full5x5_e1x5`[1], `full5x5_e2x5Max`[1], `full5x5_e5x5`[1], `full5x5_r9`[1], `full5x5_hcalDepth1OverEcal`[1], `full5x5_hcalDepth2OverEcal`[1], `full5x5_hcalDepth1OverEcalBc`[1], `full5x5_hcalDepth2OverEcalBc`[1], `full5x5_eLeft`[1], `full5x5_eRight`[1], `full5x5_eBottom`[1], `full5x5_eTop`[1], `full5x5_e2x5Left`[1], `full5x5_e2x5Right`[1], `full5x5_e2x5Bottom`[1], `full5x5_e2x5Top`[1]

A.3 TauTransformer Hyperparameters

Table A.6: Summary of key hyperparameters for the TauT architecture. Here, d denotes the dimensionality of feature vectors or hidden layers, N denotes the number of layers, heads, or categories, and p denotes dropout probabilities. The encoder dimensions are given as input and output widths of the feed-forward sublayers. Parameters adjusted from [55] and self-attention layers according to [63].

Component	**TauTransformer (TaT)**
Input collections	Particle Flow, RECO e/μ, global variables
Token type embedding	$N_{\text{in}} = 10 \rightarrow d_{\text{cat}} = 2$
Feature embedding	$d_{\text{ff}} = 256 \rightarrow d_{\text{model}} = 64$
Dropout (all)	$p = 0.1$
Encoder layers	$N_{\ell} = 6$ self-attention
Positional encoding	None (η and ϕ part of features)
Normalization layers	Yes
Residual connections	Yes
Attention heads	$N_h = 8$, $d_{\text{head}} = 8$
Encoder channels	$d_{\text{ff},1} = 256$, $d_{\text{ff},2} = 64$
Decoder layers	$N_{\text{ff}} = 3$: $256 \rightarrow 128 \rightarrow 4$
Global pooling	Sum over tokens
Activation function	GeLu [159]
Parameters	$466k$ (416k encoder + 50k decoder)
Training data	S&M 10% – 90% train, 10% valid
Batch size	280 to 350, depending on hardware
Batching scheme	Dynamic token binned
Learning rate	$\alpha = 1 \times 10^{-4}$
Optimizer	Adam [74]: $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-7}$
Weight initialization	Glorot [160]
Loss	Categorical cross-entropy [61]
Gradient clipping	No
Early stopping	After 10 epochs

A.4 DeepTau V2.5 Loss and Gradient Computation

All information in this section is based exclusively on the configurations defined in the TauMLTools repository [59]. The DeepTau Gradient computation involves four parts, split into two losses:

1. a Categorical Cross-Entropy (CCE) loss [61] that discriminates between the signal and the combined three background classes
2. a focal loss [62] that separates τ_h from all other classes by penalizing high confidence misidentification with an $F(1-y_\tau^{label}, 1-y_\tau^{pred})$-approach
3. a focal loss that rewards non-tau categorical high confidence discrimination under the condition that the prediction score for the τ_h class is at least 0.1
4. a binary cross-entropy loss that discriminates between simulated and experimental data, independent of the signal and background classes

The losses are given by:

$$
\begin{aligned}
L_{cat}\left(\mathbf{y}^{label}, \mathbf{y}^{pred}\right) = {} & \underbrace{\kappa_\tau H_\tau(y_\tau^{label}, y_\tau^{pred})}_{\text{Separation of } e,\mu,\tau,\text{jet}} && \text{(A.1)}\\
& + \underbrace{\left(\kappa_e + \kappa_\mu + \kappa_\text{jet}\right) N_\tau F\left(1-y_\tau^{label}, 1-y_\tau^{pred}; \gamma_{cmb}\right)}_{\text{Focused separation of } e,\mu,\text{jet from } \tau} && \text{(A.2)}\\
& + \underbrace{\kappa_F \sum_{i\in\{e,\mu,\text{jet}\}} \kappa_i \hat{\Theta}\left(y_\tau^{pred} - 0.1\right) N_i F\left(y_i^{label}, y_i^{pred}; \gamma_i\right)}_{\text{Focused separation of } \tau \text{ from } e,\mu,\text{jet for } y_\tau^{pred} > 0.1} && \text{(A.3)}\\
L_{adv}\left(y_{adv}^{label}, y_{adv}^{pred}\right) = {} & H_{bin}\left(y_{adv}^{label}, y_{adv}^{pred}\right), && \text{(A.4)}
\end{aligned}
$$

with the constants listed in Table A.7.

In this, $\mathbf{y} = (y_e, y_\mu, y_\tau, y_\text{jet})$ are the predictions and generator-level truth. H_τ, H_{bin}, F, and $\hat{\Theta}$ are the CCE concerning only the τ output class, the binary cross-entropy, the focal loss, and a smoothed Heaviside function, respectively:

$$
\begin{aligned}
H_\tau(y^{label}, y^{pred}) &= -y^{label}\log y^{pred} && \text{(A.5)}\\
H_{bin}(y^{label}, y^{pred}) &= -y^{label}\log y^{pred} - \left(1-y^{label}\right)\log\left(1-y^{pred}\right) && \text{(A.6)}\\
F(y^{label}, y^{pred}; \gamma) &= -y^{label}\left(1-y^{pred}\right)^\gamma \log y^{pred} && \text{(A.7)}\\
\hat{\Theta}(x) &= \left(\tanh\left(70\cdot x\right)+1\right)/2 && \text{(A.8)}
\end{aligned}
$$

The two losses are used separately for the non-shared parts of the gradient computation. For the shared part of the architecture, they combine as shown in Figure 2.5:

$$
G_i = k_1\frac{\partial L_{cat}}{\partial w_i} - k_2\frac{\partial L_{adv}}{\partial w_i}, \tag{A.9}
$$

Table A.7: Summary of all constants involved in the DeepTau V2.5 loss for the initial training and the later adversarial fine-tuning.

Variable	Initial Value	Adv. Value
κ_e	0.4	0.4
κ_μ	1.0	0.75
κ_τ	2.0	2.0
κ_{jet}	0.6	0.85
κ_F	5.0	5.0
γ_{cmb}	0.5	0.5
γ_e, γ_μ, γ_{jet}	2.0	2.0
N_τ	1.17153	1.17153
N_e, N_μ, N_{jet}	1.63636	1.63636
k_1	—	1
k_2	—	10

with w_i denoting the weights of the feature-extraction layers. Two optimizers are applied separately: one for the shared and categorical parts of the architecture and one for the adversarial part exclusively.

A.5 TauTransformer Working Point Efficiencies

Tables A.8 to A.22 list the False Positive Rate (FPR) of the DeepTau Working Points (WPs) derived in Chapter 3. The values listed here are 12 of the 20 DeepTau originated WPs, which are commonly used in Compact Muon Solenoid (CMS) analyses [69]. All results are based on the TauT architecture, except for the DeepTau V2.1 and DeepTau V2.5 results presented in Tables A.11 to A.13. The tables are listed in the same order as the studies in Chapter 3: stability study, comparison with DeepTau, data volume scaling, number of epoch scaling, particle constituent scaling.

Table A.8: WP FPR against electron stability.

Mis-ID	60% eff.	80% eff.	95% eff.	99% eff.
Mean	5.55×10^{-5}	3.90×10^{-4}	3.21×10^{-3}	1.31×10^{-2}
Abs. $\pm\sigma$	1.02×10^{-5}	2.79×10^{-5}	1.33×10^{-4}	6.65×10^{-4}
Rel. $\pm\sigma$	18.46%	7.15%	4.16%	5.07%

Table A.9: WP FPR against muon stability.

Mis-ID	99.5% eff.	99.8% eff.	99.9% eff.	99.95% eff.
Mean	1.34×10^{-4}	1.93×10^{-4}	3.41×10^{-4}	1.01×10^{-3}
Abs. $\pm\sigma$	5.66×10^{-6}	4.76×10^{-6}	1.54×10^{-5}	9.73×10^{-5}
Rel. $\pm\sigma$	4.21%	2.47%	4.53%	9.61%

Table A.10: WP FPR against jet stability.

Mis-ID	50% eff.	60% eff.	70% eff.	80% eff.
Mean	1.01×10^{-3}	1.75×10^{-3}	3.01×10^{-3}	5.55×10^{-3}
Abs. $\pm\sigma$	1.11×10^{-4}	1.56×10^{-4}	2.17×10^{-4}	3.42×10^{-4}
Rel. $\pm\sigma$	11.01%	8.92%	7.21%	6.16%

Table A.11: WP FPR against electron TauT and DeepTau comparison.

Network	60% eff.	80% eff.	95% eff.	99% eff.
TauT full	1.67×10^{-5}	1.13×10^{-4}	1.30×10^{-3}	6.65×10^{-3}
TauT partial	5.20×10^{-5}	3.96×10^{-4}	3.33×10^{-3}	1.37×10^{-2}
DeepTau 2.1	1.58×10^{-4}	1.06×10^{-3}	8.44×10^{-3}	4.45×10^{-2}
DeepTau 2.5	1.04×10^{-4}	8.07×10^{-4}	6.99×10^{-3}	3.11×10^{-2}

Table A.12: WP FPR against muon TauT and DeepTau comparison.

Network	99.5% eff.	99.8% eff.	99.9% eff.	99.95% eff.
TauT full	1.03×10^{-4}	1.44×10^{-4}	2.15×10^{-4}	5.80×10^{-4}
TauT partial	1.28×10^{-4}	1.90×10^{-4}	3.53×10^{-4}	1.05×10^{-3}
DeepTau 2.1	3.74×10^{-4}	4.70×10^{-4}	7.40×10^{-4}	2.55×10^{-3}
DeepTau 2.5	3.77×10^{-4}	5.01×10^{-4}	6.24×10^{-4}	8.60×10^{-4}

Table A.13: WP FPR against jet TauT and DeepTau comparison.

Network	50% eff.	60% eff.	70% eff.	80% eff.
TauT full	6.72×10^{-4}	1.11×10^{-3}	1.95×10^{-3}	3.76×10^{-3}
TauT partial	1.00×10^{-3}	1.71×10^{-3}	2.94×10^{-3}	5.54×10^{-3}
DeepTau 2.1	8.51×10^{-4}	1.44×10^{-3}	2.48×10^{-3}	4.60×10^{-3}
DeepTau 2.5	5.13×10^{-4}	8.45×10^{-4}	1.44×10^{-3}	2.68×10^{-3}

Table A.14: WP FPR against electron for varying training data volume.

τ_h candidates	60% eff.	80% eff.	95% eff.	99% eff.
2.00×10^5	6.48×10^{-4}	2.72×10^{-3}	1.26×10^{-2}	5.93×10^{-2}
4.00×10^5	5.00×10^{-4}	2.25×10^{-3}	1.08×10^{-2}	4.83×10^{-2}
8.00×10^5	5.26×10^{-4}	2.35×10^{-3}	1.08×10^{-2}	4.93×10^{-2}
1.60×10^6	2.11×10^{-4}	1.15×10^{-3}	6.35×10^{-3}	2.45×10^{-2}
3.20×10^6	1.28×10^{-4}	8.76×10^{-4}	5.63×10^{-3}	2.20×10^{-2}
6.40×10^6	6.94×10^{-5}	4.91×10^{-4}	3.68×10^{-3}	1.52×10^{-2}
1.28×10^7	5.06×10^{-5}	3.47×10^{-4}	2.89×10^{-3}	1.19×10^{-2}
2.56×10^7	2.55×10^{-5}	2.10×10^{-4}	2.23×10^{-3}	9.76×10^{-3}
5.12×10^7	1.58×10^{-5}	1.24×10^{-4}	1.50×10^{-3}	7.44×10^{-3}
9.00×10^7	1.69×10^{-5}	1.10×10^{-4}	1.31×10^{-3}	6.62×10^{-3}

Table A.15: WP FPR against muon for varying training data volume.

τ_h candidates	99.5% eff.	99.8% eff.	99.9% eff.	99.95% eff.
2.00×10^5	3.87×10^{-4}	6.39×10^{-4}	1.67×10^{-3}	7.73×10^{-3}
4.00×10^5	2.95×10^{-4}	4.92×10^{-4}	1.04×10^{-3}	3.95×10^{-3}
8.00×10^5	3.30×10^{-4}	5.48×10^{-4}	1.18×10^{-3}	4.18×10^{-3}
1.60×10^6	1.93×10^{-4}	2.99×10^{-4}	7.13×10^{-4}	2.39×10^{-3}
3.20×10^6	1.70×10^{-4}	2.37×10^{-4}	5.01×10^{-4}	1.71×10^{-3}
6.40×10^6	1.67×10^{-4}	2.18×10^{-4}	3.94×10^{-4}	1.17×10^{-3}
1.28×10^7	1.16×10^{-4}	1.88×10^{-4}	3.30×10^{-4}	9.41×10^{-4}
2.56×10^7	1.14×10^{-4}	1.57×10^{-4}	2.76×10^{-4}	7.83×10^{-4}
5.12×10^7	1.07×10^{-4}	1.47×10^{-4}	2.37×10^{-4}	6.63×10^{-4}
9.00×10^7	1.00×10^{-4}	1.40×10^{-4}	2.10×10^{-4}	5.39×10^{-4}

Table A.16: WP FPR against jet for varying training data volume.

τ_h candidates	50% eff.	60% eff.	70% eff.	80% eff.
2.00×10^5	2.94×10^{-3}	4.77×10^{-3}	7.90×10^{-3}	1.36×10^{-2}
4.00×10^5	2.26×10^{-3}	3.74×10^{-3}	6.21×10^{-3}	1.09×10^{-2}
8.00×10^5	1.47×10^{-3}	2.65×10^{-3}	4.74×10^{-3}	8.91×10^{-3}
1.60×10^6	1.46×10^{-3}	2.44×10^{-3}	4.10×10^{-3}	7.30×10^{-3}
3.20×10^6	1.18×10^{-3}	2.07×10^{-3}	3.52×10^{-3}	6.59×10^{-3}
6.40×10^6	1.13×10^{-3}	1.92×10^{-3}	3.23×10^{-3}	5.90×10^{-3}
1.28×10^7	8.98×10^{-4}	1.55×10^{-3}	2.70×10^{-3}	5.02×10^{-3}
2.56×10^7	8.69×10^{-4}	1.50×10^{-3}	2.53×10^{-3}	4.70×10^{-3}
5.12×10^7	6.41×10^{-4}	1.17×10^{-3}	2.11×10^{-3}	3.90×10^{-3}
9.00×10^7	6.64×10^{-4}	1.14×10^{-3}	1.97×10^{-3}	3.79×10^{-3}

Table A.17: WP FPR against electron for varying number of epochs.

# Epochs	60% eff.	80% eff.	95% eff.	99% eff.
9	3.27×10^{-5}	2.34×10^{-4}	2.38×10^{-3}	1.02×10^{-2}
25	2.25×10^{-5}	1.61×10^{-4}	1.77×10^{-3}	8.32×10^{-3}
54	1.58×10^{-5}	1.01×10^{-4}	1.37×10^{-3}	6.88×10^{-3}
70	1.69×10^{-5}	1.10×10^{-4}	1.31×10^{-3}	6.62×10^{-3}

Table A.18: WP FPR against muon for varying number of epochs.

# Epochs	99.5% eff.	99.8% eff.	99.9% eff.	99.95% eff.
9	1.19×10^{-4}	1.70×10^{-4}	2.69×10^{-4}	7.18×10^{-4}
25	1.16×10^{-4}	1.56×10^{-4}	2.47×10^{-4}	6.36×10^{-4}
54	9.86×10^{-5}	1.44×10^{-4}	2.17×10^{-4}	5.63×10^{-4}
70	1.00×10^{-4}	1.40×10^{-4}	2.10×10^{-4}	5.39×10^{-4}

Table A.19: WP FPR against jet for varying number of epochs.

# Epochs	50% eff.	60% eff.	70% eff.	80% eff.
9	8.11×10^{-4}	1.45×10^{-3}	2.62×10^{-3}	4.94×10^{-3}
25	7.04×10^{-4}	1.25×10^{-3}	2.26×10^{-3}	4.32×10^{-3}
54	6.61×10^{-4}	1.23×10^{-3}	2.17×10^{-3}	3.99×10^{-3}
70	6.64×10^{-4}	1.14×10^{-3}	1.97×10^{-3}	3.79×10^{-3}

Table A.20: WP FPR against electron for varying number of maximum particle constituents (tokens) per category. Additionally, comparison to a $\Delta R < 0.5$ cut on the constituent tokens.

# tokens	60% eff.	80% eff.	95% eff.	99% eff.
16	7.51×10^{-5}	4.02×10^{-4}	3.45×10^{-3}	1.44×10^{-2}
32	3.37×10^{-5}	3.11×10^{-4}	2.93×10^{-3}	1.21×10^{-2}
64	2.04×10^{-5}	1.96×10^{-4}	2.42×10^{-3}	5.02×10^{-2}
128	4.60×10^{-5}	1.46×10^{-4}	1.30×10^{-3}	6.91×10^{-3}
256	1.99×10^{-5}	1.56×10^{-4}	1.52×10^{-3}	7.51×10^{-3}
All	1.69×10^{-5}	1.10×10^{-4}	1.31×10^{-3}	6.62×10^{-3}
$\Delta R < 0.5$	1.79×10^{-5}	1.03×10^{-4}	1.22×10^{-3}	6.26×10^{-3}

Table A.21: WP FPR against muon for varying number of maximum particle constituents (tokens) per category. Additionally, comparison to a $\Delta R < 0.5$ cut on the constituent tokens.

# tokens	99.5% eff.	99.8% eff.	99.9% eff.	99.95% eff.
16	1.03×10^{-4}	1.65×10^{-4}	2.71×10^{-4}	9.09×10^{-4}
32	1.14×10^{-4}	1.63×10^{-4}	2.73×10^{-4}	7.81×10^{-4}
64	1.00×10^{0}	1.00×10^{0}	1.00×10^{0}	1.00×10^{0}
128	9.73×10^{-5}	1.40×10^{-4}	2.20×10^{-4}	5.91×10^{-4}
256	1.13×10^{-4}	1.45×10^{-4}	2.26×10^{-4}	7.13×10^{-4}
All	1.00×10^{-4}	1.40×10^{-4}	2.10×10^{-4}	5.39×10^{-4}
$\Delta R < 0.5$	1.07×10^{-4}	1.50×10^{-4}	2.23×10^{-4}	5.49×10^{-4}

Table A.22: WP FPR against jet for varying number of maximum particle constituents (tokens) per category. Additionally, comparison to a $\Delta R < 0.5$ cut on the constituent tokens.

# tokens	50% eff.	60% eff.	70% eff.	80% eff.
16	1.26×10^{-2}	1.43×10^{-2}	1.61×10^{-2}	1.86×10^{-2}
32	6.77×10^{-3}	8.13×10^{-3}	9.88×10^{-3}	1.26×10^{-2}
64	1.49×10^{-1}	1.52×10^{-1}	1.55×10^{-1}	1.60×10^{-1}
128	6.33×10^{-4}	1.17×10^{-3}	2.05×10^{-3}	3.81×10^{-3}
256	8.54×10^{-4}	1.48×10^{-3}	2.53×10^{-3}	4.70×10^{-3}
All	6.64×10^{-4}	1.14×10^{-3}	1.97×10^{-3}	3.79×10^{-3}
$\Delta R < 0.5$	1.83×10^{-3}	2.48×10^{-3}	3.67×10^{-3}	5.99×10^{-3}

A.6 TauTransformer Discriminant Score Distribution

The distribution of the signal and background scores determines the Receiver Operating Characteristic (ROC) shape. This section provides an outline of how specific features of the distribution translate into ROC behavior. Figures A.3 and A.2 illustrate the two distributions of the worst- and best-performing classifiers of my studies: the ones with minimum and maximum training data available to them. Figures A.1 shows the ROC curve of the classifiers from Section 3.5, including the two classifiers for which the distributions are shown.

The four-segmented shape of the ROC curve introduced in Section 3.3.2 is directly caused by the features of the score distribution. Traversing from high score to low score, or left to right on the ROC curve:

1. The initial steep ascent relates to the U-shaped contamination on the opposite side of the high-confidence correctly identified examples. In the shown double-logarithmic scaling, this population is magnified despite a signal-to-background ratio exceeding three orders of magnitude.

