Quantum–inspired Machine Learning for High–Energy Physics Applications

EMA PULJAK

Quantum-inspired Machine Learning for High-Energy Physics Applications

EMA PULJAK

Submitted in total fulfilment of the requirements of the degree of

Doctor of Philosophy

DEPARTMENT OF PHYSICS
UNIVERSITAT AUTÒNOMA DE BARCELONA

October 2025

ORCID: 0000-0002-6011-9965

To my whole family,
for being a constant source of optimism,
for igniting my love of research,
and for teaching me never to give up.

Quantum-inspired Machine Learning for High-Energy Physics Applications

EMA PULJAK
Principal Supervisor: Artur Garcia Saez
CERN Supervisor: Maurizio Pierini
Tutor: John Calsamiglia Costa

Abstract

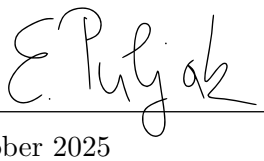
This thesis explores the intersection of quantum computing, quantum-inspired approaches and machine learning (ML), focusing on applications in high-energy physics (HEP). As ML becomes embedded in scientific workflows, both quantum and quantum-inspired methods are being investigated for their potential to tackle complex, data-driven tasks. However, there exists a significant gap between theoretical advances and their practical deployment in domains where data is high-dimensional and structured, such as HEP, specifically under the implementation constraints present at the Large Hadron Collider (LHC) at CERN. Driven by relevant tasks in physics applications at the LHC, this work presents several contributions across anomaly detection and classification tasks, particularly in the context of jet substructure analysis. The exploration of hybrid quantum-classical approaches demonstrates the practical utility of quantum components integrated into classical pipelines. However, the current limitations of quantum hardware have inspired the investigation of practical usage of Tensor Networks (TN), which model quantum-like structures while being executable on classical hardware, and are thus suitable for LHC applications.

A central effort of this thesis is the development of the open-source library `tn4ml`, designed for training, customization and benchmarking of TN-based models in ML pipelines. This library supports the development of real-life physics applications, including: a probabilistic TN-based model using Matrix Product States for detecting anomalies in the latent space of LHC events; and a TN classification model for jet tagging with potential deployment in the real-time trigger system at the LHC. On top of this, a hybrid quantum-classical method is studied for anomaly detection in proton-proton collision events. These applications are developed and evaluated through systematic performance analysis, including problem-specific metrics, such as execution time and model complexity. Our results suggest that hybrid and TN-based models can offer competitive and scalable solutions in HEP-specific and applied ML settings. From a broader perspective, this work contributes to bridging a gap between theoretical quantum and quantum-inspired methods and their practical implementation by showcasing domain-specific models, providing tools for their adaptation in ML workflows and evaluating their utility in realistic scenarios.

Declaration

I declare that this thesis and the work presented in it are my own. I confirm that:

- the thesis comprises only my original work towards the Doctor of Philosophy, except where indicated in the preface;
- due acknowledgement has been made in the text to all other material used; and
- the thesis is less than 100,000 words in length, exclusive of tables, maps, bibliographies and appendices.



EMA PULJAK, October 2025

Preface

This thesis research was carried out at the European Organization for Nuclear Research (CERN) and at the Department of Physics, Universitat Autònoma de Barcelona, in collaboration with the Barcelona Supercomputing Center (BSC). The main contributions of this thesis are presented in Chapters 6, 8, 9, 10 and are based on the following publications:

- E. Puljak, V. Belis, K. A. Woźniak, *et al.* **Quantum anomaly detection in the latent space of proton collision events at the LHC**, *Nature Communications Physics*, 2024 [1]. This study is described in Chapter 8.
- E. Puljak, *et al.*, `tn4ml`**: Tensor Network Training and Customization for Machine Learning**, submitted to *Quantum Journal*. [2]. This study is detailed in Chapter 6.
- E. Puljak, *et al.*, **Tensor Network for Anomaly Detection in the Latent Space of Proton Collision Events at the LHC**, *Machine Learning: Science and Technology*, 2025 [3]. Chapter 9 describes this study.
- A manuscript in preparation, where the important part of it is based on the study presented in Chapter 10. This study focuses on developing a **Tensor Network model for jet classification as a realistic model for future deployment in the LHC trigger system**. It is being prepared as a collaborative effort between CERN, Universitat Autònoma de Barcelona, and the University of Padova.

Funding

This work was supported by the *Quantum Technology Initiative* at CERN.