2. The following segment describes the distribution up to the knee point, where the background becomes dominant over the signal. The slope in this region indicates the population of medium-confidence background examples relative to the high-confidence misidentified background. Although the absolute changes in population for FPR and TPR are small in this region, it is also where the majority of WPs reside.

3. Similarly, the slope of the segment after the knee point describes the population of medium-confidence signal examples relative to the population of high-confidence misidentified signal examples. While the slope is mirrored along the (1-TPR)/FPR axis, the length of the second and this third segment is determined by the location of the score at which the dominant population flips from signal to background, as seen in the *vs.-jet* case.

4. The final segment flattens out in the same way as the first segment showed a near vertical slope. The double-log scaling magnifies the dwindling number of remaining signal examples relative to the total number of signal examples.

In a direct comparison between high- and low-performant classifiers, the entire ROC curve shifts diagonally towards higher TPR values and lower FPR values. In the context of the score distribution, this can be seen as a uniform shift of examples from the misidentified side to the correctly identified side. While the relative number of examples between medium- and high-confidence misidentification remains unchanged, the absolute number decreases. Therefore, the slope of the two center segments changes little, while the overall position of the curve shifts significantly.

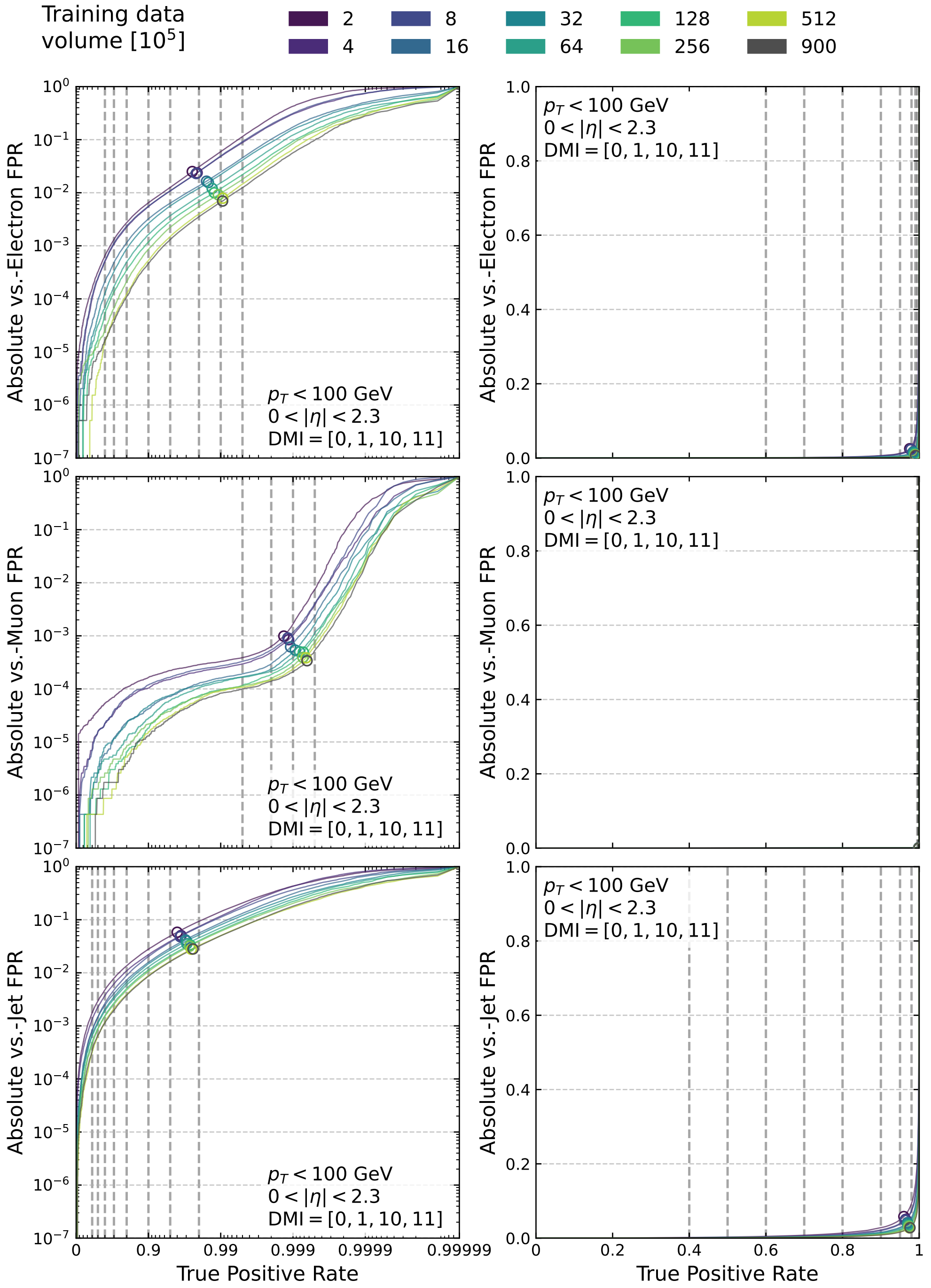


Figure A.1: Comparison of absolute classifier ROC performance against electron, muon, and jet backgrounds with logarithmic and linear scaling. The runs are shown in a gradient from green to blue as more training data becomes available. Full-scale TauT training with all available training data is shown in grey. Dashed vertical lines mark the target True Positive Rate (TPR) of the respective DeepTau WPs. Circles on the ROC curve indicate the curve's knee point. Decay Mode Integer (DMI) describes the generator-level truth decay mode, as defined in Section 2.1.3.

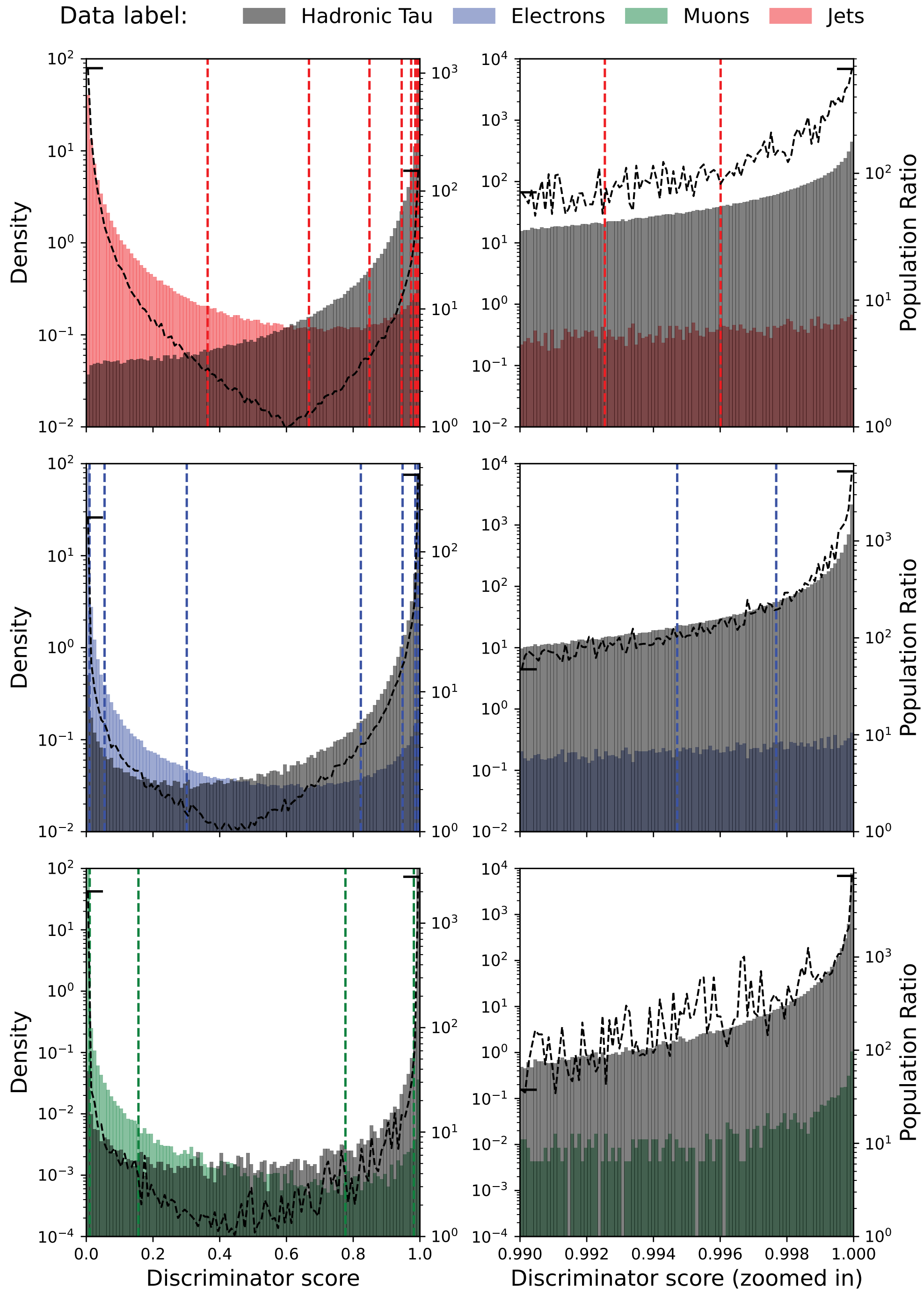


Figure A.2: Discriminant score distribution derived from the TauT, trained on 0.5% of the maximum training data, as described in Section 3.5. Each row shows the discrimination against a specific background (electron, muon, jet), with DeepTau WPs described in Section 2.4.4, ordered loosest to tightest from left to right (dashed lines). The right column presents a zoomed-in view of the highest bin from the left column. The dashed black line indicates the ratio between the major and minor population of the respective bin.

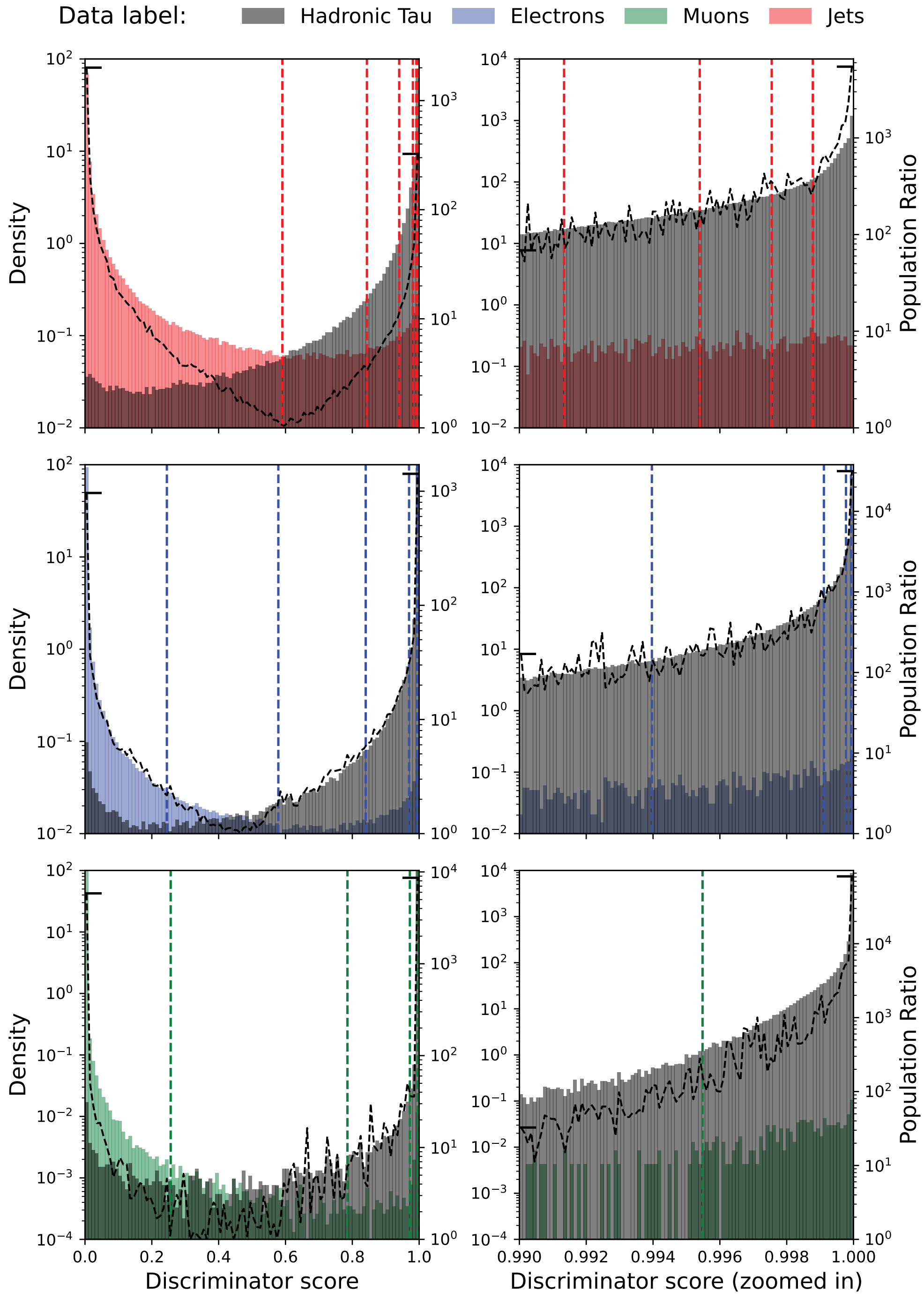


Figure A.3: Discriminant score distribution derived from the fully trained TauT described in Section 3.5. Each row shows the discrimination against a specific background (electron, muon, jet), with DeepTau WPs described in Section 2.4.4, ordered loosest to tightest from left to right (dashed lines). The right column presents a zoomed-in view of the highest bin from the left column. The dashed black line indicates the ratio between the major and minor population of the respective bin.

A.7 TauTransformer Data Scaling Power Law Fit Results

Table A.23: Fit results for data power-law scaling of the discriminant WPs against electron, muon, and jet FPR using a $A \cdot N^{-\alpha} + \mathrm{FPR}_\infty$ model.

WP	TPR	$A \pm \sigma_A$	$\alpha \pm \sigma_\alpha$	$\mathrm{FPR}_\infty \pm \sigma_{\mathrm{FPR}_\infty}$
		Electrons		
VVTight	0.6	$(3.50 \pm 7.45) \times 10^{-1}$	0.509 ± 0.177	$(0.00 \pm 7.69) \times 10^{-5}$
VTight	0.7	$(4.63 \pm 9.80) \times 10^{-1}$	0.473 ± 0.178	$(0.00 \pm 1.80) \times 10^{-4}$
Tight	0.8	$(5.71 \pm 10.70) \times 10^{-1}$	0.430 ± 0.159	$(0.00 \pm 3.92) \times 10^{-4}$
Medium	0.9	$(6.68 \pm 10.20) \times 10^{-1}$	0.373 ± 0.132	$(0.00 \pm 9.82) \times 10^{-4}$
Loose	0.95	$(8.78 \pm 11.50) \times 10^{-1}$	0.341 ± 0.115	$(0.00 \pm 1.91) \times 10^{-3}$
VLoose	0.98	2.47 ± 2.95	0.355 ± 0.104	$(0.00 \pm 3.85) \times 10^{-3}$
VVLoose	0.99	5.63 ± 7.82	0.369 ± 0.120	$(0.00 \pm 8.08) \times 10^{-3}$
VVVLoose	0.995	$(1.18 \pm 1.55) \times 10^{1}$	0.377 ± 0.113	$(0.00 \pm 1.39) \times 10^{-2}$
		Muons		
Tight	0.995	$(5.31 \pm 9.12) \times 10^{-2}$	0.418 ± 0.146	$(7.64 \pm 4.09) \times 10^{-5}$
Medium	0.998	$(5.41 \pm 8.72) \times 10^{-2}$	0.370 ± 0.139	$(6.44 \pm 8.81) \times 10^{-5}$
Loose	0.999	$(2.59 \pm 3.56) \times 10^{-1}$	0.419 ± 0.117	$(0.69 \pm 1.57) \times 10^{-4}$
VLoose	0.9995	$(1.43 \pm 2.15) \times 10^{1}$	0.625 ± 0.124	$(4.49 \pm 3.76) \times 10^{-4}$
		Jets		
VVTight	0.4	1.00 ± 1.30	0.537 ± 0.108	$(3.23 \pm 0.86) \times 10^{-4}$
VTight	0.5	1.60 ± 1.70	0.537 ± 0.088	$(6.35 \pm 1.13) \times 10^{-4}$
Tight	0.6	1.26 ± 0.94	0.478 ± 0.063	$(1.03 \pm 0.16) \times 10^{-3}$
Medium	0.7	1.50 ± 0.70	0.451 ± 0.039	$(1.72 \pm 0.18) \times 10^{-3}$
Loose	0.8	1.43 ± 0.49	0.402 ± 0.030	$(3.01 \pm 0.29) \times 10^{-3}$
VLoose	0.9	1.33 ± 0.69	0.338 ± 0.046	$(5.98 \pm 1.22) \times 10^{-3}$
VVLoose	0.95	1.62 ± 0.99	0.307 ± 0.055	$(1.03 \pm 0.31) \times 10^{-2}$
VVVLoose	0.98	2.91 ± 1.89	0.303 ± 0.058	$(2.07 \pm 0.62) \times 10^{-2}$

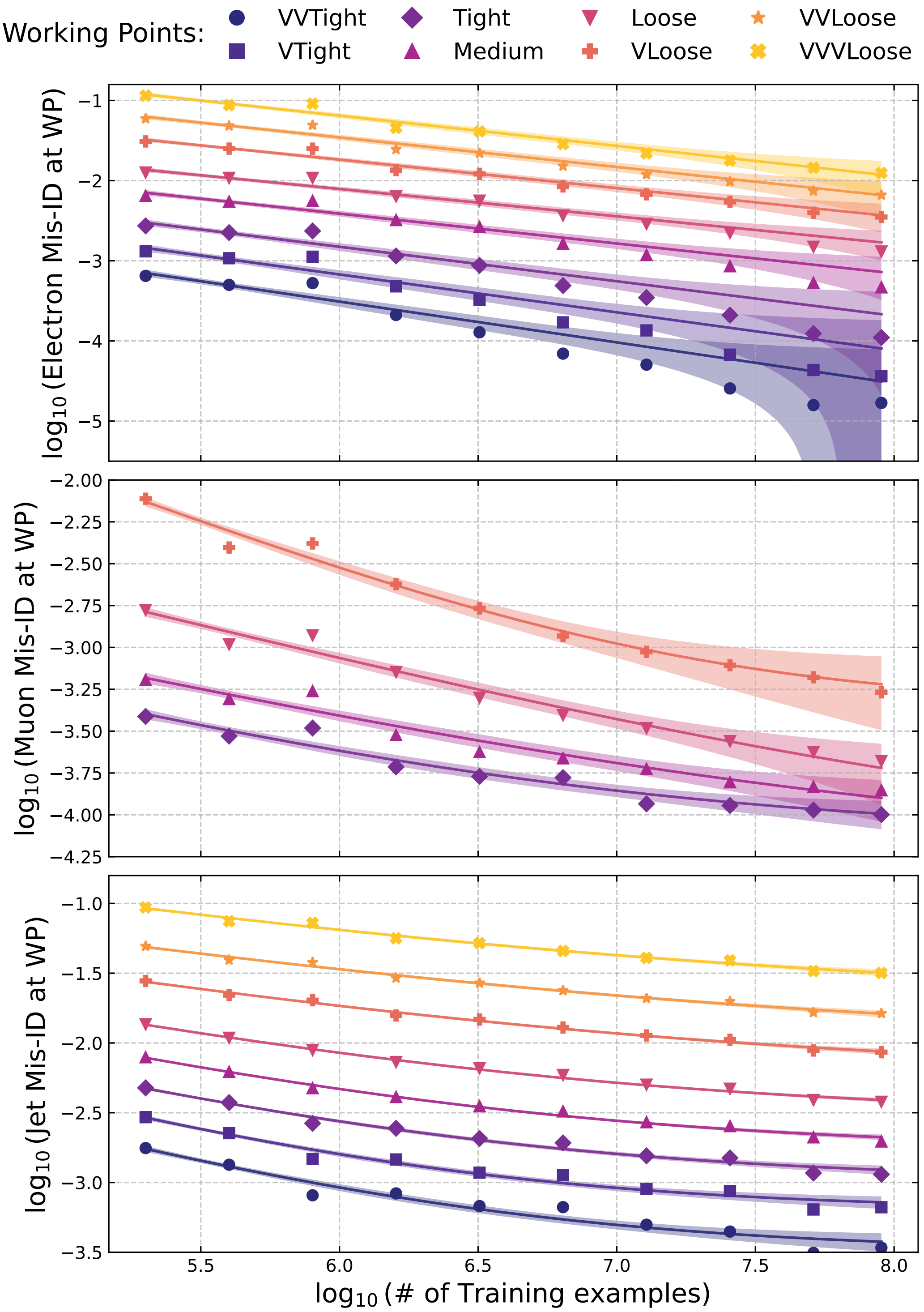


Figure A.4: Data volume power-law fit of the ROC kneepoint TPR and FPR for each background category. Each marker indicates the FPR (TPR) of the respective kneepoint derived from the Neural Network (NN) model that was trained on the indicated volume of data.

Table A.24: Fit results for data power-law scaling of the ROC kneepoint against electron, muon, and jet backgrounds, along the TPR and FPR axes using a $A \cdot N^{-\alpha} + \mathrm{R}_\infty$ model.

Fit Target	$A \pm \sigma_A$	$\alpha \pm \sigma_\alpha$	$R_\infty \pm \sigma_{R_\infty}$
		Electrons	
1-TPR	0.310 ± 0.275	0.218 ± 0.088	$(3.214 \pm 4.829) \times 10^{-3}$
FPR	0.380 ± 0.350	0.217 ± 0.091	$(0.000 \pm 6.242) \times 10^{-3}$
		Muons	
1-TPR	0.889 ± 2.616	0.582 ± 0.243	$(7.090 \pm 0.867) \times 10^{-4}$
FPR	0.014 ± 0.016	0.222 ± 0.119	$(1.033 \pm 2.620) \times 10^{-4}$
		Jets	
1-TPR	0.352 ± 0.363	0.236 ± 0.099	$(1.988 \pm 0.440) \times 10^{-2}$
FPR	0.627 ± 0.262	0.225 ± 0.041	$(1.751 \pm 0.392) \times 10^{-2}$

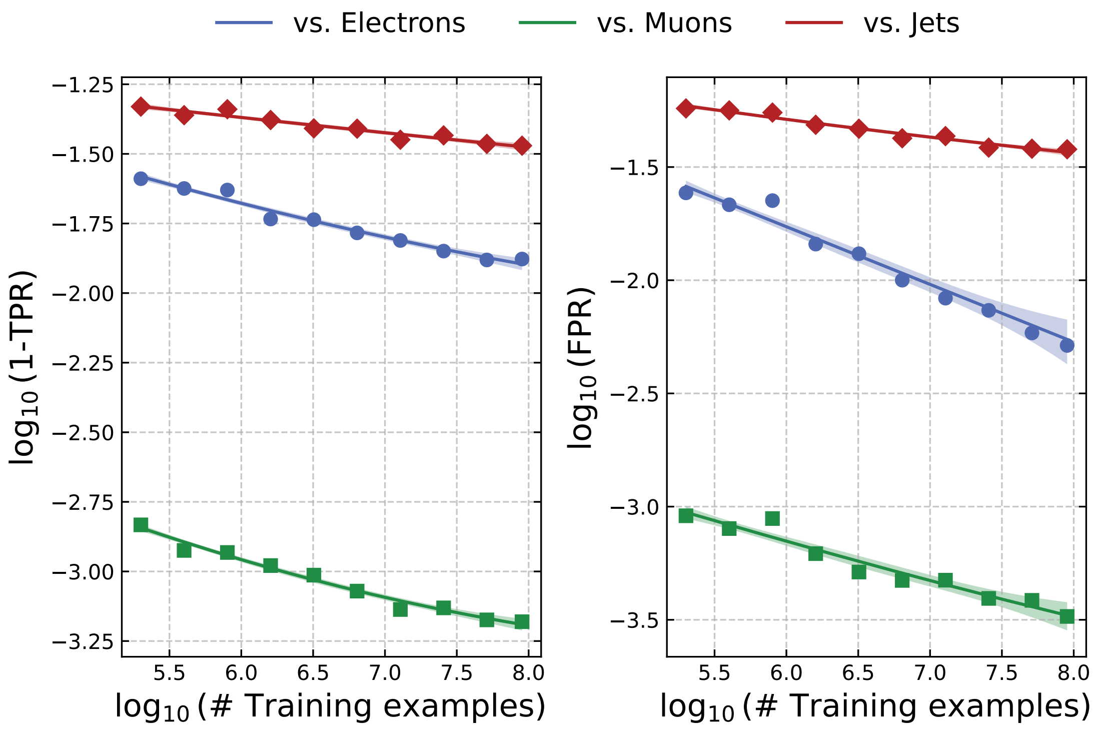


Figure A.5: Data volume power-law fit of each WP for each background category. Each marker indicates the FPR of the respective WP derived from the NN model that was trained on the indicated volume of data. The TPR values of WPs across the three background types differ. Table 2.2 details the respective mapping.

A.8 GPU utilization metrics

This appendix provides a brief comparison of the three GPU utilization metrics available through NVIDIA Data Center GPU Manager (DCGM) and describes why I chose Streaming Multiprocessor Activity (SMACT) for my hardware benchmarks. DCGM provides a wide range of metrics that describe individual components of its hardware [161]. While these provide a breadth of information on GPU stress during a benchmark workload, they are also difficult to interpret correctly. Three of the metrics lend themselves as general GPU utilization metrics and are described by [161] as follows:

- Graphic Activity (GRACT): the ratio of time the graphics engine is active. The graphics engine is active if a graphics/compute context is bound and the graphics pipe or compute pipe is busy.
- SMACT: the ratio of cycles an SM has at least one warp assigned (computed from the number of cycles and elapsed cycles).
- Streaming Multiprocessor Occupancy (SMOCC): the ratio of the number of warps resident on an SM.

Notably, the default GPU utilization reported by System Management Interface (SMI) is based on GRACT.

Each of the three metrics represents utilization in some way, but not all of them are equally representative of the amount of work performed by a GPU. Figures A.6 to A.8 showcase the behavior of the three utilization metrics at the hands of the preliminary Time-Slicing and Multi-Process Service (MPS) runs of the *MSSM Training* workload from Section 6.6.3.

Based on the runtime progression, I expected a mostly flat utilization curve in the non-MPS scenario and a linear increase in utilization in the MPS scenario. For both SMACT and SMOCC metrics, this held, although the absolute values of the metrics differed by roughly a factor of 2.5 throughout all runs. This behavior is expected, as activity does not imply 100% occupation.

In the case of the GRACT metric, results looked much different. The metric quickly rose in the scenarios without MPS, reaching nearly 100% with four concurrent instances of the workload. At the same time, the runtime increased close to linearly. I take from this that, in the usage scenario with multiple processes assigned to a single GPU, GRACT does not accurately reflect GPU utilization. Instead, it only shows accurate utilization for a single concurrent process, then transitions to 100% utilization for multiple simultaneous processes. The scenario with MPS activated shows results more similar to the other utilization metrics. While the progression is less smooth than with SMACT or SMOCC, the metric still increases close to linearly.

The conclusion I drew from these results is that both SMACT and SMOCC show how much of a GPU's compute capacity is utilized by the respective workload. On the other hand, GRACT is more in line with a description of how busy the GPU is, regardless of the amount of actual work performed. While the two interpretations of Utilization

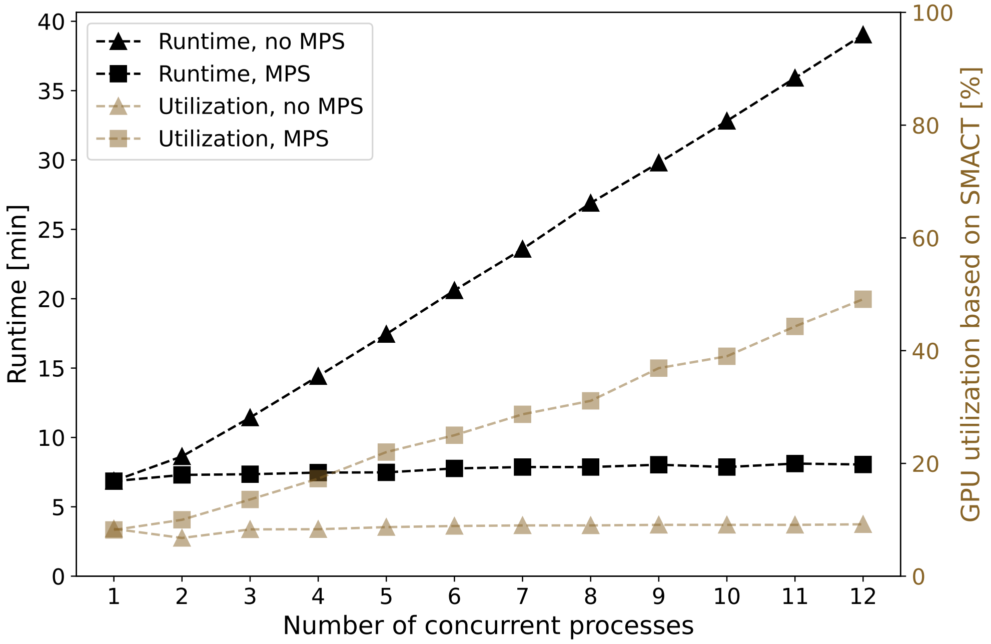


Figure A.6: Streaming Multiprocessor Activity during the preliminary scan of the Time-Slicing and MPS approaches from Section 6.6.3. All processes were simultaneously placed on the same GPU.

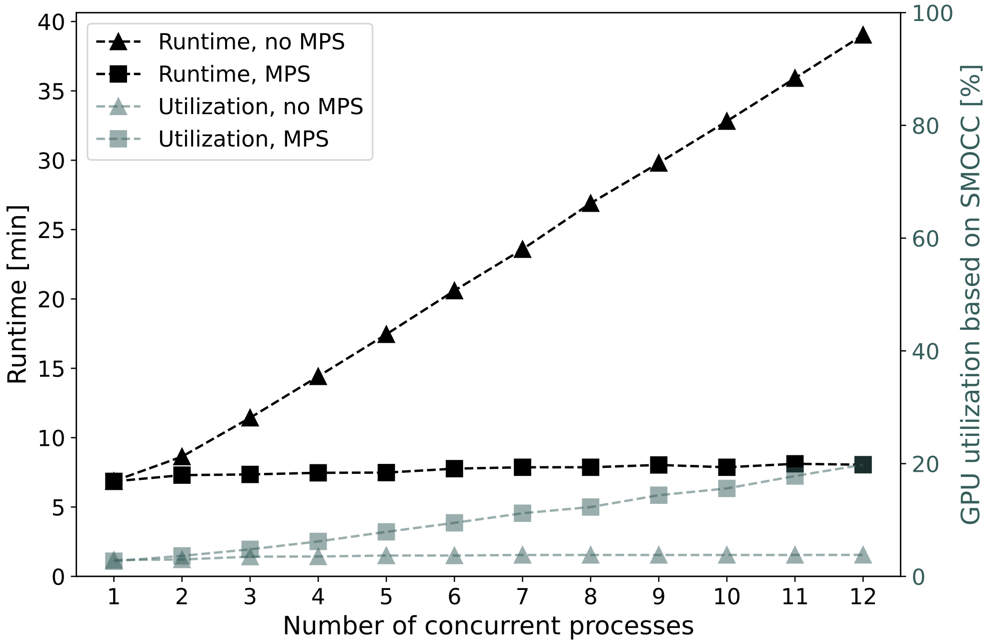


Figure A.7: Streaming Multiprocessor Occupancy during the preliminary scan of the Time-Slicing and MPS approaches from Section 6.6.3. All processes were simultaneously placed on the same GPU.

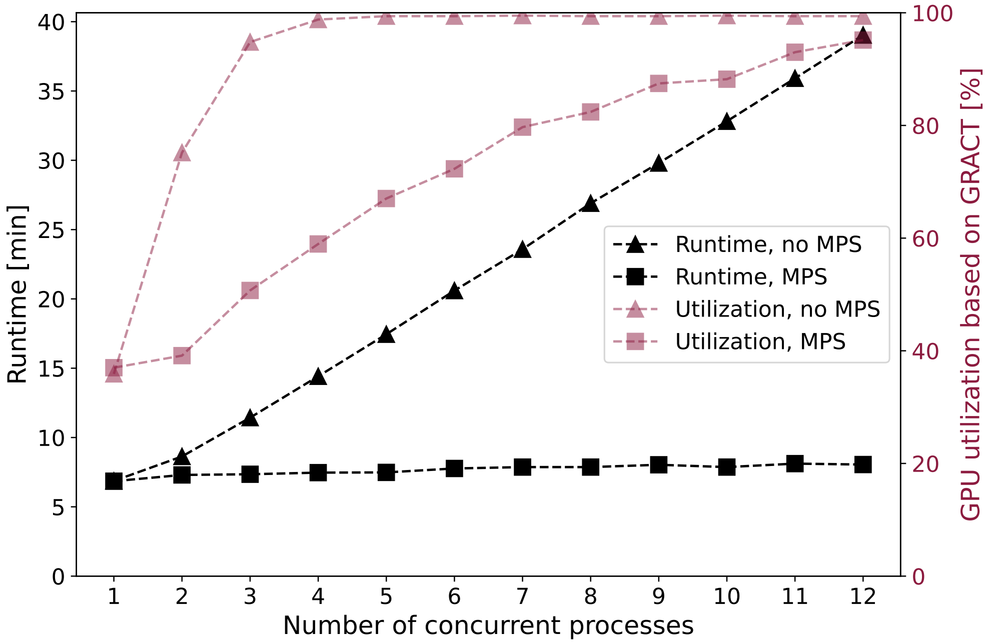


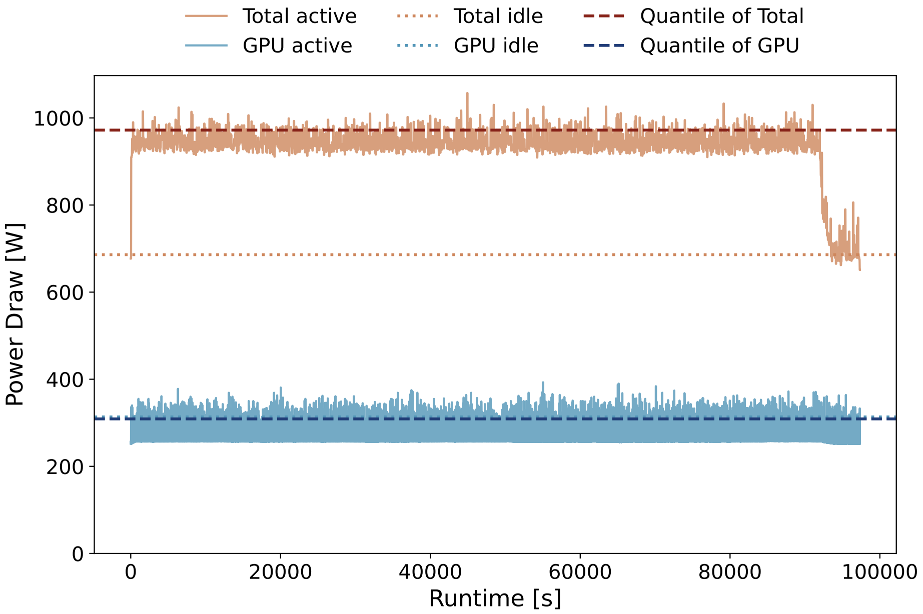
Figure A.8: Graphics Engine Activity during the preliminary scan of the Time-Slicing and MPS approaches from Section 6.6.3. All processes were simultaneously placed on the same GPU.

align when running a single process, they diverge quickly as more concurrent processes get added.

For studies with a fixed workload and a variable number of processes, GRACT is therefore unsuited. I consider the choice between SMACT and SMOCC to be largely equal and chose SMACT, as the larger absolute values are easier to compare.

Additionally, in scenarios with many concurrent processes accessing the same GPU and multiple GPUs on the same machine, all utilization metrics are questionable. A prime example of this behavior is found in Section 6.8, where the Multi Instance GPU (MIG) variation of the benchmark achieved a higher SMACT than the MPS scenario, despite lower total throughput. For this reason, GPU utilization is treated as secondary to runtime and throughput in this thesis.

A.9 Benchmark Power Draw Recordings



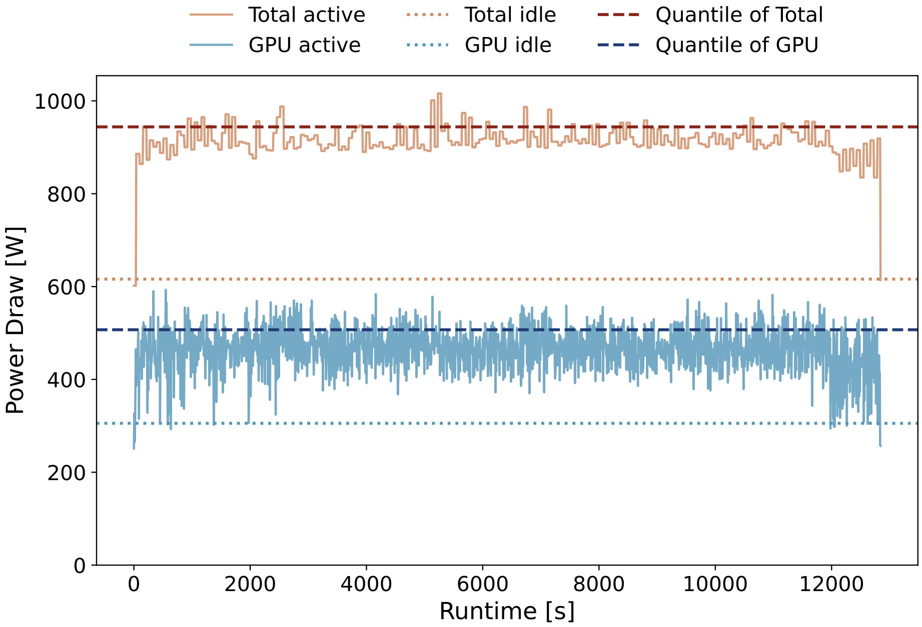
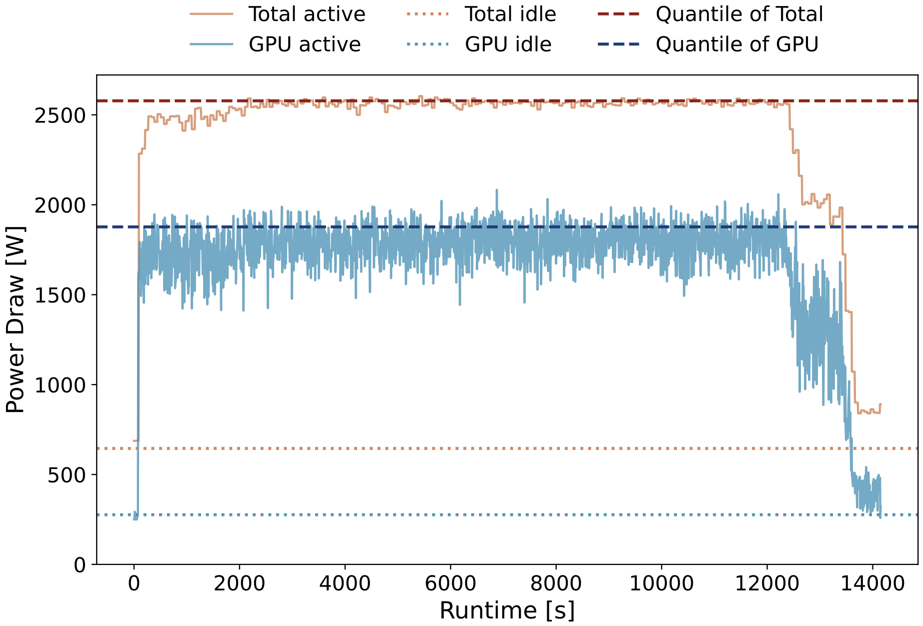


Figure A.9: Recorded power draw data of the *TauTransformer Training* workload in the CPU-only (upper) and baseline single GPU (lower) benchmark runs as described in Chapter 5, with idle power draw (dotted) and 85% quantile of active power draw. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.