Acknowledgements

I thank my supervisors, Artur and Maurizio, for believing in my project ideas, for their constant support and guidance, and for meeting with me very frequently (important to note!). I am grateful for the freedom they gave me and for their words of wisdom. Thank you, Artur, for the opportunity to be a part of the BSC group, for always making me feel welcome in Barcelona, and for always reminding me to take care of my mental health throughout my PhD. Thank you, Maurizio, for inspiring me to give my best even when I doubted myself, for teaching me valuable things on so many levels, for always having my back at CERN and for showing me what science should be. Special thanks to my aunt Mirjana and uncle Jordi, for giving me a second home in Barcelona during this journey, and for always being there for me. I thank my dad for igniting my passion for research, for planting a seed of love for CERN, and for giving me advice along the way. To my mum, for always letting me be myself, accepting whoever I wanted to be, and inspiring me to believe in myself. To both my parents, brother and sister - thank you for being there for me and for supporting me throughout this journey. A special thanks to my brother for trying to understand quantum computing just to be able to help me in the early stages of this PhD. To my partner Aljoša, thank you for being there for me during the most difficult months of this journey, and for understanding my mental state. To all my friends and family who have made my life full of love throughout these four years and supported me in every way they could - thank you. I thank my angel, Sergio Sanchez Ramirez, for being my constant support, day and night, both professionally and personally, thank you for inspiring me to be more confident in my work. Thank you, Sergi Masot, for thoroughly helping me review papers, for always being present when I needed help, and for surviving seven days in Italy. To Vasilis Belis, thank you for being a good friend, an even better colleague, an inspiration to look up to, and a great support throughout all the research years. Thank you, Jofre V. Muns, for being very helpful and engaging when we began developing `tn4ml`. I'm grateful to Michele Grossi and Sofia Vallecorsa for welcoming me into the Quantum Technology Initiative group at CERN and for giving me a chance to lecture at the CERN Summer School. Thank you to Miguel A. González Ballester for his support and constant availability to participate in the medical project. I thank Gabriele D'Angeli for his contribution to the development of `tn4ml`. To the entire BSC group - thank you for making me feel like I belong and showing me what a healthy research environment looks like. To the quantum group at CERN, for creating such a comforting and supportive workplace. To all my fellow colleagues, many of whom became dear friends, thank you for showing me what research is truly about: a collaborative, optimistic, and healthy environment that we should all strive to preserve!

Thank you all from the bottom of my heart!

Contents

Chapter 1

Introduction

The main motivation behind this thesis comes from the growing integration of machine learning (ML) into fundamental scientific domains and the increasing interest in its synergy with quantum computing. As ML becomes an essential component of real-life scientific workflows, both pure quantum algorithms and quantum-inspired techniques are being actively explored for their potential to enhance performance in complex data settings. However, although there has been a large development in theoretical studies of quantum and quantum-inspired algorithms, there remains a significant gap between foundational research and practical application. This thesis aims to bridge that gap and show how, with a proper motivation and a deep understanding of the domain, one can design the architectures and workflows that lead to impactful and deployable outcomes.

The focus of this work is on high-energy physics (HEP) applications, a domain characterized by high-dimensional, structured data, real-time constraints, and the need for high-performance, interoperable models. These challenges make HEP a great candidate for investigating the practical value of quantum and quantum-inspired advanced ML approaches. At the Large Hadron Collider (LHC), ML techniques are widely used for a large number of studies, particularly in searches for physics beyond the Standard Model (BSM) of particle physics. Traditional supervised ML methods often rely on model-specific assumptions, limiting the sensitivity to unknown phenomena. This limitation has motivated the use of anomaly detection techniques to discover BSM events in a model-independent and unsupervised manner. In this context, the high dimensionality and complexity of the LHC data further motivate dimensionality reduction and latent space analysis, with

the hypothesis of discovering hidden patterns in the data. One key question this thesis investigates is when and where quantum algorithms can offer an advantage in the data setting with an inherently quantum origin, and whether this results in a pure quantum architecture or still leverages classical algorithms in some places.

However, practical deployment of quantum algorithms at the LHC still remains constrained by the limitations of current quantum hardware and the engineering challenges surrounding the integration of any quantum hardware in experiments at CERN. This motivates the exploration of quantum-inspired models, specifically Tensor Networks (TNs), which preserve structural and computational advantages of quantum systems while remaining executable on classical hardware. To demonstrate their potential, this thesis evaluates the application of TNs for two important tasks in HEP: the discovery of new physics signatures and multi-classification in jet substructure analysis. Advancing ML models for the detection of anomalous, previously unseen physics scenarios can significantly increase sensitivity and improve the precision of measurements. These classification tasks are also directly relevant for the development of real-time selection algorithms in the LHC trigger systems, where the inference latency and resource utilization are tightly constrained. To support the practical implementation of TN-based, this work introduces the `tn4ml` [2], an open-source Python library that enables smooth integration of TNs into ML pipelines and facilitates experimentation, reproducibility, and potential deployment in realistic settings.

In summary, the central motivation of this thesis is to enhance the practical applicability of methods at the intersection of quantum computing and machine learning, with a focus on HEP applications. By designing models, validating them in realistic settings and developing supporting tools for their implementation, this work contributes to bridging the gap between theoretical innovations and practical utility.

1.1 Objectives of the Thesis

The primary objective of this thesis is to explore the practical applicability of quantum-inspired machine learning techniques in realistic high-energy physics scenarios. Building on theoretical foundations and implementing several methodological improvements, this

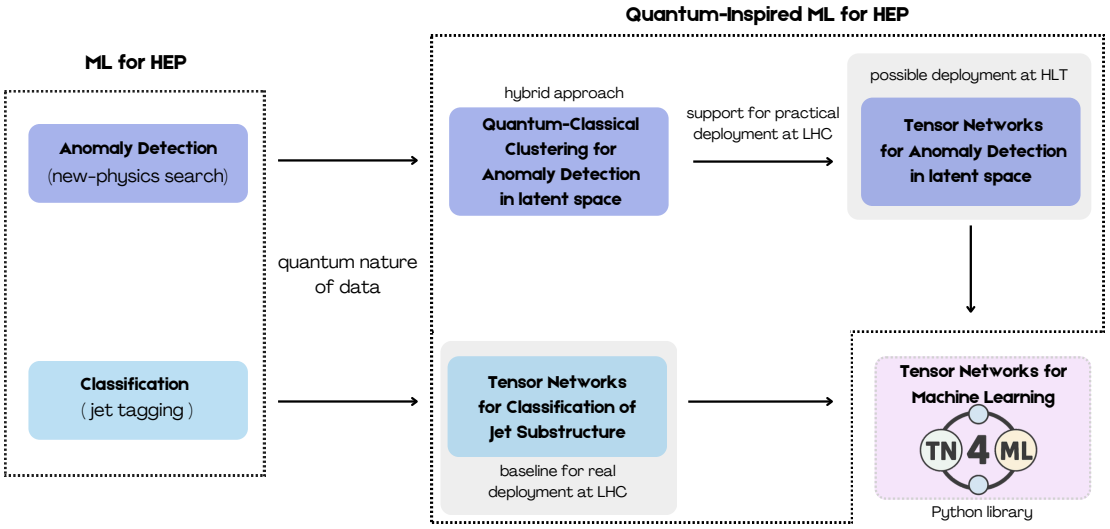


FIGURE 1.1: Overview of this research path and key topics covered in this thesis. The focus is on machine learning applications in high-energy physics (HEP), particularly anomaly detection for new physics searches and the classification of jet events (jet tagging). Given the underlying quantum nature of the data, these applications are explored with quantum / quantum-inspired algorithms. The first application is a hybrid quantum-classical clustering for anomaly detection. Due to challenges with practical implementation at the LHC, Tensor Networks are investigated as quantum-inspired models suitable for deployment on classical hardware. Both the classification task and anomaly detection are modeled with TNs (gray shaded regions). `tn4ml` Python library is developed and used for their implementation and evaluation.

work aims to demonstrate how these approaches can be developed and tested in practical settings. More precisely, the objectives of the thesis are:

- Investigate the advantage of using quantum algorithms for anomaly detection in HEP, and understand where quantum components can provide clear benefits in detection pipelines.

- To explore Tensor Networks as ML models, probing their efficiency and potential integration into low-latency, resource-constrained environments such as the LHC trigger system.

- To develop and release the `tn4ml` Python software library for flexible implementation and benchmarking of TN-based ML models, including support for data embedding, training routines, model customization, and evaluation workflows.

- To assess the practical utility of Tensor Networks in domains where the data have underlying quantum characteristics, such as collision events at the LHC.

These objectives shaped the research path of the thesis, illustrated in Fig. 1.2, together with the final definition of contributions in the latter section.

1.2 Event Selection System at CERN

At CERN, scientists are tackling challenges in high-energy physics by processing data from many experiments at the LHC. The LHC is a circular particle accelerator designed to search for new physics signatures in proton-proton (pp) and heavy-ion collisions. It contains two high-energy beams where particle bunches travel at almost the speed of light, colliding at four points, producing more than 30 petabytes of data per year. Two collision points host general-purpose detectors, the Compact Muon Solenoid (CMS)(see Appendix C) and A Toroidal LHC ApparatuS (ATLAS), with the purpose of discovering a wide range of particles and physics processes. Collisions take place at the bunch crossing frequency of 40 MHz, i.e., 40 million events per second, which makes it impossible to read everything out. Therefore, there is a two-stage event selection system, which filters important events in real time and decreases the data rates. As visualized in Fig. **??**, the first part of the system, the Level-1 Trigger (L1T), consists of custom-designed boards with field-programmable gate array (FPGA) devices, which host selection models, and reduce the event rate to $\mathcal{O}(100)$ kHz. At the second stage, in a CPU farm, is the High Level Trigger (HLT), where data rates are reduced further to $\mathcal{O}(1)$ kHz using mostly complex rule-based filter algorithms. Data at this rate can be used for offline analysis and to improve future selection algorithms. This motivated the development of algorithms and applications in this thesis, intending to advance trigger algorithms at the L1T and HLT stages.