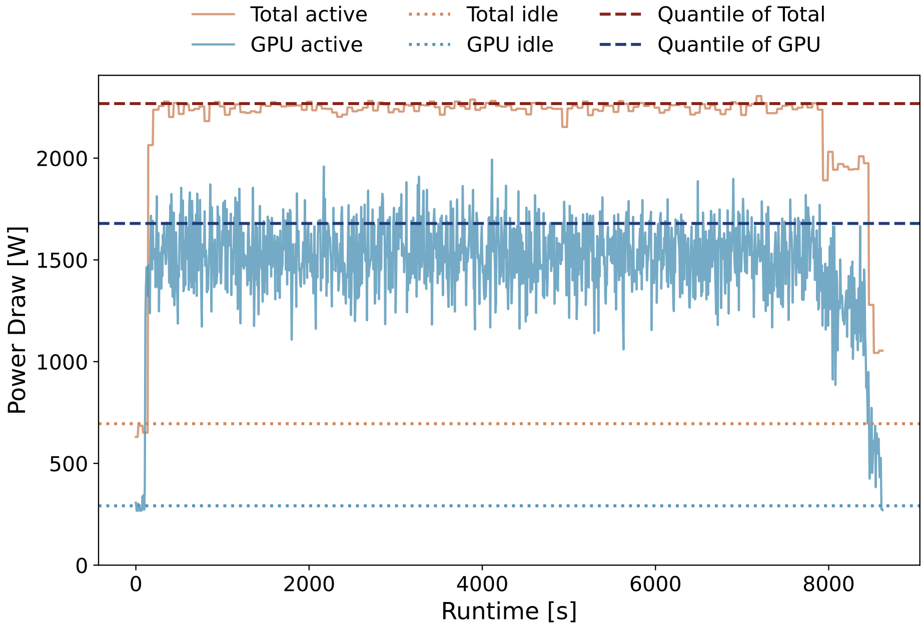
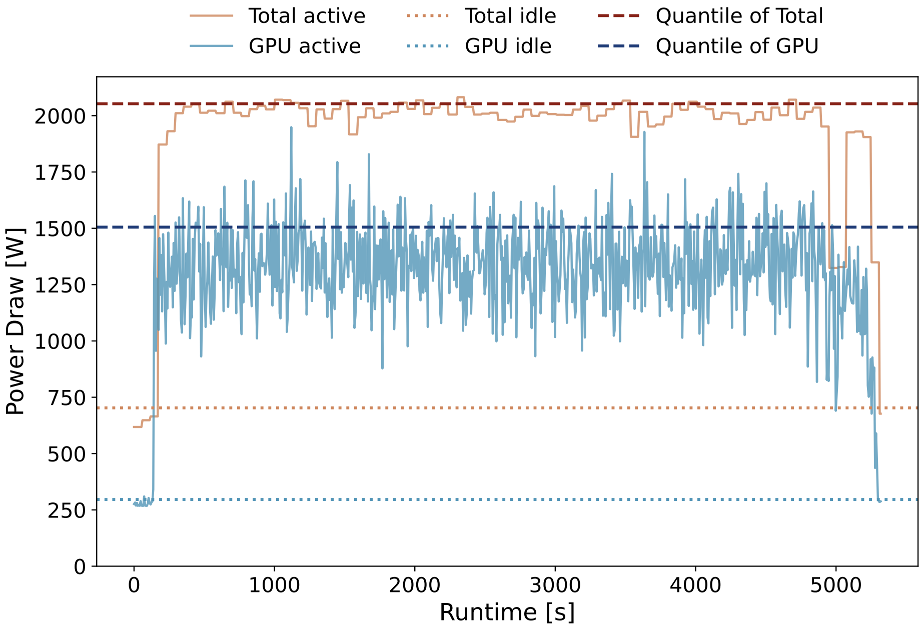


Figure A.10: Recorded power draw data of the *TauTransformer Training* workload in the 1-GPU per workload instance (upper) and 2-GPU per workload instance (lower) benchmark runs as described in Chapter 5, with idle power draw (dotted) and 85% quantile of active power draw. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.



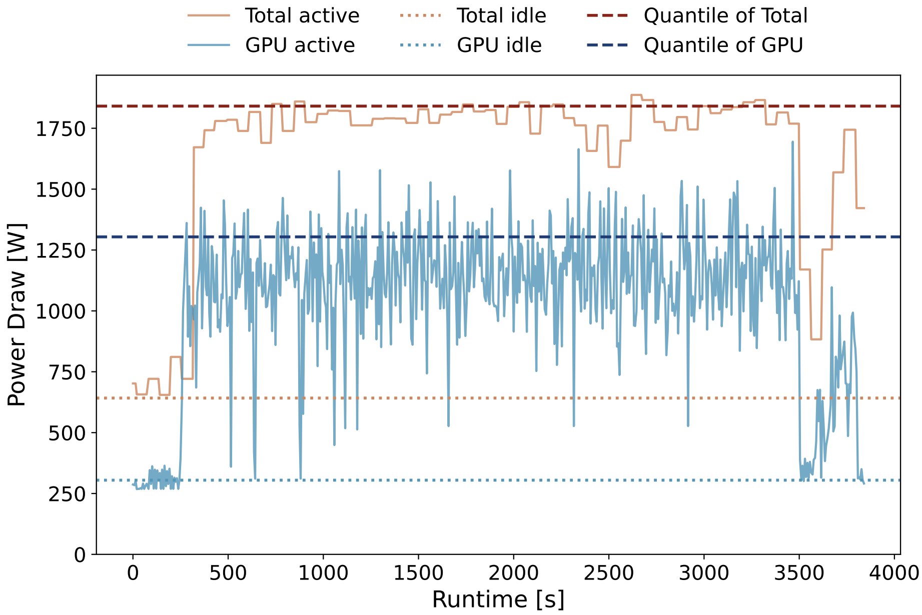
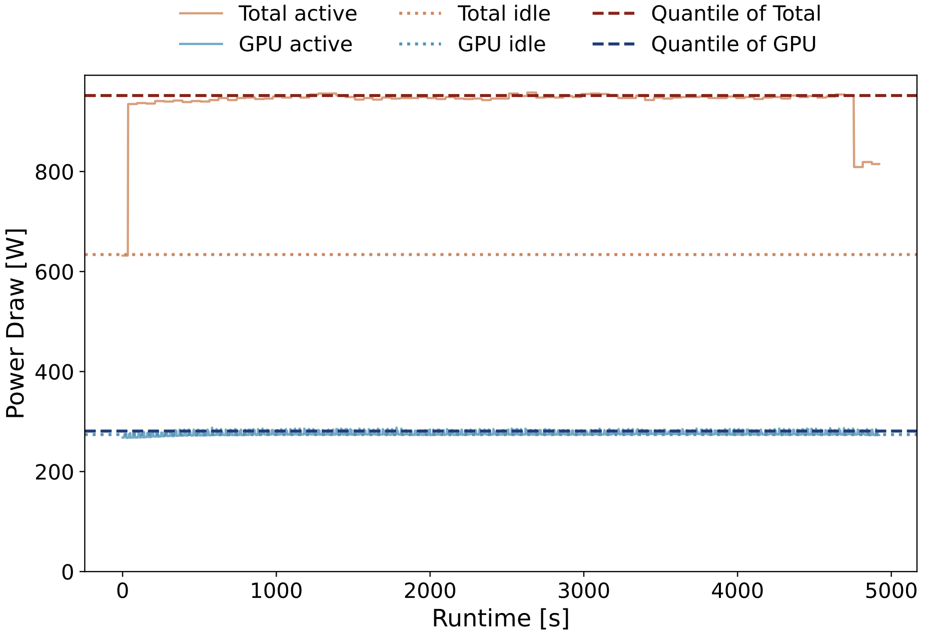


Figure A.11: Recorded power draw data of the *TauTransformer Training* workload in the 4-GPU per workload instance (upper) and 8-GPU per workload instance (lower) benchmark runs as described in Chapter 5, with idle power draw (dotted) and 85% quantile of active power draw. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.



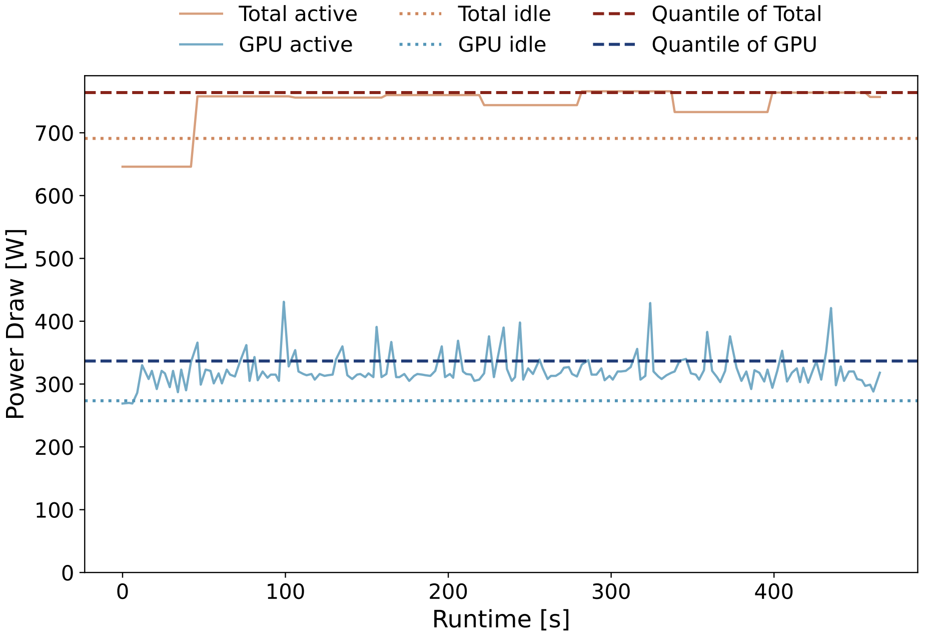


Figure A.12: Recorded power draw data of the *MSSM Training* workload in the CPU-only (upper) and baseline single GPU (lower) benchmark runs as described in Chapter 6, with idle power draw (dotted) and 85% quantile of active power draw. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.

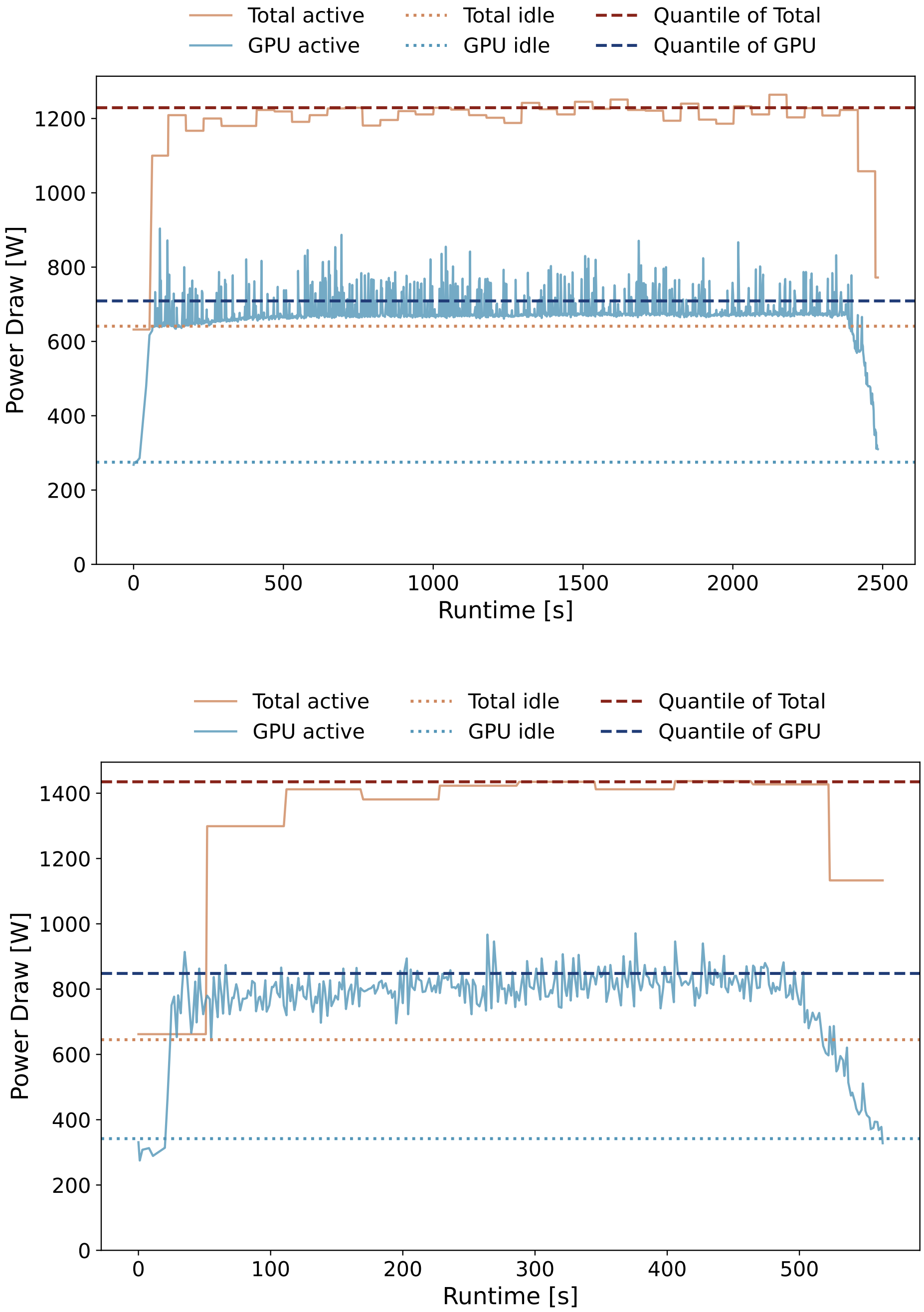


Figure A.13: Recorded power draw data of the *MSSM Training* workload in the Time-Slicing (upper) and MIG (lower) benchmark runs as described in Chapter 5, with idle power draw (dotted) and 85% quantile of active power draw. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.

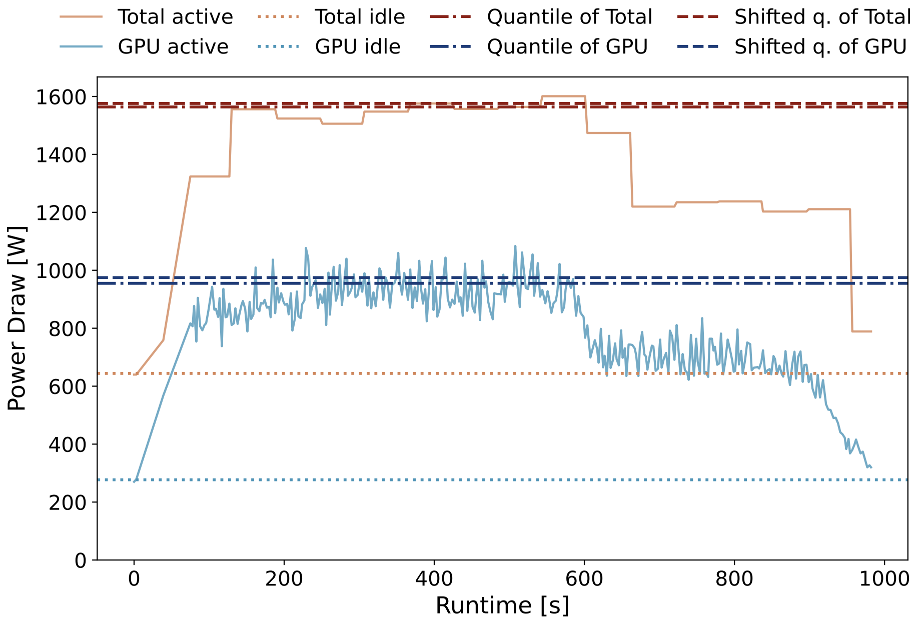
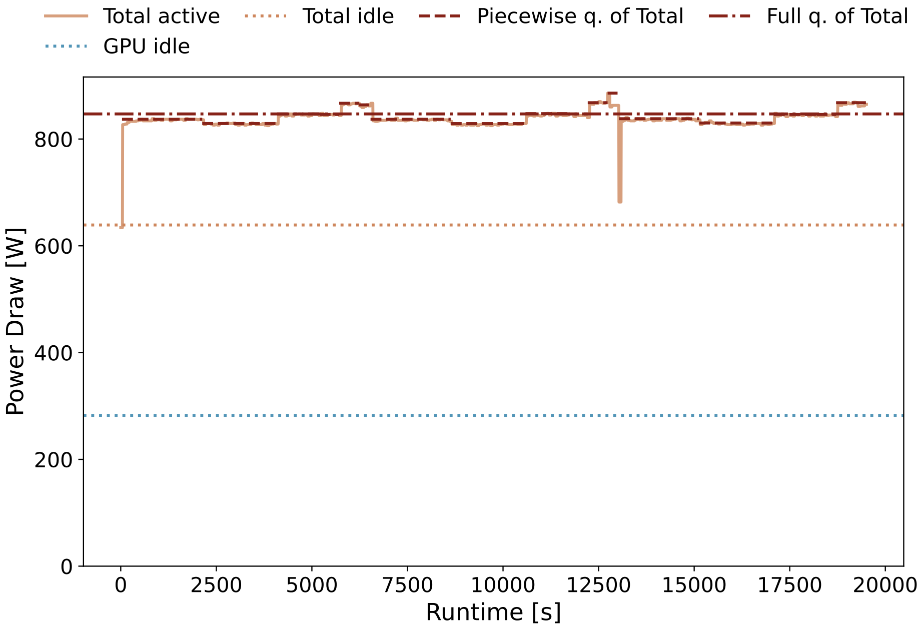


Figure A.14: Recorded power draw data of the *MSSM Training* workload MPS benchmark run as described in Chapter 5, with idle power draw (dotted) and 85% quantile of active power draw. Dot-dashed lines show the 85% quantiles of only the time frame from the start to the downwards step around 600 s. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.



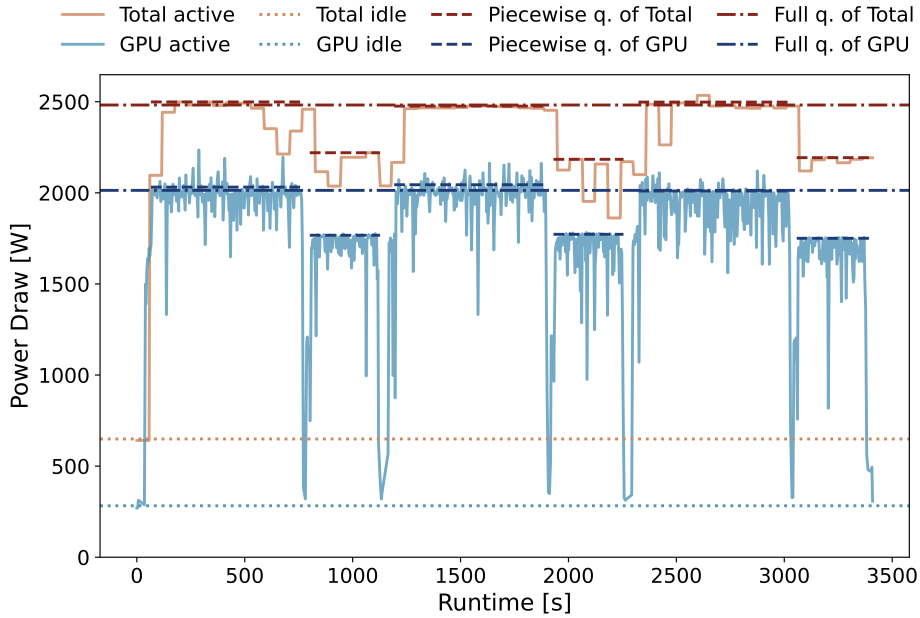


Figure A.15: Recorded power draw data of the $gg \to t\bar{t}gg$ *Generator* workload in the CPU-only (upper) and GPU-accelerated (lower) benchmark runs as described in Chapter 8, with idle power draw (dotted). The 85% quantile of the active power draw is shown for the entire duration of the runs (dot-dashed) and split by individual workloads and run repetitions (dashed). Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only.

A.10 MPS Performance Investigation

After compiling the *MSSM Training* benchmark results of Section 6.7, I decided to run an additional scan for the MPS scenario. The goal of the investigation was to understand better the nature of the performance discrepancy between the initial MPS performance scan outlined in Section 6.6.1 and the full benchmark performance described in Section 6.8. The scan was intended to demonstrate how the number of concurrent instances affects the runtime of the MPS setup at high levels of concurrency.

For this purpose, I reused the MPS setup as outlined for the full MPS benchmark run in Section 6.7.3, except that the number of active GPUs was varied from 1 to 8. In each setup, all active GPUs were fully occupied with 12 concurrent instances, and each instance was pinned to two CPU threads, as before. The results of the scan are listed in Table A.25 and illustrated in Figure A.16.

The scan's outcome was a surprise, revealing unexpected limitations in my benchmark setup. The runtime progression in terms of the number of concurrent instances was uneven. Between 1 and 5 active GPUs (12 to 60 concurrent instances), the mean runtime rose by 17.5%. Beyond that point, runtime growth accelerated sharply. With eight active GPUs (96 concurrent instances), the mean runtime increased by 60.5% compared to the one-GPU scenario. Total throughput painted a similar picture, stagnating at 5 active GPUs (60 concurrent instances), until the 8th GPU added 2% to performance compared to the setup with 7 active GPUs.

Finally, power draw continued to rise with the number of active GPUs, even though total throughput stagnated. For the MPS scenario scan in this section, I simulated a system with fewer than eight GPUs by removing the idle power draw of the inactive GPUs from the total power draw. As a result, the minimum energy per instance was observed on six active GPUs, corresponding to 72 concurrent instances of the *MSSM Training* workload. With 3.42 Wh per instance, it required 6.6% less energy than the setup with all eight GPUs (96 instances).

Figures A.17 and A.18 show the power draw and utilization timelines for six of the eight configurations tested as part of this MPS throughput scan. The behavior of the benchmark run remained stable up until 60 concurrent instances. With 72 or more concurrent instances, a step appeared in both the power draw and the utilization recorded throughout the benchmark runs. After a portion of the total runtime (roughly 60%), power draw and utilization decreased significantly, but did not return to idle levels.

This step coincided with a subset of the workload instances completing, while other instances were still ongoing. All instances of the workload were identical, with no reason why specific instances should have taken significantly longer than others. In this section, the group of instances that completed faster is referred to as the *Fast instances*, and the group that completed slower is referred to as the *Slow instances*.

Some details have stood out to us:

- The behavior was consistent across reruns of the benchmark scenario.

- All *Slow instances* showed the same speed before and after the point in time where

Table A.25: Summary of runtime, total throughput, and power draw for the second MPS throughput scan of the *MSSM Training* benchmark. The power draw uses an adjusted subset of the benchmark data to account for the fraction of the runtime during which the benchmark machine was not fully saturated. In addition, the idle power draw of inactive GPUs was subtracted from the power draw data. Details regarding the adjusted metrics are outlined in Appendix A.10.

# Active GPUs	1	2	3	4	5	6	7	8
# Concurrent instances	12	24	36	48	60	72	84	96
Avg. Runtime [min]	8.25	8.78	9.43	9.27	9.69	10.63	11.81	13.24
Throughput [inst./h]	87.3	164.0	229.0	310.7	371.5	406.3	426.7	435.1
Power draw [W]	612	775	962	1267	1313	1390	1521	1593
Energy / instance [Wh]	7.01	4.73	4.20	4.08	3.53	3.42	3.56	3.66

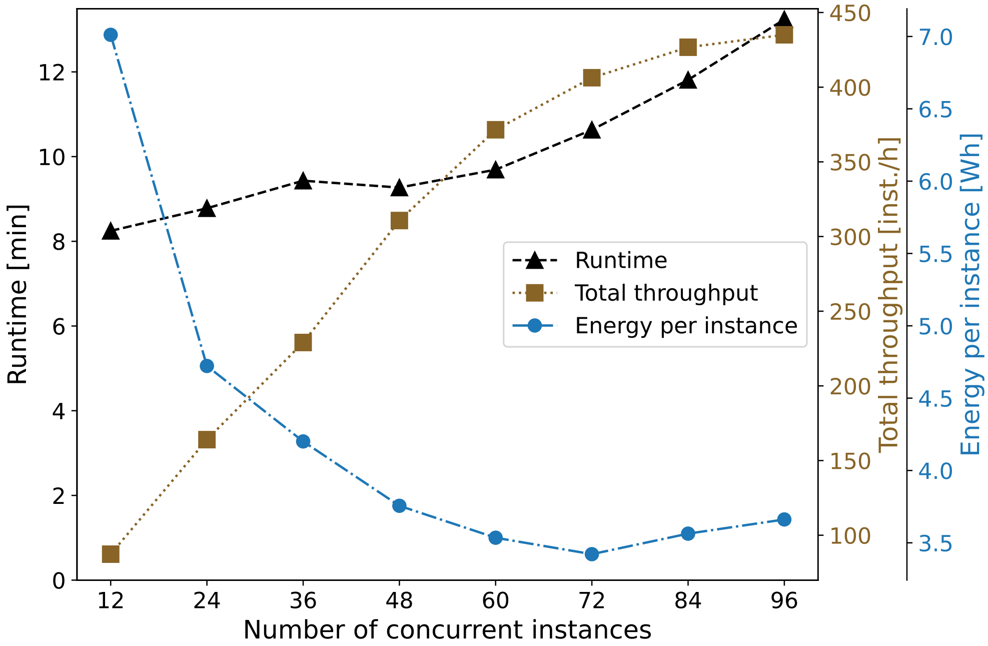


Figure A.16: Plot of average runtime, total throughput, and energy efficiency for the second MPS throughput scan of the *MSSM Training* benchmark. In addition, the idle power draw of inactive GPUs was subtracted from the power draw data used to calculate energy efficiency.

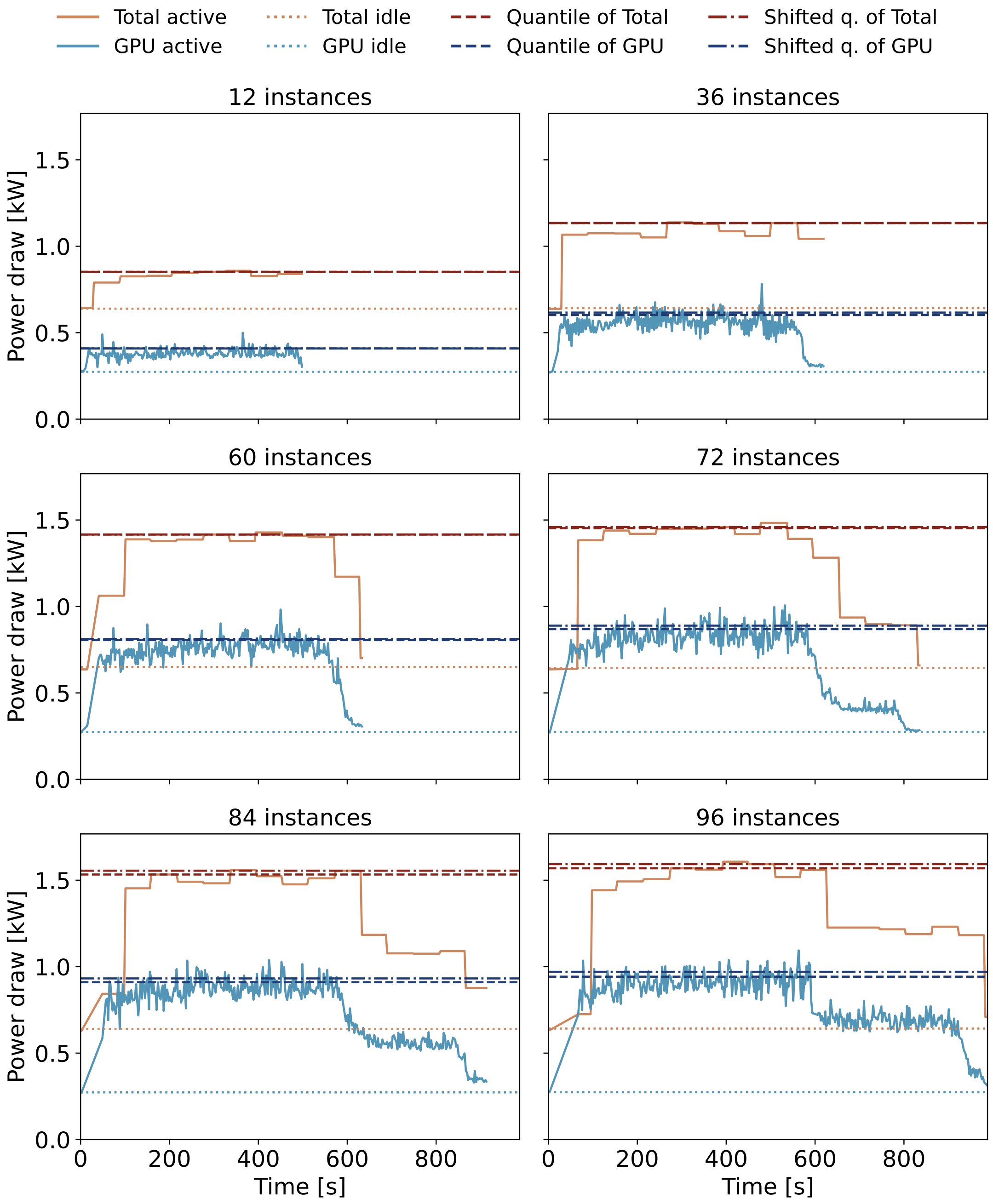


Figure A.17: Power draw timeseries for the 60, 72, 84, and 96 concurrent instance scenarios of the MPS throughput scan described in Appendix A.10. All subplots have a common axis scaling. Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only. Horizontal lines represent the values for the 85% quantile of the active power draw (dashed), the 85% quantile of the active power draw when only taking into account data points where all instances were still running (dot-dashed), and the 85% quantile of the idle power draw (dotted).

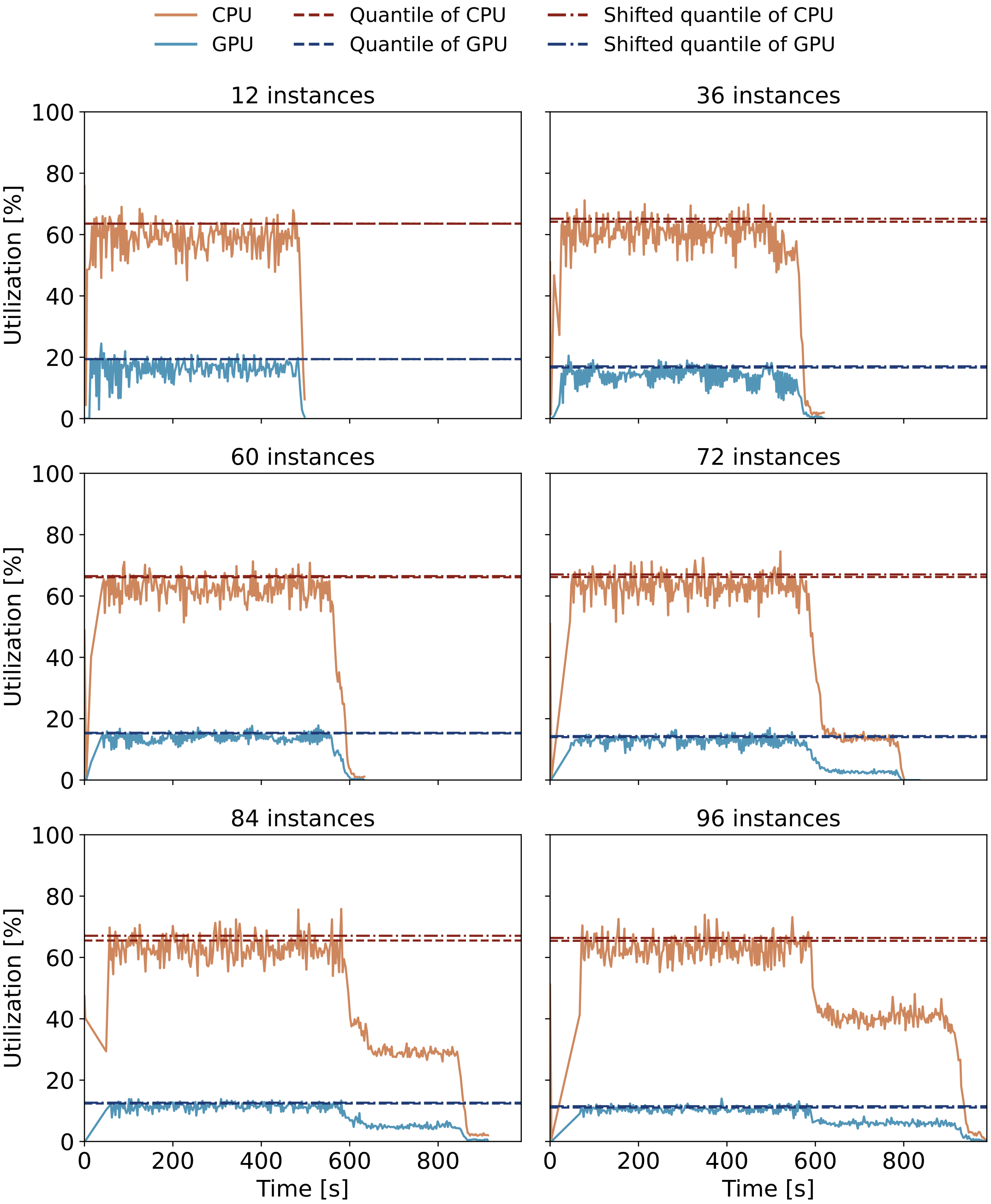


Figure A.18: CPU and GPU utilization timeseries for the 60, 72, 84, and 96 concurrent instance scenarios of the MPS throughput scan described in Appendix A.10. All subplots have a common axis scaling. Orange lines represent the CPU utilization, while blue lines represent the GPU utilization. The horizontal lines represent the 85% quantiles of utilization (dashed) and the 85% quantile of utilization for data points where all instances were still running (dotted).

Table A.26: Summary of on which GPUs the *Slow* and *Fast instances* were located throughout the benchmark scenarios and how many of the *MSSM Training* workload instances were assigned to shared CPU threads via Simultaneous Multithreading (SMT). GPUs were identified by their CUDA index on the machine.

# Active GPUs	Fast GPUs	Slow GPUs	# Instances with shared threads
1	0	—	0
2	0-1	—	0
3	0-2	—	0
4	0-3	—	0
5	0-4	—	0
6	1-4	0, 5	16
7	2-4	0-1, 5-6	40
8	3-4	0-2, 5-7	64

the *Fast instances* completed. The *Slow instances* did not accelerate once the *Fast instances* completed, as measured by the workload-internal progress monitoring.

- The split into *Slow* and *Fast instances* happened roughly along the boundaries of the GPUs. In other words, nearly all instances placed on the same GPU showed similar runtime speeds.

- The GPUs with *Slow instances* follow a pattern. Table A.26 summarizes which GPUs hosted the *Slow instances* and *Fast instances* for each of the benchmark scenarios of this scan.

Based on these observations, I conclude that the behavior was caused by the use of logical CPU threads provided through SMT. The point at which the *Slow instances* appear coincides with a step over 64 concurrent instances of the benchmark workload, each of which is assigned two CPU threads. With 128 physical and 128 logical CPU threads, this is also where work is pinned to logical CPU threads. As listed in Table A.26 for 72 *MSSM Training* workload instances, 56 were assigned a private physical CPU thread, while eight pairs of 2 had to share their assigned CPU resources. CPU threads were assigned to the workload instances in blocks of two, starting from 0, and the numbering of CPU threads after 127 refers to the range of 0 to 127 in the same order. As a result, both instances assigned to the 1st and 5th GPUs were slowed down due to CPU resource contention. For 84 and 96 workload instances, this pattern continues, with more and more instances sharing CPU resources rather than having exclusive access.

In the case of 96 concurrent instances, 32 have exclusive access to real CPU threads, while 64 share their access with another instance. In a comparison between a workload instance with private CPU threads and a pair of instances that share CPU resources, the latter achieves higher total throughput. Based on the performance of the 60- and 96-instance runs of the benchmark scan, total throughput increases by 14.9% for instances that share physical CPU resources. The impact of SMT on performance varies widely and depends on the workload, though results in the 15-30% range are common.

While SMT provides a plausible explanation for the performance ceiling observed in the MPS benchmark scenario, it does not explain why a CPU-based limitation affects a workload presumed to be GPU-bound. CPU utilization indicated 66.6% for the MPS workload, even in runs where total performance seemed limited by CPU. It is unclear which CPU component limits overall throughput, as the CPU utilization measurement fails to detect the bottleneck.

I conclude this excursion by summarizing the findings relevant to the MPS benchmark scenario: Degrading performance for high degrees of concurrency in the MPS scenario is caused by CPU resource contention. The direct cause is the assignment of workload instances to logical cores beyond the actual 128 CPU cores. As a result, a subset of instances shares CPU resources and is slowed down. This behavior is reproducible and affects instances independent of their workload. Even after non-competing CPU resources are freed, instances with competing resources will not use them due to CPU pinning. Therefore, my benchmark assumptions for total throughput in Section 6.2 hold for the MPS scenario.

A.11 MPS Production Setup

This appendix details the MPS setup used in production for the MPS-enabled machines in our batch system.

In the first step, the scheduler transforms jobs. If they do not set the request_GPUMemoryMB variable themselves, it is set to the maximum value of the worker machine. For the V100S GPUs, this value is 32 GB. In addition, the job request is denied if the job provides a request_GPUMemoryMB value without also requesting a GPU, or if it requests more than one GPU. The relevant code for this step is shown in Listing 1.

In the second step, a job with a request_GPUMemoryMB value is matched to one of the MPS-enabled machines[4]. Each machine with the MPS setup provides eight replicas of each physical GPU. They are all assigned to a common partitionable slot that limits the total amount of GPUMemoryMB among these to the Video Random-Access Memory (VRAM) limit of a V100S. Listing 2 shows the code necessary to partition the slots.

In the third step, the environmental variables CUDA_MPS_PIPE_DIRECTORY and MPS_PINNED_DEVICE_MEM_LIMIT are set during the startup of the job. In our case, this happens during a Python-based Docker wrapper script. The relevant commands for the job process are shown in Listing 3 as pseudo code. The value of the request_GPUMemoryMB variable is read directly from the job's classadd file.

In the final step, the job runs are supervised by MPS. For this purpose, the service shown in Listing 4 is added, which starts a normal MPS server with an adjusted CUDA_MPS_PIPE_DIRECTORY location. This ensures that only GPU processes with the HTCondor-adjusted non-default pipe directory interact with the MPS daemon. Any other GPU process is subject to the default Time-Slicing behavior outlined in Section 6.6.1. In our specific setup, all jobs ran under a common user account. If this setup were used with multiple unique users, the -multiuser-server option of nvidia-cuda-mps-control is necessary. This option allows MPS to optimize GPU processes attributed to more than one user at a time. I have not tested this feature in production and can only assume that it would work, based on the documentation.