1.3 Overview of Quantum-Inspired Machine Learning

The intersection of quantum computing and machine learning has emerged as an exciting and rapidly evolving area of research, both in theoretical and applied domains. On one hand, quantum techniques are being investigated for their potential to enhance the performance of machine learning algorithms, and on the other, machine learning is increasingly employed to optimize quantum systems and design better quantum algorithms. Within this interdisciplinary domain, two promising directions are: *hybrid* approaches, which

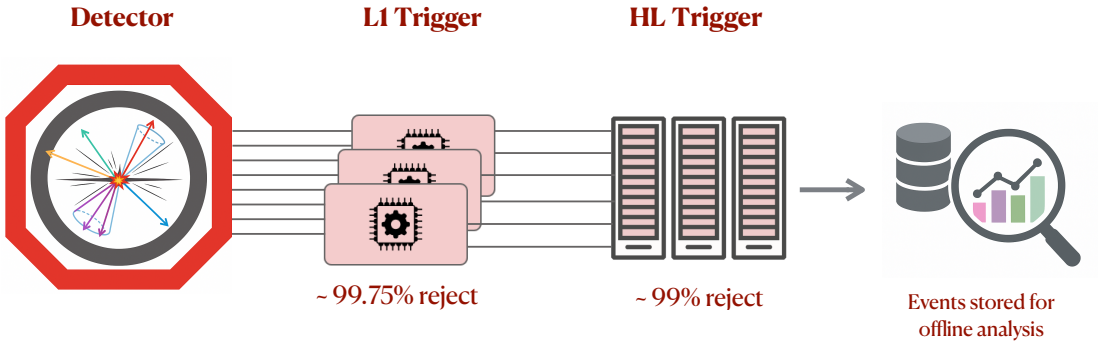


FIGURE 1.2: Two-stage trigger system at the general-purpose detectors (CMS/ATLAS). The first stage is the hardware-based Level-1 (L1) trigger, implemented on dedicated electronics and Field Programmable Gate Arrays (FPGAs, depicted as chips). The L1 system rejects about 99.75% of incoming events. The second stage is a software-based High-Level Trigger (HLT), running on a CPU farm. It performs further selection by rejecting 99% of L1-accepted events for offline storage and detailed analysis.

incorporate quantum computation to complement classical processing where considered advantageous, and *Tensor Networks*, as classical methods that use quantum principles while running on classical hardware. To bridge the two directions, it is helpful first to understand the fundamental concepts of both quantum computing and TN methods. These core concepts include:

(1) **Qubit** represents a two-dimensional quantum system and is mathematically described as a unit vector (i.e., rank-1 tensor) in a complex Hilbert space, expressed as a linear combination of computational basis states.

(2) **Entanglement** is a quantum mechanical correlation between qubits that cannot be described classically, capturing complex, non-local dependencies. In TN language, it is represented by a bond (sharing index) between tensors.

(3) **Quantum gates** are reversible (unitary) operations that act on one or more qubits, represented by rank-2 or higher-order tensors. Multiple sequences of such operations form **quantum circuits** which can often be efficiently simulated, under certain constraints, on classical hardware using Tensor Networks.

(4) **Measurement** is a probabilistic process that collapses the state into one of the basis states, as defined by the measurement operator. In the TN formalism, this corresponds to the contraction of the state TN with a tensor representing the basis vector or measurement operator, effectively computing an inner product that results in observable probabilities or expectation values.

These shared definitions provide the relationship between quantum algorithms and quantum-inspired models in their structure and behavior.

Hybrid models, as explored in this work, are motivated by practical considerations, particularly the observation that sometimes using a fully quantum algorithm can over-complicate rather than enhance the learning process. In such models, only selected stages of the pipeline employ quantum algorithms, while other parts are executed using classical methods. Given the current limitations of quantum hardware [4], most quantum ML approaches are, in fact, hybrid by design. In high-dimensional domains in ML - such as images, natural language processing, molecular structures, or HEP data - a classical pre-processing step is often essential to reduce the problem size before operating with quantum algorithms. Also, the optimization procedures are mostly handled by classical routines and are often more effective in that form. The most widely studied examples of hybrid models fall into two categories: 1) parametrized and 2) non-parametrized quantum circuits. The first category includes *Variational Quantum Classifiers* (VQCs) and *Quantum Neural Networks* (QNNs), which optimize their parameters classically to perform an ML task. The latter contains models such as Quantum Support Vector Machines (QSVMs), which use quantum feature maps to compute kernel functions in high-dimensional Hilbert spaces, and Quantum Boltzmann Machines (QBMs) aim to model probability distributions using quantum sampling techniques. Despite the great theoretical potential of these methods, they are currently struggling with major challenges in real-life settings. Current quantum hardware offers only a limited number of qubits, often with short coherence times and high noise contamination. Furthermore, encoding classical data into quantum states remains an open question, and training quantum models faces optimization problems, including barren plateaus - flat regions in parameter space with vanishing gradients. Nevertheless, research into QML continues to evolve, with active efforts to investigate its application in domains where data has an intrinsic quantum structure, such as high-energy physics or quantum chemistry.