[4] Worker machines without the MPS setup can still run jobs with a request_GPUMemoryMB, though they ignore the variable entirely and are therefore limited to one job per GPU.

```
# Job transformation, e.g. set defaults
JOB_TRANSFORM_NAMES = GPUJobs
# Add RequestGPUMemoryMB to maximum if GPU requested
JOB_TRANSFORM_GPUJobs @=end
   # set default to hole GPU memory by requesting 1 GPU
   REQUIREMENTS RequestGPUs =?= 1
   DEFAULT RequestGPUMemoryMB TARGET.TotalSlotGPUMemoryMB
@end
# Check job attributes during submission
SUBMIT_REQUIREMENT_NAMES = GPUMemoryCheck
# Abort submission if GPUMemoryMB is requested with 0 or >1 GPU
SUBMIT_REQUIREMENT_GPUMemoryCheck = \
ifThenElse( isUndefined(RequestGPUMemoryMB), \
True, RequestGPUMemoryMB > 0 && RequestGPUs=?=1)
SUBMIT_REQUIREMENT_GPUMemoryCheck_REASON = \
"You request GPUMemoryMB without requesting a GPU or more than one GPU!"
```

Listing 1: HTCondor configuration to add defaults to GPU jobs and reject mis-configured jobs. Code is added to the batch system's schedulers.

```
# Hard code GPU properties
NUM_MAX_GPU_DIVIDE = 8
NUM_GPUS = 8
DEVICE_MEMORY_PER_GPU = 32000
# Define GPUMemoryMB as new resource
MACHINE_RESOURCE_GPUMemoryMB = $(DEVICE_MEMORY_PER_GPU) * $(NUM_GPUS)
# Adapt GPU detection to return multiple copies for each GPU
GPU_DISCOVERY_EXTRA = -repeat $(NUM_MAX_GPU_DIVIDE) -packed
# Define partitionable slots with all discovered GPUs that share a name
# Each slot shares a common GPUMemoryMB pool
SLOT_TYPE_1_PARTITIONABLE = TRUE
NUM_SLOTS_TYPE_1 = $(NUM_GPUS)
SLOT_TYPE_1 = \
GPUMemoryMB=$(DEVICE_MEMORY_PER_GPU), GPUs=$(NUM_MAX_GPU_DIVIDE), auto
# Enable the condor user to read from the run dir to access the classads
JOB_EXECDIR_PERMISSIONS = world
# Only allow single GPU jobs on this machine
GPU_CONDITION = ifthenelse( \
IsUndefined(Target.RequestGPUs), true, Target.RequestGPUs =?= 1 )
START = $(START) && $(GPU_CONDITION)
# Add the nvidia-mps socket to the Docker mounts
DOCKER_VOLUMES = $(DOCKER_VOLUMES) MPS
DOCKER_VOLUME_DIR_MPS = /tmp/nvidia-mps:/tmp/nvidia-mps
DOCKER_MOUNT_VOLUMES = $(DOCKER_MOUNT_VOLUMES) MPS
```

Listing 2: HTCondor configuration to provide shared GPU slots to jobs that request less than the maximum available VRAM. Code is added to the worker machines that should use MPS.

```
# Export necessary environmental variables for MPS
export CUDA_MPS_PINNED_DEVICE_MEM_LIMIT= \
"${CUDA_VISIBLE_DEVICES}\=${RequestedGPUMemoryMB}MB"
export CUDA_MPS_PIPE_DIRECTORY="/tmp/nvidia-mps/condor"
```

Listing 3: Pseudo code that adds the MPS-specific environmental variables to the HTCondor job environment. The real code is Python-based and part of a Docker wrapper. The `RequestedGPUMemoryMB` variable is read from the job `classadd` file during the setup.

```
[Unit]
Description=NVIDIA MPS daemon
Documentation=https://docs.nvidia.com/deploy/mps/index.html
After=condor.service
# Restart service if condor.service is restarted
PartOf=condor.service
[Service]
Type=simple
ExecStart=/usr/bin/nvidia-cuda-mps-control -f
Environment="CUDA_MPS_PIPE_DIRECTORY=/tmp/nvidia-mps/condor"
# The condor.service starts with a non-standard pipe directory
# Only CUDA processes that share the pipe directory will use MPS
ExecStartPre=/bin/sleep 20
Restart=on-failure
RestartSec=10s
[Install]
WantedBy=multi-user.target
```

Listing 4: Service to start an MPS daemon with a custom pipe directory location.

A.12 $gg \to t\bar{t}gg$ *Generator* Benchmark Code Changes

The `mg5amc-madgraph4gpu-2022-bmk:V0.7` benchmark container used in Chapter 8 was modified to allow for multiple GPUs to be used. Without modification, only the GPU with the first index receives the designated number of copies of the benchmark workload. The code in Listing 5 was added at the start of the `doOne` function to distribute the copies across the available GPUs.

```
...
# Get list of available GPUs
GPU_LIST=($(nvidia-smi -L | sed 's/.*\(GPU-.*\))/\1/'))
# Determine GPU index, based on copy index
index=$((($1-1)%${#GPU_LIST[@]}))
# Pin copy to GPU
export CUDA_VISIBLE_DEVICES=${GPU_LIST[${index}]}
echo "[doOne ($1)] Using GPU with UUID: ${CUDA_VISIBLE_DEVICES}"
...
```

Listing 5: Added code to allow the $gg \to t\bar{t}gg$ *Generator* benchmark to use all GPUs of a multi-GPU machine. Workload copies are assigned to the GPUs in a round-robin manner.

The code first parses the Universally Unique Identifier (UUID) of the GPUs from SMI. It then determines the target GPU based on the workload copy index (`$1`) in a round-robin manner. Finally, the benchmark process is pinned to the designated GPU via the `CUDA_VISIBLE_DEVICES` environmental variable. Notably, this method for determining available GPUs is not compatible with the MIG feature outlined in Section 6.6.2. While it's possible to adapt the UUID `regex` to start with `MIG` instead of `GPU`, allowing for both variants will clash in the MIG setup. In that case, both the GPU and its respective MIG are listed, even though only the MIG can accept work, resulting in over-assignment.

A.13 Shared Madgraph4GPU Benchmark

This section covers my investigation into possible performance improvements for the *gg* $\rightarrow$ *$t\bar{t}gg$ Generator* benchmark. The main goal of this excursion was to measure how performance changes if more than one Copy of the *gg* $\rightarrow$ *$t\bar{t}gg$ Generator* workload was placed on an NVIDIA A100 GPU. The benchmark documentation describes a roughly 10% drop in overall performance for this use case. This behavior is expected, as the default use of GPU hardware by multiple processes can lead to performance-limiting issues. In Chapter 6, I explored both this limiting behavior of Time-Slicing (Section 6.6.1), as well as options to improve on this behavior. I decided on an MPS approach to cross-check whether the drop in performance also applied with an optimized multi-tenant GPU usage. The MPS software was introduced in Section 6.6.3, and appeared to be the easiest to implement for the *gg* $\rightarrow$ *$t\bar{t}gg$ Generator* benchmark.

The core concept of MPS is to allow for multiple concurrent processes to submit kernels to a common GPU. Generally, this is not necessary for scalable processes like MadGraph4GPU, which can already utilize multiple streams to interleave kernels. Because this feature was seemingly not implemented in the benchmark code, MPS presented itself as a rudimentary replacement for testing purposes.

For the sake of completeness, I first tested the original GPU run of the *gg* $\rightarrow$ *$t\bar{t}gg$ Generator* benchmark with eight additional copies of the benchmark workload, but no further changes. Because Copies were distributed to the eight available GPUs in a round-robin manner, each GPU received two Copies compared to the original singular Copy. As claimed, the benchmark's total throughput decreased by roughly 10% compared to the original GPU setup.

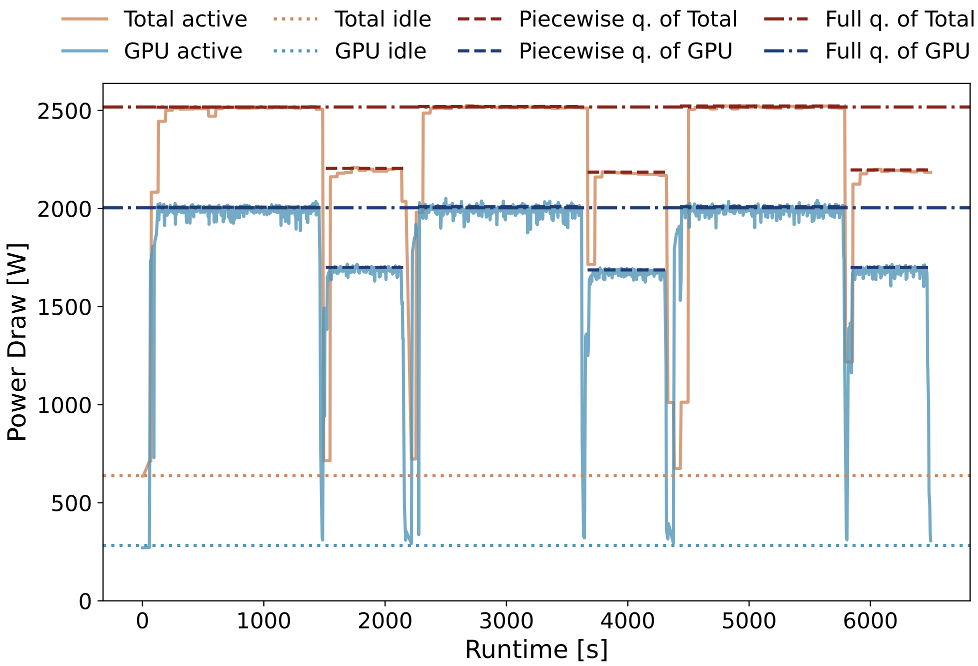
The second follow-up run was identical to the first, in that two Copies of the benchmark workload were placed on each GPU, though this time an MPS daemon was started to optimize the performance of the two processes per GPU.

A.13.1 Benchmark Results

Instead of listing the quantile metrics of power draw and utilization for this benchmark comparison, I focus on the throughput scores in this section. The quantile returns nearly identical values, whereas the time series of the respective metrics provides much more variance.

Figure A.19 shows the power draw and hardware utilization during the MPS-assisted run, respectively. Compared with Figures 8.3 and 8.4, the raw values of both GPU power draw and GPU utilization appear much smoother. This indicates reduced GPU flickering and suggests an overall more stable performance.

Table A.27 shows the respective scores that were achieved by the original GPU run and the MPS-assisted GPU run in the upper section. Based solely on these values, an MPS-assisted run with two concurrent Copies per GPU increased performance by 75%. There was no significant increase in power draw during the MPS-assisted run, so this improvement would also directly translate into improved energy efficiency. Unfortunately, there is reason to doubt this increase.



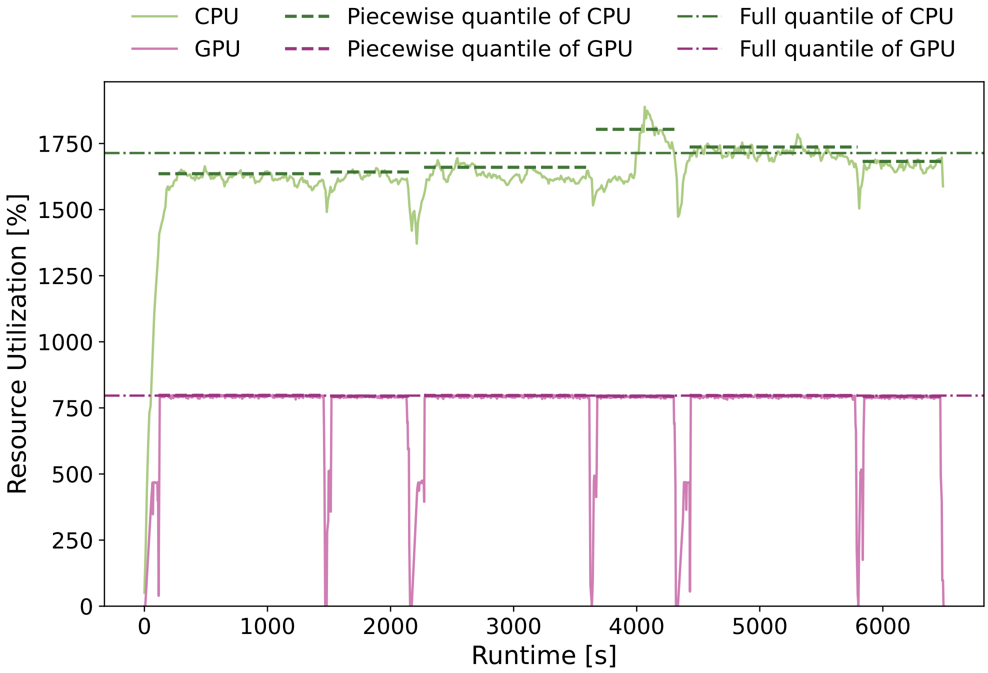


Figure A.19: Recorded power draw (upper) and utilization (lower) data of the $gg \rightarrow t\bar{t}gg$ *Generator* workload in the MPS-supported Benchmark Run with two Copies of the benchmark workload per GPU. Idle power draw is shown as dotted lines. The 85% quantile of the active power draw/utilization is shown for the entire duration of the runs (dot-dashed) and split by individual workloads and run repetitions (dashed). Orange lines represent the total power draw of the entire machine, while blue lines represent the GPU's power draw only. Green lines represent the CPU utilization, while magenta lines represent the GPU utilization.

Table A.27: Throughput of *$gg \rightarrow t\bar{t}gg$ Generator* benchmark for configuration with and without MPS. The non-MPS run is identical to the default GPU run of Section 8.3. For the MPS run, two Copies of the benchmark workload were placed on each GPU. The upper section states the throughput of both Variations when only the `SigmaKin` step is used for the throughput calculation, and the `CpDTHwgt` step is treated as overhead. The second section outlines the scores if the `CpDTHwgt` step is not considered overhead.

Configuration	1 Copy without MPS	2 Copies with MPS
Score (`SigmaKin`) [10^6 Events / s]		
Double precision	6.299	11.022
Float precision	14.219	25.075
Score (`SigmaKin` + `CpDTHwgt`) [10^6 Events / s]		
Double precision	6.292	6.441
Float precision	14.201	13.692

A.13.2 Results Discussion

While the reported score draws a clear picture, other data from the Benchmark Run contradict it. Twice the number of Copies per GPU also results in twice the number of total events processed in the Benchmark Run. With the score increasing by nearly a factor of two, this should have resulted in a total runtime close to that of the original GPU Benchmark Run. Instead, the total runtime for the MPS-assisted Benchmark Run was nearly twice as long as the original GPU run.

There are two possible reasons for this discrepancy between throughput and runtime: Either the overhead the Benchmark Run experienced between individual Variations had risen significantly, or the Variations' runtime had increased. The first option is rejected based on the data shown in Figure A.19. The time between Variations did not increase compared to 8.2. Therefore, the focus shifts to the second possible cause: that the runtime of the Variations themselves had increased by a factor of two.

To investigate this phenomenon, I reviewed the Benchmark Run's log files for the runtime breakdown of the steps. As outlined in Section 8.2, only the step `SigmaKin` was considered for the computation of the throughput. To determine whether any specific non-`SigmaKin` step was overhead or another step that should have been considered in addition to the `SigmaKin` step required nuance.

Figure A.20 showcases the breakdown of the time per event for the original and the MPS-assisted GPU runs. Further tests with three and four Copies of the benchmark workload per GPU returned nearly identical runtime breakdowns as the run with two Copies per GPU.

As the primary goal of the *$gg \rightarrow t\bar{t}gg$ Generator* benchmark was to measure the performance of the `SigmaKin`, it was reasonable to consider only that specific step for the

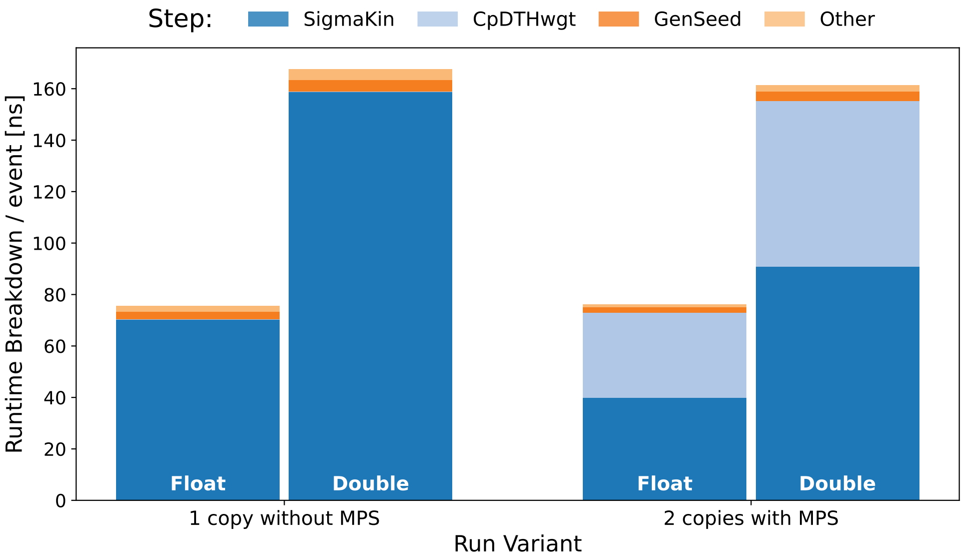


Figure A.20: Runtime breakdown of the most prominent time steps of the $gg \to t\bar{t}gg$ *Generator* benchmark for configurations with and without MPS. The non-MPS run was identical to the default GPU run of Section 8.3. For the MPS run, two Copies of the benchmark workload were placed on each GPU.

performance metric. Still, what was and was not part of that step should be considered carefully to reflect real-world applications.

Only 52% of the total runtime per event was attributed to the `SigmaKin` step in the MPS-supported Benchmark Run. The majority of the remaining runtime was instead attributed to the `CpDTHwgt` step, 44% and 40% for the float and double precision Variations, respectively. A runtime fraction of this magnitude cannot be ignored without good reason. The lower section of Table A.27 presents the throughput if the `CpDTHwgt` step were added to the `SigmaKin` step for throughput calculation.

A.13.3 Time Attribution

The increase in total runtime is attributed to the `CpDTHwgt` step, which was far less dominant in the original GPU run. Still, the question of whether this step should be considered overhead remains unresolved. The response to my initial inquiries with the original creators of the benchmark was that the benchmark was intended to only consider the `SigmaKin` step in the calculation of throughput. Therefore, the improved result achieved in the MPS-assisted run should have been valid.

The general argument was that the `CpDTHwgt` step transferred the randomly generated weights from the GPU to the CPU before they were used in the `SigmaKin` step on the GPU. It did this before the `SigmaKin` step, so it should not have had a direct impact on the performance of the steps that follow it. After the transfer, the weights were not used on the CPU side, so in theory, there should be no functional reason to include the step.

Still, this reasoning did not explain why the time attributed to the copy step increased from less than 1% to over 40% by adding a second Copy. The increased transfer time may have been a direct result of interplay with the MPS software. I did not experience this phenomenon in the benchmarks of Chapters 6 or 7, so this option is unlikely. What is more likely is that by lowering the runtime per event of the `SigmaKin` step, other instructions became the bottleneck.

CUDA GPU code heavily interleaves operations, such as data transfers and kernel execution, whenever possible. In compiled code, individual pieces may be rearranged for optimization based on compiler arguments. As a result, timekeeping via CPU-based instructions (such as `chrony`) can struggle to track time spent during GPU-offloaded parts. If multiple instructions were parallelized, it's possible that some of the time spent was attributed to only a single step, even though multiple steps occurred at once.

The initial absence and sudden increase of the `CpDTHwgt` step could be explained in this manner. On the one hand, if the connection between CPU and GPU became the limiting step in the total runtime, then it could no longer be entirely hidden by the `SigmaKin` step. On the other hand, if the GPU runtime's attribution were this volatile, it could not be ensured that the time assigned to step `CpDTHwgt` didn't partially belong to the `SigmaKin` step.

Another point to consider is that even though the total runtime per event was mostly unchanged, the utilization metric in Figure A.19 still reported a steady, near 100%, utilization throughout the MPS-supported run. If 40% of the runtime were taken up by transfer-overhead, then similar spikes in utilization as seen in the initial GPU run would have been expected. The fact that they were not present implied that there was little to gain in total throughput compared to the initial GPU run.