Tensor Networks, as quantum-inspired models, were originally developed in condensed-matter physics to model quantum many-body interactions [5–7]. They provide a powerful and efficient framework for representing quantum-like structure and modeling complex correlations in data, while remaining executable on classical hardware, even in low-latency environments [8]. Furthermore, they offer compact representations of high-dimensional

tensors, such as quantum states or neural network layers, by factorizing them into networks of lower-rank tensors connected by shared indices. More recently, they have gained attention in ML due to their favorable computational and representational properties [9–12]. Various TN architectures have been adapted for ML tasks. These include Matrix Product State (MPS) [13, 14], Matrix Product Operator (MPO) [15, 16], Projected Entangled Pair States (PEPS) [14, 17], Tree Tensor Network (TTN) [7], and the Multiscale Entanglement Renormalization Ansatz (MERA) [18]. Each structure captures different types of data correlations: MPS is suited for linear, one-dimensional data; PEPS for two-dimensional grid-like structure; TTN for hierarchical dependencies; and MERA for scale-invariant or multi-scale correlations. These models can be used for a variety of ML problems, including classification, generative modeling, compression of deep neural networks, or analysis of quantum datasets (detailed in Chapter 5). Depending on the data encoding strategy, they can be integrated into hybrid quantum-classical pipelines or used as purely classical models.

The implementation of quantum ML and hybrid quantum-classical models has become accessible due to the growing ecosystem of quantum computing and classical ML software packages. Popular Python-based quantum computing packages, such as `Pennylane` [19], `Qiskit` [20], and `Cirq` [21], offer many tools for designing, simulating and deploying quantum algorithms. In this work, we use `Qibo` [22, 23], both for its capabilities and to support our colleagues in the development of the library.

In contrast, implementing TN models for ML presents more of a challenge. While some libraries already exist for this task, they offer different usability, application scope and compatibility with modern ML workflows. Python-based packages include `TorchMPS` [24], `TensorKrowch` [25], `TenPy` [26], `quimb` [27], and `TensorNetwork` [28]. In the Julia ecosystem, the most developed libraries are `ITensors.jl` [29, 30], `TensorKit.jl` [31] and `Tenet.jl` [32]. These packages have many different benefits and are often closely related to specific research domains, but we identified a lack of integration into end-to-end ML pipelines. In our opinion, more effort is needed to create user-friendly and ML-oriented libraries for the construction, training, and evaluation of TN-based models. This motivates the development of the `tn4ml`, a Python library providing a practical interface for applying TNs to real-life ML tasks.

1.4 Achievements of the Thesis

Following the motivations and objectives, the main achievements of the thesis work are:

- **Design and implementation of a hybrid quantum-classical clustering algorithm for anomaly detection** in the latent space of dijet events at the LHC, integrating a quantum subroutine for distance estimation with classical optimization, and a comparative study with its classical counterpart. This approach is detailed in Ref. [1].

- **A novel application of Matrix Product State** for probabilistic anomaly detection and classification task in HEP. Both tasks include a full pipeline for model initialization, embedding scheme selection and performance benchmarking in a realistic scenario at the LHC. The full work for anomaly detection is described in Ref. [3].

- **Development of `tn4ml` open-source Python library** for TN-based ML models, detailed in Ref. [2] and publicly available at github.com/bsc-quantic/tn4ml.

- **Proof-of-concept study for integrating TNs into a real-time trigger system** at the LHC, demonstrating MPS-based models can achieve competitive performance compared to established deep learning methods.

- **Systematic benchmarking of performance-complexity trade-offs** in TN models by analyzing the impact of bond dimension, data embeddings, and initialization techniques on relevant evaluation metrics, model robustness and inference time. This is demonstrated in practical examples throughout this thesis.

1.5 Thesis Structure

The thesis is organized as follows. The first part introduces the theoretical foundations across several relevant domains. Chapter 2 outlines the structure of the classical ML learning pipeline, with the emphasis on applications in HEP. Chapter 3 provides a historical overview of quantum computing, followed by a detailed explanation of the already presented core concepts of quantum computation. This builds the connections to the material

in Chapter 4, which introduces tensors and tensor operations as the foundation for describing various types of TNs and their characteristics. Furthermore, Chapter 5 explores the potential of TNs as ML models, focusing on supervised learning and probabilistic modeling. Chapter 6 presents the `tn4ml` software library, developed for the practical use of TNs in ML pipelines, and demonstrates its application through two public dataset examples.