Therefore, while the MPS-assisted benchmark configuration reported nearly twice the baseline GPU throughput, it should not be considered as a conclusive result due to the inconsistencies in time attribution. Instead, I consider this as a promising direction for future work: Additional research will be necessary to confirm whether total throughput was increased in this configuration, or if the result was due to an accounting error in the timekeeping. Depending on the outcome, MPS or multi-stream approaches could significantly improve performance for the $gg \to t\bar{t}gg$ *Generator* use case.

Acronyms

ALICE A Large Ion Collider Experiment.

ATLAS A Toroidal LHC Apperatus.

CCE Categorical Cross-Entropy.

CERN Organisation européenne pour la recherche nucléaire.

CL Confidence Level.

CMS Compact Muon Solenoid.

DCGM NVIDIA Data Center GPU Manager.

DM Decay Mode.

DMI Decay Mode Integer.

EB ECAL Barrel.

ECAL Electromagnetic Calorimeter.

EE ECAL Endcap.

EM Electro-Magnetic.

ES ECAL Preshower.

EWK Electroweak.

FNR False Negative Rate.

FPR False Positive Rate.

GFS Gaussian Sum Filter.

GRACT Graphic Activity.

GridKa Grid Computing Centre Karlsruhe.

HB HCAL Barrel.

HCAL Hadronic Calorimeter.

HE HCAL Endcap.

HEP High-Energy Physics.

HF HCAL Forward.

HL-LHC High-Luminosity Large Hadron Collider.

HO HCAL Outer.

HoreKa Hochleistungsrechner Karlsruhe.

HPS Hadrons-Plus-Strips.

HTC High Throughput Compute.

IPMI Intelligent Platform Management Interface.

LHC Large Hadron Collider.

LHCb Large Hadron Collider Beauty.

MIG Multi Instance GPU.

ML Machine Learning.

MPS Multi-Process Service.

MSSM Minimal Supersymmetric extension of the Standard Model.

NHR National High Performance Computing.

NMSSM Next-to-Minimal Supersymmetric extension of the Standard Model.

NN Neural Network.

NNLO Next-to-Next-to Leading Order.

NVME Nonvolatile Memory Express.

OOM Out-Of-Memory.

PCIe Peripheral Component Interconnect Express.

PF Particle Flow.

PFcand Particle Flow candidate.

QCD Quantum Chromodynamics.

QCD jets quark and gluon induced jets.

RAM Random-Access Memory.

ROC Receiver Operating Characteristic.

S&M Shuffle and Merge.

SIMD Single-Instruction Multiple-Data.

SM Standard Model.

SMACT Streaming Multiprocessor Activity.

SMI System Management Interface.

SMOCC Streaming Multiprocessor Occupancy.

SMT Simultaneous Multithreading.

TauT TauTransformer.

TDP Thermal Design Power.

TNR True Negative Rate.

TOpAS Throughput Optimized Analysis System.

TPR True Positive Rate.

UUID Universally Unique Identifier.

VRAM Video Random-Access Memory.

WLCG Worldwide LHC Computing Grid.

WP Working Point.

List of Figures

List of Tables

References

[1] K. Bos et al. „LHC computing Grid: Technical Design Report. Version 1.06 (20 Jun 2005)“. Technical design report. LCG. Geneva: CERN, 2005.

[2] S. Chatrchyan et al. „Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC“. *Physics Letters B* 716.1 (Sept. 2012), pp. 30–61. ISSN: 0370-2693.
DOI: 10.1016/j.physletb.2012.08.021.

[3] G. Aad et al. „Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC“. *Physics Letters B* 716.1 (Sept. 2012), pp. 1–29. ISSN: 0370-2693.
DOI: 10.1016/j.physletb.2012.08.020.

[4] E. S. Group. „2020 Update of the European Strategy for Particle Physics“. Tech. rep. Geneva, 2020.
DOI: 10.17181/ESU2020.

[5] I. Béjar Alonso et al., eds. „High-Luminosity Large Hadron Collider (HL-LHC): Technical Design Report“. Vol. CERN-2020-010. ISBN 978-92-9083-587-5. CERN Yellow Reports: Monographs, 2020.

[6] CMS Collaboration. „The CMS experiment at the CERN LHC“. *Journal of Instrumentation* 3.08 (Aug. 2008), S08004.
DOI: 10.1088/1748-0221/3/08/S08004.

[7] V. Khachatryan et al. „The CMS trigger system“. *Journal of Instrumentation* 12.01 (Jan. 2017), P01020.
DOI: 10.1088/1748-0221/12/01/P01020.

[8] C. O. Software and Computing. „CMS Phase-2 Computing Model: Update Document“. Tech. rep. Geneva: CERN, 2022.

[9] L. Evans and P. Bryant. „LHC Machine“. *Journal of Instrumentation* 3.08 (Aug. 2008), S08001.
DOI: 10.1088/1748-0221/3/08/S08001.

[10] CERN. *Full house in the LHC.* https://home.web.cern.ch/news/news/accelerators/accelerator-report-full-house-lhc. Accessed: 2025-09-15. Sept. 2023.

[11] A. Sirunyan et al. „Particle-flow reconstruction and global event description with the CMS detector“. *Journal of Instrumentation* 12.10 (Oct. 2017), P10003. DOI: `10.1088/1748-0221/12/10/P10003`.

[12] „Reconstruction and identification of τ lepton decays to hadrons and $\nu\tau$at CMS“. *Journal of Instrumentation* 11.01 (Jan. 2016), P01019–P01019. ISSN: 1748-0221. DOI: `10.1088/1748-0221/11/01/p01019`.

[13] CMS Collaboration. „Identification of tau leptons using a convolutional neural network with domain adaptation in the CMS experiment“. Tech. rep. Geneva: CERN, 2025. DOI: `10.17181/CERN.K8AI.EM4N`.

[14] A. Pais and S. B. Treiman. „How Many Charm Quantum Numbers are There?“ *Phys. Rev. Lett.* 35 (23 Dec. 1975), pp. 1556–1559. DOI: `10.1103/PhysRevLett.35.1556`.

[15] H. Murayama. *Physics Beyond the Standard Model and Dark Matter.* 2007. arXiv: `0704.2276 [hep-ph]`.

[16] D. J. Griffiths. „Introduction to elementary particles“. Physics textbook. New York, NY: Wiley, 2008.

[17] W. N. Cottingham and D. A. Greenwood. „An Introduction to the Standard Model of Particle Physics“. 2nd ed. Cambridge University Press, 2007.

[18] Particle Data Group Collaboration. „Review of Particle Physics“. *Phys. Rev. D* 110 (3 Aug. 2024), p. 030001. DOI: `10.1103/PhysRevD.110.030001`.

[19] N. Cabibbo. „Unitary Symmetry and Leptonic Decays“. *Phys. Rev. Lett.* 10 (12 June 1963), pp. 531–533. DOI: `10.1103/PhysRevLett.10.531`.

[20] M. Kobayashi and T. Maskawa. „CP-Violation in the Renormalizable Theory of Weak Interaction“. *Progress of Theoretical Physics* 49.2 (Feb. 1973), pp. 652–657. ISSN: 0033-068X. DOI: `10.1143/PTP.49.652`. eprint: `https://academic.oup.com/ptp/article-pdf/49/2/652/5257692/49-2-652.pdf`.

[21] Particle Data Group Collaboration. „Review of particle physics“. *Phys. Rev. D* 110.3 (2024), p. 030001. DOI: `10.1103/PhysRevD.110.030001`.

[22] W. Commons. *File: Standard Model of Elementary Particles.svg — Wikimedia Commons, the free media repository.* [Online; accessed 20-August-2025]. 2025.

[23] P. W. Higgs. „Broken Symmetries and the Masses of Gauge Bosons“. *Physical Review Letters* 13 (1964), pp. 508–509. DOI: `10.1103/PhysRevLett.13.508`.

[24] P. W. Higgs. „Spontaneous Symmetry Breakdown without Massless Bosons“. *Physical Review* 145 (1964), pp. 1156–1163.
DOI: 10.1103/PhysRev.145.1156.

[25] F. Englert and R. Brout. „Broken Symmetry and the Mass of Gauge Vector Mesons“. *Physical Review Letters* 13 (1964), pp. 321–323.
DOI: 10.1103/PhysRevLett.13.321.

[26] G. S. Guralnik, C. R. Hagen, and T. W. B. Kibble. „Global Conservation Laws and Massless Particles“. *Physical Review Letters* 13 (1964), pp. 585–587.
DOI: 10.1103/PhysRevLett.13.585.

[27] R. K. Ellis, W. J. Stirling, and B. R. Webber. „QCD and Collider Physics“. Cambridge Monographs on Particle Physics, Nuclear Physics and Cosmology. Cambridge University Press, 1996.

[28] M. E. Peskin and D. V. Schroeder. „An Introduction to Quantum Field Theory“. 1st. Reading, MA: Addison-Wesley, 1995. ISBN: 978-0201503975.

[29] R. P. Feynman and M. Gell-Mann. „Theory of the Fermi Interaction“. *Phys. Rev.* 109 (1958), pp. 193–198.
DOI: 10.1103/PhysRev.109.193.

[30] R. E. Marshak and E. C. G. Sudarshan. „Note on the Nature of the Four-Fermion Interaction“. *Phys. Rev.* 109 (1958), pp. 1860–1862.
DOI: 10.1103/PhysRev.109.1860.

[31] C. S. Wu et al. „Experimental Test of Parity Conservation in Beta Decay“. *Phys. Rev.* 105 (1957), pp. 1413–1415.
DOI: 10.1103/PhysRev.105.1413.

[32] S. L. Glashow. „Partial Symmetries of Weak Interactions“. *Nuclear Physics* 22 (1961), pp. 579–588.
DOI: 10.1016/0029-5582(61)90469-2.

[33] S. Weinberg. „A Model of Leptons“. *Physical Review Letters* 19 (1967), pp. 1264–1266.
DOI: 10.1103/PhysRevLett.19.1264.

[34] A. Salam. „Weak and Electromagnetic Interactions“. *Elementary Particle Theory: Proceedings of the Eighth Nobel Symposium.* Ed. by N. Svartholm. Stockholm: Almquist & Wiksell, 1968, pp. 367–377.

[35] J. M. Butterworth et al. „Jet Substructure as a New Higgs-Search Channel at the Large Hadron Collider“. *Physical Review Letters* 100.24 (June 2008). ISSN: 1079-7114.
DOI: 10.1103/physrevlett.100.242001.

[36] CMS Collaboration. *Collaboration.* CERN website. Online; accessed 16 September 2025.

[37] A. Tumasyan et al. „Measurement of the top quark mass using a profile likelihood approach with the lepton + jets final states in proton-proton collisions at

$$\sqrt{s} = 13\,\mathrm{TeV}$$

“. *The European Physical Journal C* 83.10 (Oct. 2023). ISSN: 1434-6052. DOI: 10.1140/epjc/s10052-023-12050-4.

[38] U. Sarkar. „Searches for supersymmetry in CMS“. *International Journal of Modern Physics A* 37.33 (Mar. 2022). ISSN: 1793-656X. DOI: 10.1142/s0217751x22400048.

[39] R. Assmann, M. Lamont, and S. Myers. „A brief history of the LEP collider“. *Nucl. Phys. B Proc. Suppl.* 109 (2002). Ed. by F. L. Navarria, M. Paganoni, and P. G. Pelfer, pp. 17–31. DOI: 10.1016/S0920-5632(02)90005-8.

[40] K. Bernhard-Novotny. *First Run 3 physics result by CMS*. https://home.web.cern.ch/news/news/physics/first-run-3-physics-result-cms. Accessed: 2025-09-15. Nov. 2022.

[41] The ATLAS Collaboration. „The ATLAS Experiment at the CERN Large Hadron Collider“. *Journal of Instrumentation* 3.08 (Aug. 2008), S08003. DOI: 10.1088/1748-0221/3/08/S08003.

[42] The LHCb Collaboration. „The LHCb Detector at the LHC“. *Journal of Instrumentation* 3.08 (Aug. 2008), S08005. DOI: 10.1088/1748-0221/3/08/S08005.

[43] The ALICE Collaboration. „The ALICE experiment at the CERN LHC“. *Journal of Instrumentation* 3.08 (Aug. 2008), S08002. DOI: 10.1088/1748-0221/3/08/S08002.

[44] R. Steerenberg. *Accelerator Report: LHC Run 3 achieves record-breaking integrated luminosity*. CERN News, Accelerators section. Published online 4 September 2024. Sept. 2024.

[45] D. Fournier and T. Virdee. „The ATLAS and CMS detectors at the LHC“. en. *Comptes Rendus. Physique* 16.4 (2015), pp. 356–367. DOI: 10.1016/j.crhy.2015.03.018.

[46] M. Ressegotti and O. behalf of the CMS Collaboration. „Overview of the CMS Detector Performance at LHC Run 2“. *Universe* 5.1 (2019). ISSN: 2218-1997. DOI: 10.3390/universe5010018.

[47] CMS Open Data Workshop 2024. *Transverse slice of the CMS detector*. Accessed: 2025-09-01. 2024.

[48] W. Adam et al. „The CMS Phase-1 pixel detector upgrade“. *Journal of Instrumentation* 16.02 (Feb. 2021), P02027. DOI: 10.1088/1748-0221/16/02/P02027.

[49] W. Elmetenawee. „CMS track reconstruction performance during Run 2 and developments for Run 3“. *Proceedings of 40th International Conference on High Energy physics — PoS(ICHEP2020)*. Vol. 390. 2021, p. 733.
DOI: `10.22323/1.390.0733`.

[50] R. Frühwirth. „Application of Kalman filtering to track and vertex fitting“. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 262.2 (1987), pp. 444–450. ISSN: 0168-9002.
DOI: `https://doi.org/10.1016/0168-9002(87)90887-4`.

[51] W. Adam et al. „Reconstruction of electrons with the Gaussian-sum filter in the CMS tracker at the LHC“. *Journal of Physics G: Nuclear and Particle Physics* 31.9 (July 2005), N9.
DOI: `10.1088/0954-3899/31/9/N01`.

[52] A. Sirunyan et al. „Performance of the CMS muon detector and muon reconstruction with proton-proton collisions at $\sqrt{s}$=13 TeV“. *Journal of Instrumentation* 13.06 (June 2018), P06015–P06015. ISSN: 1748-0221.
DOI: `10.1088/1748-0221/13/06/p06015`.

[53] M. Cacciari, G. P. Salam, and G. Soyez. „The anti-kt jet clustering algorithm“. *Journal of High Energy Physics* 2008.04 (Apr. 2008), p. 063.
DOI: `10.1088/1126-6708/2008/04/063`.

[54] CMS Collaboration. „Performance of the DeepTau algorithm for the discrimination of taus against jets, electron, and muons“ (2019).

[55] O. Filatov. „Tau identification algorithms and study of the CP structure of the Yukawa coupling between the Higgs boson and tau leptons in CMS“. Dissertation, Universität Hamburg, 2023. Dissertation. Hamburg: Universität Hamburg, 2023, p. 191.
DOI: `10.3204/PUBDB-2023-01292`.

[56] L. Russell. *Identification of Hadronic Tau Lepton Decays with Domain Adaptation using Adversarial Machine Learning at CMS*. Oct. 2022.

[57] F. J. Massey. „The Kolmogorov-Smirnov Test for Goodness of Fit“. *Journal of the American Statistical Association* 46.253 (1951), pp. 68–78. ISSN: 01621459, 1537274X.

[58] CMS Collaboration. „Identification of hadronic tau lepton decays using a deep neural network“. *Journal of Instrumentation* 17.07 (July 2022), P07023.
DOI: `10.1088/1748-0221/17/07/P07023`.

[59] CMS Tau POG. *TauMLTools: Tools for Machine Learning for Tau reco & Id.* `https://github.com/cms-tau-pog/TauMLTools/tree/c0fc52f`. Commit `c0fc52f`, accessed: 2025-09-09. 2019.

[60] CMS Tau Physics Object Group. *DeepTau DataLoader implementation in DataLoader_main.h*. https://github.com/cms-tau-pog/TauMLTools/blob/c0fc52f/Training/interface/DataLoader_main.h. Commit c0fc52f, accessed 2025-10-14. 2025.

[61] I. Goodfellow, Y. Bengio, and A. Courville. „Deep Learning“. http://www.deeplearningbook.org. MIT Press, 2016.

[62] T.-Y. Lin et al. *Focal Loss for Dense Object Detection*. 2018. arXiv: 1708.02002 [cs.CV].

[63] A. Vaswani et al. *Attention Is All You Need*. 2023. arXiv: 1706.03762 [cs.CL].

[64] W. Ansar, S. Goswami, and A. Chakrabarti. *A Survey on Transformers in NLP with Focus on Efficiency*. 2024. arXiv: 2406.16893 [cs.CL].

[65] H. Qu, C. Li, and S. Qian. *Particle Transformer for Jet Tagging*. 2024. arXiv: 2202.03772 [hep-ph].

[66] Raikwar, Piyush et al. „Transformers for Generalized Fast Shower Simulation“. *EPJ Web of Conf.* 295 (2024), p. 09039.
DOI: 10.1051/epjconf/202429509039.

[67] S. V. Stroud et al. *Transformers for Charged Particle Track Reconstruction in High Energy Physics*. 2024. arXiv: 2411.07149 [hep-ex].

[68] J. A. Hanley and B. J. McNeil. „The meaning and use of the area under a receiver operating characteristic (ROC) curve.“ *Radiology* 143.1 (1982). PMID: 7063747, pp. 29–36.
DOI: 10.1148/radiology.143.1.7063747. eprint: https://doi.org/10.1148/radiology.143.1.7063747.

[69] CMS Collaboration. *Tau identification recommendations for Run 3*. https://twiki.cern.ch/twiki/bin/view/CMS/TauIDRecommendationForRun3. Accessed: 2025-09-15. Sept. 2025.

[70] T. Voigtlaender. *Thesis TauMLTools fork*. https://github.com/tvoigtlaender/TauMLTools/commit/b3f3377c. Commit b3f3377, 03 Nov 2025. 2025.

[71] M. Abadi et al. „TensorFlow: A system for large-scale machine learning“. *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*. 2016, pp. 265–283.

[72] T. Developers. *TensorFlow*. Version v2.10.0. Sept. 2022.
DOI: 10.5281/zenodo.7604243.

[73] H. Team. *HTCondor*. Version 24.2.1. Nov. 2024.
DOI: 10.5281/zenodo.14238998.

[74] D. P. Kingma and J. Ba. *Adam: A Method for Stochastic Optimization*. 2017. arXiv: 1412.6980 [cs.LG].

[75] N. Srivastava et al. „Dropout: A Simple Way to Prevent Neural Networks from Overfitting“. *J. Machine Learning Res.* 15 (2014), pp. 1929–1958.

[76] K. He et al. *Deep Residual Learning for Image Recognition.* 2015. arXiv: 1512.03385 [cs.CV].

[77] L. Prechelt. „Early Stopping - But When?“ *Neural Networks: Tricks of the Trade.* Ed. by G. B. Orr and K.-R. Müller. Berlin, Heidelberg: Springer Berlin Heidelberg, 1998, pp. 55–69. ISBN: 978-3-540-49430-0.
DOI: 10.1007/3-540-49430-8_3.

[78] O. Filatov. *Personal correspondence regarding his thesis work.* Correspondence with the author. 2023.

[79] V. Satopaa et al. „Finding a Kneedle in a Haystack: Detecting Knee Points in System Behavior“. July 2011, pp. 166–171.
DOI: 10.1109/ICDCSW.2011.20.

[80] N. C. Frey et al. „Neural scaling of deep chemical models“. *Nature Machine Intelligence* 5.11 (2023), pp. 1297–1305. ISSN: 2522-5839.
DOI: 10.1038/s42256-023-00740-3.

[81] J. Kaplan et al. *Scaling Laws for Neural Language Models.* 2020. arXiv: 2001.08361 [cs.LG].

[82] I. Alabdulmohsin, B. Neyshabur, and X. Zhai. *Revisiting Neural Scaling Laws in Language and Vision.* 2022. arXiv: 2209.06640 [cs.LG].