To understand the relevance of the HEP applications addressed in this thesis, Chapter 7 describes the datasets used, detailing physics behind the event structures, dataset formation, and feature distributions. The following chapters cover applied research conducted in this work: Chapter 8 investigates a hybrid quantum-classical clustering method for anomaly detection in proton-proton collision events; Chapter 9 presents the use of an MPS for probabilistic modeling and anomaly detection in the continuous latent space of collision events; and Chapter 10 demonstrates the application of an MPS model for a classification task relevant to real-time trigger selection algorithms at the LHC.

Finally, Chapters 11 and 12 provide a broader discussion of the thesis contributions, mention existing limitations and challenges, and summarize the overall findings and future directions of the research.

Chapter 2

Classical Machine Learning

Machine learning (ML), as a branch of computer science, is a crucial part of nearly every aspect of our lives, and its synergy with other scientific fields is increasingly important. ML focuses on developing systems that can learn patterns from data and make decisions without being explicitly programmed. These models can improve over time through optimization strategies and are adaptable to a wide range of tasks, including image recognition, natural language processing, medical diagnosis, and recommendation systems. Different real-world problems require different types of learning, which also depend on the nature of the data. These include supervised learning, where the model learns from labeled samples; unsupervised learning, where models learn from the patterns of unlabeled data; and other methods like semi-supervised or reinforcement learning. For each of these learning problems, there is a defined objective function, which reflects the final goal. To optimize the model, the objective function leads the optimization process in the direction of smaller errors. The most common optimization strategy in ML is the gradient descent method, especially effective when the objective function is convex, which guides the search in the direction of the steepest decrease of the function by iteratively updating the model's parameters to minimize the error. In this work, we explore a supervised classification problem and anomaly detection tasks, which are instances of unsupervised learning.

2.1 Learning Pipeline

There are three important parts of the learning pipeline. The first one is data, as it is crucial to present it correctly and be able to exploit all its characteristics. The second one is the learning model, defining the learning space either deterministically or probabilistically. Lastly, it is important to define the optimization strategy to be able to select the best model describing the learning problem.

2.1.1 Data

Data are the foundation of the learning process. The characteristics of data - such as discrete, continuous, time-dependent, periodic, non-negative, or complex - define the data domain and impose structural constraints on the learning process. In most ML applications, more data leads to a better solution. Depending on the type of learning, data can be represented in different ways. For supervised learning, the dataset is defined in pairs of inputs and labels (target output), $\mathcal{D} = \{(x_1, y_1), ..., (x_m, y_m)\}$. These pairs are assumed to be drawn from a joint distribution $p(x, y)$. On the other hand, for an unsupervised learning task, data samples only consist of inputs, $\mathcal{D} = \{x_1, ..., x_M\}$, which were drawn from a distribution $p(x)$ [33]. A standard assumption in statistical learning theory is that the data is independently and identically distributed (*i.i.d.*) from some unknown distribution. This is an ideal assumption, mostly not true in realistic settings as data is often not independent nor identically distributed, but it is still a useful approximation since *non-i.i.d.* scenarios are significantly more difficult to analyze. In practical applications of learning algorithms, the performance is satisfying even under mild violations of the *i.i.d.* assumption. In a classical ML setting, data comes often from a numerical domain, and if that is not the case, it is required to find its numerical representation. The numerical representation can be in different formats, such as directly from a real domain $\mathbb{R}^N$, from the space of N-bit binary strings $\{0, 1\}^{\otimes N}$ or as multi-dimensional tensors each from $\mathbb{R}^N$. Before presenting data to the following parts of the learning pipeline, a common practice is the preprocessing step, which can include standardization, scaling numerical values to a specific range, grouping, or reshaping the input features. Different types of learning tasks require different structures of data, which is discussed in the following sections for specific problems investigated in this work.

2.1.2 Model

In machine learning, a *model* is a mathematical function, computational structure, or an algorithm that drives the learning problem. It is an abstract representation of patterns and relations in the data. Mathematically, a model can be viewed as a function $f_\theta : \mathcal{X} \to \mathcal{Y}$, where:

- $\mathcal{X}$ is the input space (e.g., images, vectors, tensors),
- $\mathcal{Y}$ is the output space (e.g., labels, real values, probabilities),
- θ are the parameters learned during the optimization process of an objective function.

Here, we explain deterministic and probabilistic models, drawing inspiration from descriptions in Ref. [33].

A *deterministic model* defines a supervised learning problem, and is represented as a function $f_\theta : \mathcal{X} \to \mathcal{Y}$ where $f_\theta(x) = y$ and $x \in \mathcal{X}, y \in \mathcal{Y}$. This kind of model can define a classification (discrete labels) or a regression (continuous labels) problem.