[83] J. Hestness et al. *Deep Learning Scaling is Predictable, Empirically.* 2017. arXiv: 1712.00409 [cs.LG].

[84] J. S. Rosenfeld et al. *A Constructive Prediction of the Generalization Error Across Scales.* 2019. arXiv: 1909.12673 [cs.LG].

[85] C. F. Gauss. „Relationes inter locos plures in orbita“. *Theoria Motus Corporum Coelestium in Sectionibus Conicis Solem Ambientium.* Cambridge Library Collection - Mathematics. Cambridge University Press, 2011, pp. 82–124.

[86] S. Amari et al. „Asymptotic statistical theory of overtraining and cross-validation“. *IEEE Transactions on Neural Networks* 8.5 (1997), pp. 985–996.
DOI: 10.1109/72.623200.

[87] CMS Collaboration. *b-hive: Configuration Files config/*.yml*. https://gitlab.cern.ch/cms-btv/b-hive/-/tree/be02210d/config/. Accessed: 2025-05-12. 2025.

[88] Y. Rao et al. *DynamicViT: Efficient Vision Transformers with Dynamic Token Sparsification.* 2021. arXiv: 2106.02034 [cs.CV].

[89] G. Aad et al. „Measurement of angular and momentum distributions of charged particles within and around jets in Pb + Pb and pp collisions at $\sqrt{s_{NN}} = 5.02$ TeV with the ATLAS detector“. *Physical Review C* 100.6 (Dec. 2019). ISSN: 2469-9993.
DOI: 10.1103/physrevc.100.064901.

[90] CERN Monitoring Team. „HS06 Report“. Retrieved 2025-07-04. 2025.
URL: https://monit-grafana.cern.ch/d/C8ewaCrWk/hs06-report.

[91] TOP500 Project. *Top500 List - June 2025.* Accessed: 2025-07-28. June 2025.

[92] TOP500 Project. *Green500 List - June 2025.* Accessed: 2025-07-28. June 2025.

[93] D. Giordano et al. „HEPiX Benchmarking Solution for WLCG Computing Resources“. *Computing and Software for Big Science* 5.1 (Dec. 2021), p. 28. ISSN: 2510-2044.
DOI: 10.1007/s41781-021-00074-y.

[94] C. H.-B. Group. *HEP Benchmark Suite.* Accessed: 2025-07-04. 2019.

[95] D. Giordano et al. *HEPScore: A new CPU benchmark for the WLCG.* 2023. arXiv: 2306.08118 [hep-ex].

[96] HEPiX Benchmarking Working Group. *HEP-SPEC06 (HS06) - Legacy Benchmarking Page.* https://w3.hepix.org/benchmarking/HS06.html. HS06 is the HEP-wide benchmark for CPU performance, based on the all_cpp subset of SPEC CPU 2006; official WLCG metric since 1 April 2009. July 2025.

[97] R. Duncan. „ A survey of parallel computer architectures “. *Computer* 23.02 (Feb. 1990), pp. 5–16. ISSN: 1558-0814.
DOI: 10.1109/2.44900.

[98] J. Nickolls et al. „Scalable parallel programming with CUDA“. *ACM SIGGRAPH 2008 Classes.* SIGGRAPH '08. Los Angeles, California: Association for Computing Machinery, 2008. ISBN: 9781450378451.
DOI: 10.1145/1401132.1401152.

[99] NVIDIA Corporation. „NVIDIA A100 Tensor Core GPU Architecture“. Tech. rep. Accessed: 2025-04-28. NVIDIA Corporation, May 2020.

[100] S. Markidis et al. „NVIDIA Tensor Core Programmability, Performance & Precision“. *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* IEEE, May 2018, pp. 522–531.
DOI: 10.1109/ipdpsw.2018.00091.

[101] A. Mishra et al. *Accelerating Sparse Deep Neural Networks.* 2021. arXiv: 2104.08378 [cs.LG].

[102] I. Bird. „Computing for the Large Hadron Collider“. *Annual Review of Nuclear and Particle Science* 61.1 (2011), pp. 99–118.
DOI: 10.1146/annurev-nucl-102010-130059. eprint: https://doi.org/10.1146/annurev-nucl-102010-130059.

[103] E. Zenker et al. „Alpaka – An Abstraction Library for Parallel Kernel Acceleration“. *2016 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW).* IEEE, May 2016.
DOI: 10.1109/ipdpsw.2016.50.

[104] M. Atif et al. *Evaluating Portable Parallelization Strategies for Heterogeneous Architectures in High Energy Physics.* 2023. arXiv: 2306.15869 [hep-ex].

[105] Karlsruhe Institute of Technology (KIT). *Supercomputer Inaugurated at KIT: HoreKa, among the fastest and most energy-efficient supercomputers.* Press Release PI 071/2021, KIT. HoreKa reached a peak capacity of 17 PetaFLOPS, ranking among Europe's 15 fastest and 13th most efficient globally. July 2021.

[106] O. Mock and C. J. Swift. „The Share 709 System: Programmed Input-Output Buffering". *J. ACM* 6.2 (Apr. 1959), pp. 145–151. ISSN: 0004-5411. DOI: `10.1145/320964.320970`.

[107] G. Thain. „Cgroups, containers and HTCondor, oh my". *Center for High Throughput Computing User School.* CERN Indico event 733513, contribution 3118608. Slides on HTCondor's use of Linux cgroups to enforce memory limits and OOM holds. 2019.

[108] P. Malzacher et al. „Requirements for a Regional Data and Computing Centre in Germany (RDCCG)". 2001. URL: `https://www.scc.kit.edu/downloads/SDM/GridKa/RDCCG-answer-v8.pdf` (visited on 08/08/2022).

[109] N. Corporation. „NVIDIA Tesla V100 GPU Architecture". Tech. rep. WP-08608-001_v1.1. Accessed: 2025-04-28. NVIDIA Corporation, Aug. 2017.

[110] M. Giffels. *Private communication.* Private talk in office, KIT, May 2025. 2025.

[111] CMS Collaboration. „Quantifying the computational speedup with madgraph4gpu for CMS worflow" (2024).

[112] A. Valassi et al. „Developments in Performance and Portability for MadGraph5_-aMC@NLO". *PoS* ICHEP2022 (2022), p. 212. DOI: `10.22323/1.414.0212`.

[113] HEP-Benchmarks Developers. *HEP-Benchmarks: High Energy Physics Benchmarking Suite.* `https://gitlab.cern.ch/hep-benchmarks`. Accessed: 2025-05-12. 2025.

[114] Michael Kerrisk and the man-pages project. *date(1): Linux manual page.* Accessed via man7.org. man-pages project / man7.org. July 2025.

[115] D. Laurie and contributors. *IPMItool: Utility for Managing IPMI-Enabled Devices.* `https://github.com/ipmitool/ipmitool`. Accessed: 2025-05-12. 2025.

[116] R. Kavanagh, D. Armstrong, and K. Djemame. „Accuracy of Energy Model Calibration with IPMI". *2016 IEEE 9th International Conference on Cloud Computing (CLOUD).* 2016, pp. 648–655. DOI: `10.1109/CLOUD.2016.0091`.

[117] *GOCDB — Site Details GridKa.* `https://goc.egi.eu/portal/index.php?Page_Type=Site&id=116`. Accessed: 2025-08-11. 2025.

[118] NVIDIA Corporation. *NVIDIA Data Center GPU Manager (DCGM).* `https://developer.nvidia.com/dcgm`. Accessed: 2025-05-12. 2025.

[119] NVIDIA Corporation. *NVIDIA System Management Interface (nvidia-smi).* https://developer.nvidia.com/system-management-interface. Accessed: 2025-05-12. 2025.

[120] Z. Yang, K. Adamek, and W. Armour. „Accurate and Convenient Energy Measurements for GPUs: A Detailed Study of NVIDIA GPU's Built-In Power Sensor". *SC24: International Conference for High Performance Computing, Networking, Storage and Analysis.* IEEE, Nov. 2024, pp. 1–17.
DOI: 10.1109/sc41406.2024.00028.

[121] Michael Kerrisk and the man-pages project. *top(1): Linux manual page.* Accessed via man7.org. man-pages project / man7.org. July 2025.

[122] U. Hölzle and L. A. Barroso. „ The Case for Energy-Proportional Computing ". *Computer* 40.12 (Dec. 2007), pp. 33–37. ISSN: 1558-0814.
DOI: 10.1109/MC.2007.443.

[123] J. A. Martinez et al. *Pileup mitigation at the Large Hadron Collider with Graph Neural Networks.* 2019. arXiv: 1810.07988 [hep-ph].

[124] CMS Collaboration. *A DNN for CMS track classification and selection.* 2023. arXiv: 2311.05157 [hep-ex].

[125] A. A. Pol et al. *Detector monitoring with artificial neural networks at the CMS experiment at the CERN Large Hadron Collider.* 2018. arXiv: 1808.00911 [physics.data-an].

[126] CMS Collaboration. *Development of systematic uncertainty-aware neural network trainings for binned-likelihood analyses at the LHC.* 2025. arXiv: 2502.13047 [hep-ex].

[127] HEP ML Community. *A Living Review of Machine Learning for Particle Physics.*

[128] U. Sarkar. *Run 3 performance and advances in heavy-flavor jet tagging in CMS.* 2024. arXiv: 2412.05863 [hep-ex].

[129] L. Vaughan et al. *PileUp Mitigation at the HL-LHC Using Attention for Event-Wide Context.* 2025. arXiv: 2503.02860 [hep-ex].

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

[131] K. Sievers, A. Zakrzewski, and util-linux contributors. *taskset(1) — retrieve or set a process's CPU affinity.* Accessed: 2025-06-16. The Linux Foundation. 2024.

[132] S. Brommer. „A data-driven method for Higgs boson analyses in di-τ final states for the LHC Run II and beyond". PhD thesis. Karlsruher Institut für Technologie (KIT), 2023. 161 pp.
DOI: 10.5445/IR/1000155107.

[133] M. Stone. „Cross-Validatory Choice and Assessment of Statistical Predictions“. *Journal of the Royal Statistical Society: Series B (Methodological)* 36.2 (Dec. 2018), pp. 111–133. ISSN: 0035-9246.
DOI: 10.1111/j.2517-6161.1974.tb00994.x. eprint: https://academic.oup.com/jrsssb/article-pdf/36/2/111/49096683/jrsssb_36_2_111.pdf.

[134] G. M. Amdahl. „Validity of the single processor approach to achieving large scale computing capabilities“. *AFIPS Conference Proceedings*. Vol. 30. ACM. 1967, pp. 483–485.

[135] NVIDIA. *Multi-Process Service (MPS)*. Accessed: 2025-06-16. NVIDIA Corporation.

[136] NVIDIA. *NVIDIA A100 MIG User Guide*. Accessed: 2025-06-16. NVIDIA Corporation.

[137] J. Bechtel. „A novel search for di-Higgs events in the $\tau^-\tau^+ + \mathrm{b}\bar{\mathrm{b}}$ final state in pp collisions at 13 TeV at the LHC“. PhD thesis. Karlsruher Institut für Technologie (KIT), 2021. 158 pp.
DOI: 10.5445/IR/1000130103.

[138] J. D. C. Little. „A proof for the queuing formula: $\mathrm{L} = \lambda\mathrm{W}$“. *Operations Research* 9.3 (1961), pp. 383–387.
DOI: 10.1287/opre.9.3.383.

[139] T. Developers. *TensorFlow*. Version v2.8.2. May 2022.
DOI: 10.5281/zenodo.6574269.

[140] M. Kerrisk. *Control Groups (cgroups)*. Accessed: 2025-06-16. The Linux Foundation. 2024.

[141] HEPiX Benchmarking Working Group. *HEP-SPEC06 (HS06)*. https://w3.hepix.org/benchmarking/HS06.html. Accessed: 2025-08-11. n.d.

[142] J. Alwall et al. „The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations“. *Journal of High Energy Physics* 2014.7 (July 2014). ISSN: 1029-8479.
DOI: 10.1007/jhep07(2014)079.

[143] H.-B. Team. *DESCRIPTION file for mg5amc-madgraph4gpu-2022 benchmark*. https://gitlab.cern.ch/hep-benchmarks/hep-workloads/-/blob/master/mg5amc/madgraph4gpu-2022/mg5amc-madgraph4gpu-2022/DESCRIPTION. Accessed on 2025-07-06. 2022.

[144] madgraph5/madgraph4gpu contributors. *Madgraph4gpu commit fcd4130*. https://github.com/madgraph5/madgraph4gpu/commit/fcd4130. Accessed 2025-07-04. July 2025.

[145] madgraph5/madgraph4gpu contributors. *Benchmark CUDA code commit fcd4130*. https://github.com/madgraph5/madgraph4gpu/blob/fcd4130/epochX/cudacpp/gg_ttgg.sa/SubProcesses/P1_Sigma_sm_gg_ttxgg/check_sa.cc. Accessed 2025-07-06. July 2025.

[146] T. Voigtländer et al. „Optimized GPU usage in high energy physics applications“. *Journal of Physics: Conference Series* (2025). Accepted for publication.

[147] T. Voigtländer et al. *Pinpoint resource allocation for GPU batch applications.* Accepted for publication in *Journal of Physics: Conference Series (ACAT 2024).* 2025. arXiv: `2505.08562 [hep-ex]`.

[148] J. Alwall et al. „The automated computation of tree-level and next-to-leading order differential cross sections, and their matching to parton shower simulations“. *JHEP* 07 (2014), p. 079.
DOI: `10.1007/JHEP07(2014)079`. arXiv: `1405.0301 [hep-ph]`.

[149] J. Alwall et al. „Comparative study of various algorithms for the merging of parton showers and matrix elements in hadronic collisions“. *Eur. Phys. J. C* 53 (2008), pp. 473–500.
DOI: `10.1140/epjc/s10052-007-0490-5`. arXiv: `0706.2569 [hep-ph]`.

[150] P. Nason. „A New method for combining NLO QCD with shower Monte Carlo algorithms“. *JHEP* 11 (2004), p. 040.
DOI: `10.1088/1126-6708/2004/11/040`. arXiv: `hep-ph/0409146`.

[151] S. Frixione, P. Nason, and C. Oleari. „Matching NLO QCD computations with parton shower simulations: the POWHEG method“. *JHEP* 11 (2007), p. 070.
DOI: `10.1088/1126-6708/2007/11/070`. arXiv: `0709.2092 [hep-ph]`.

[152] S. Alioli et al. „A general framework for implementing NLO calculations in shower Monte Carlo programs: the POWHEG BOX“. *JHEP* 06 (2010), p. 043.
DOI: `10.1007/JHEP06(2010)043`. arXiv: `1002.2581 [hep-ph]`.

[153] S. Frixione, P. Nason, and G. Ridolfi. „A Positive-weight next-to-leading-order Monte Carlo for heavy flavour hadroproduction“. *JHEP* 09 (2007), p. 126.
DOI: `10.1088/1126-6708/2007/09/126`. arXiv: `0707.3088 [hep-ph]`.

[154] J. M. Campbell et al. „Top-pair production and decay at NLO matched with parton showers“. *JHEP* 04 (2015), p. 114.
DOI: `10.1007/JHEP04(2015)114`. arXiv: `1412.1828 [hep-ph]`.

[155] „An introduction to PYTHIA 8.2“. *Computer Physics Communications* 191 (2015), pp. 159–177. ISSN: 0010-4655.
DOI: `https://doi.org/10.1016/j.cpc.2015.01.024`.

[156] N. Davidson et al. „Universal interface of TAUOLA: Technical and physics documentation“. *Computer Physics Communications* 183.3 (2012), pp. 821–843. ISSN: 0010-4655.
DOI: `https://doi.org/10.1016/j.cpc.2011.12.009`.

[157] CMS Collaboration. „Event generator tunes obtained from underlying event and multiparton scattering measurements“. *Eur. Phys. J. C* 76.3 (2016), p. 155.
DOI: `10.1140/epjc/s10052-016-3988-x`. arXiv: `1512.00815 [hep-ex]`.

[158] S. Agostinelli et al. „Geant4—a simulation toolkit“. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment* 506.3 (2003), pp. 250–303. ISSN: 0168-9002. DOI: `https://doi.org/10.1016/S0168-9002(03)01368-8`.

[159] D. Hendrycks and K. Gimpel. *Gaussian Error Linear Units (GELUs).* 2023. arXiv: `1606.08415 [cs.LG]`.

[160] X. Glorot and Y. Bengio. „Understanding the difficulty of training deep feedforward neural networks“. *Proceedings of the Thirteenth International Conference on Artificial Intelligence and Statistics.* Ed. by Y. W. Teh and M. Titterington. Vol. 9. Proceedings of Machine Learning Research. Chia Laguna Resort, Sardinia, Italy: PMLR, 13–15 May 2010, pp. 249–256.

[161] NVIDIA. „DCGM API Field Identifiers“. 2025. URL: `https://docs.nvidia.com/datacenter/dcgm/latest/dcgm-api/dcgm-api-field-ids.html`.

Danksagung

Zu guter Letzt bleibt für mich nur noch ein sehr großes Danke an alle, die mich auf dem Weg bis hierher unterstützt haben.

Mein größter Dank geht an meinen Referenten und Supervisor Günter Quast, der an so manchem Tag wohl mehr an mich geglaubt hat als ich selbst. Ich denke nicht, dass ich jemanden anderen kenne, der mit solcher Freude seine Finger in alles steckt, was auch nur am Rande mit Datenerfassung zu tun hat. Dein Enthusiasmus hat mich immer und immer wieder angesteckt und deine Tür war immer offen, auch wenn du noch so oft von der Bürokratie belagert wurdest. Ich hoffe zutiefst, dass ich ein würdiger Abschluss in deiner langen Laufbahn als Verteidiger der Computer im Herzen der Physik sein konnte.

Ich möchte ebenso meinem Korreferenten Achim Streit danken, der mich vor nun fünf Jahren mit einer so einfachen Frage von der Festkörperphysik in die Teilchenphysik umgestimmt hat. Ja, es stellt sich heraus, dass es noch einiges zu tun gibt, bis die Gesamtheit der Teilchenphysik mehr als nur einen CPU-Thread verwendet.

Des Weiteren möchte ich Roger Wolf danken, bei dem ich gegen Ende hin mit zunehmender Frequenz in der Tür stand, um nach Rat zu fragen. Ohne dich würde ich wohl immer noch mit mir selbst debattieren, was andere vor mir gemacht haben, statt meine eigenen Ideen einzubringen.

Ein großes Danke geht an die Mitglieder der Computing-Gruppe und allen voran Matthias Schnepf, der so unendlich viele Ideen, Diskussionen und Fragen über sich ergehen lassen musste. Unter euch hatte ich das Gefühl, dass Effizienz mehr ist als nur ein Buzzword.

Ein besonderes Danke an Arthur Monsch, dem ich noch einen Kasten für zahllose Diskussionen über visuell ansprechende Plots schulde. Auch den weiteren Mitgliedern des ETP, die die Zeit hier spaßig gestaltet haben, möchte ich danken: Aritra, Cedric, Christian, Jan, Johannes, Lars, Max, Michael, Nikita, Ralf, Robin und viele mehr...

Außerhalb des Universitätstrubels möchte ich Raphael Merz und dem SCO danken. Unabhängig von allem anderen waren die Vereinsabende seit über 20 Jahren ein Fixpunkt meines Lebens. Danke auch, dass du eingesprungen bist, wenn es für mich zwischen Promotion und Verein doch mal zu viel war.

Als Abschluss möchte ich meinen Eltern Kirsten und Helge, so wie meiner Schwester Anika danken, die von allen am besten wissen wie lang und schwer dieser Weg für mich war. Ohne euch hätte es an allen Ecken und Enden an Motivation gefehlt.

Erklärung der selbständigen Anfertigung der Dissertationsschrift

Hiermit erkläre ich, Tim Voigtländer, dass ich die Dissertationsschrift mit dem Titel

»*Efficient Integration of GPUs in Particle Physics and Transformer-Based Tau Lepton Identification*«

selbständig und unter ausschließlicher Verwendung der angegebenen Hilfsmittel angefertigt habe.

Tim Voigtländer
Karlsruhe, den 06. March 2026