A *probabilistic model* for supervised learning, defined as $p_\theta : \mathcal{X} \times \mathcal{Y} \to [0,\ 1]$, can either be a joint probability distribution $p_\theta(x, y)$ or a conditional distribution $p_\theta(y \mid x)$, where $x \in \mathcal{X}, y \in \mathcal{Y}$. A probabilistic model for unsupervised learning is defined as $p_\theta : \mathcal{X} \to [0,\ 1],\ p_\theta(x),\ x \in \mathcal{X}$. This type of model can learn to define a probability of input samples, or can learn to generate new samples (*generative model*).

If a probabilistic model outputs a probability distribution over possible outcomes, e.g., $p_\theta(y|x)$, it can be translated to a deterministic outcome based on some decision rules. Common rules include:

- maximum a posteriori (MAP) which selects the label with the highest predicted probability $\hat{y} = \arg\max_y p(y \mid x)$;
- sampling-based rules used in Bayesian methods;
- applying a threshold between classes, such as in binary classification.

On the contrary, a deterministic model with continuous outputs can be turned into a probabilistic model by interpreting its outputs as parameters of a probability distribution. Similarly, introducing noise to deterministic outputs can result in a probabilistic outcome, which can help the model capture variability in the data.

2.1.3 Optimization

A model f_θ is a part of a family of functions $\{f_\theta\}$ which are defined by parameters θ, called sometimes *hyperparameters*. The model's goal is to generalize well on unseen instances by learning the underlying structure of data domain through the training process guided by the optimization. The first part of the optimization includes the selection of an objective function that defines the optimization landscape. These objective functions, or loss functions, quantify the distance of the model's parameters to the optimal point in the landscape, often the minimum. Common functions include *mean square error* (MSE) for regression, which penalizes the squared distance between predicted and actual labels, and *cross-entropy* loss for classification, which measures the alignment of predicted probability distribution and true labels. Optimization is often performed with gradient-based methods, such as Stochastic Gradient Descent (SGD) [34], Adam [35] or RMSprop [36], which update the parameters in an iterative process based on gradients of the loss function with respect to model parameters.

2.2 Types of Learning Problems

In machine learning, there are different types of learning problems: supervised, unsupervised, semi-supervised, self-supervised and reinforcement learning, where each can branch to subtasks. In this work, we detail *classification*, a discrete supervised learning problem, and an unsupervised setting in the form of *anomaly detection*.

2.2.1 Classification

Classification is a type of supervised ML problem whose goal is to assign labels to input samples from one of the predefined classes or categories. The model learns from a dataset containing pairs of data inputs with their corresponding discrete output labels,

$\mathcal{D} = \{(x_1, y_1), ..., (x_m, y_m)\}$, using this knowledge to predict the class of unseen samples. These problems can be binary (two classes), multi-class, multi-label (one sample has more classes assigned at the same time), or imbalanced. For a multi-class classification, target values are often represented as *one-hot encoded* vectors each having dimension C of the number of classes, with number 1 at the index of the corresponding class, and 0 otherwise. For example for $C = 3$, the vector for an input in class 1 is $y = (1, 0, 0)$. When labels are categorical values, it is often not efficient to represent them as a C-dimensional vector. In these cases, there exists a function that maps a real-valued C-dimensional vector $(y_1, ..., y_c)$ to a probability distribution with condition $\sum_{i=1}^{C} p_i = 1$, called a *softmax* function (Eq. (2.1)). The best solution is often the one with the highest probability of outcome.

$$p_i = \frac{e^{y_i}}{\sum_j e^{y_j}}, i \in \{1, ..., C\}. \tag{2.1}$$

Common classification problems are image recognition (e.g., identifying objects like vehicles, animals, or clothing), spam email detection (binary - spam or no spam), or medical diagnosis (e.g., classifying an image of a tumor as benign or malignant). In this work, we showcase an example of multi-class classification for a high-energy physics dataset.

2.2.2 Anomaly Detection

Anomaly detection (AD) is typically an unsupervised learning problem, where the model learns the underlying distribution of known (normal) samples, and is then used to detect unseen data points that deviate from that learned behavior. The model is trained on known samples, called the *background*, and unseen events, *anomalies*, are used to evaluate the model's detection abilities in the testing phase. This task is commonly applied to detect rare events, noise, outliers, or suspicious patterns. Several machine learning algorithms widely used to discover anomalies are decision trees, one-class Support Vector Machine (SVM), k-means clustering, Naive Bayesian and autoencoders - a type of neural network trained to learn and reconstruct input data. AD is especially useful in scenarios with imbalanced datasets, limited or no labeled anomalies and high-dimensional data that is hard to interpret. All this makes AD valuable in fields such as physics to detect new

phenomena, medical diagnosis, fraud detection, cybersecurity and many more fields of scientific research.

2.3 High-Energy Physics Applications

The search for new-physics phenomena not foreseen by the Standard Model (SM) of particle physics was traditionally led by model-dependent searches. This approach was followed to discover the Higgs boson in 2012 by the CMS and ATLAS collaborations [37, 38]. These searches have a hypothesis set by physics knowledge and assumptions about possible beyond the Standard Model (BSM) processes, and the background is those SM events that create an analogous response in the detection system. This kind of setup is used for ML supervised learning approaches, where data is simulated with the Monte Carlo (MC) method, and each possible new-physics scenario is labeled. After developing a supervised model using MC-simulated data, the testing of the approach is performed using real data coming from the experiments. The drawback of this model-dependent approach is that it is restricted only to a priori formulated hypotheses, and it might end up missing some new signal. This is where anomaly detection techniques play an important role, where models are trained in an unsupervised or self-supervised fashion, aiming to identify rare or unexpected processes in collision data without any prior assumption about their nature. Methods such as autoencoders, clustering algorithms, and generative adversarial networks have been successfully applied to HEP datasets [39–44], and some are already incorporated in the CMS real-time trigger system [45]. These searches heavily rely on the quality and amount of simulated data, which is important to keep in mind when assessing the performance [44]. After developing and testing these methods in theory and software, implementing anomaly detection models in the LHC infrastructure can face several key challenges imposed by the specific hardware that is used for inference. In the HLT system, these models are executed on a CPU farm, whereas in the L1T system, they are implemented on FPGA devices. The constraints on computational resources and inference latency for real-time model execution are discussed in more detail in the following sections.

Chapter 3

Foundations of Quantum Computation

> **R. Feynman**
> *If you think you understand quantum mechanics, you don't understand quantum mechanics*

In this chapter, we introduce the fundamental concepts of Quantum Computing, such as qubits, superposition, and entanglement, followed by an explanation of unitary operations on qubits, known as gates, which form quantum circuits. As the practical intersection of quantum computing and machine learning was explored to research the potential of quantum algorithms for machine learning applications, here we cover parts of the Quantum Machine Learning field relevant to understanding our work. We also describe hybrid models combining quantum and classical parts, as those are explored in this thesis. Then, we define a quantum state through quantum circuits and describe how to represent it with an TN. Lastly, to motivate the research done on TNs in this thesis, we provide an overview of the realistic challenges of quantum computing today and how TNs can help with some of these challenges in practice.

3.1 Story of Quantum Computing

The idea of quantum computing was motivated from multiple perspectives, combining the fields of quantum mechanics, computer science, mathematics, and information theory.

Information is physical - a foundational statement by Rolf Landauer - sets one of the key principles of quantum computation and quantum information [46]. Historically, computer science, having foundations in mathematics, treated information as an abstract entity independent of the physical systems that processed it. In the 1960s, Landauer argued that the information is tied to the physical system and is governed by the laws of physics, in particular quantum mechanics, since our world is made of atoms [47]. One illustrative example is the erasure of a bit, an irreversible operation that leads to a fundamental loss of information and waste of energy as heat. This raises an interesting question: What if we could design a computer in such a way that information is never lost, or at least minimally lost? This would mean the operations would be logically reversible, and, in principle, could be undone without losing energy. This idea led to the development of *reversible computing* in which computations are designed to preserve information and minimize energy loss. This was developed further in the 1970s by Charles Bennett, who showed that any computation could, in theory, be performed reversibly [48].

At the same time, two physicists, Richard Feynman and David Deutsch, began to question whether a classical computer can efficiently simulate a physical system governed by the laws of quantum mechanics. Feynman, one of the pioneers of quantum computing, proposed an alternative to classical computation: a quantum computer built on principles of quantum mechanics, which could outperform classical devices for specific tasks. Later, this line of thought followed the idea of Moore's law, which predicted that the computing power of classical computers would double every two years, while their parts would get smaller and smaller in size. This naturally led to the idea of using small, atomic chips for computations. Building upon all this, David Deutsch introduced the concept of a universal quantum computer in 1985, highlighting that *quantum parallelism* - the ability of quantum systems to exist in a superposition - could be used for computational speedup [49]. Later, through the 1990s, this encouraged the development of quantum algorithms, most notably Shor's algorithm for factoring a large integer [50] and Grover's algorithm for unstructured database search [51], both of which demonstrated exponential and quadratic advantages correspondingly. These breakthroughs helped define the

modern field of Quantum Computing, an interdisciplinary domain that encodes and manipulates information using quantum mechanical phenomena, such as entanglement and superposition, which opened a new frontier in science.

As the field matured, the research on the intersection of quantum computation and machine learning gave rise to the subfield of Quantum Machine Learning (QML), which covers four different intersections:

(1) Analyzing classical data using classical algorithms that simulate quantum systems;

(2) Analyzing classical data with quantum algorithms;

(3) Model quantum data with classical ML algorithms;

(4) Analyze quantum data with quantum algorithms.

In this chapter, we introduce the foundational language and concepts needed to understand quantum computation, and its connection to machine learning - specifically focusing on points (1) and (2) - and explain how quantum-inspired methods can help us model and interpret quantum systems more efficiently.

3.2 Qubits, Superposition, Entanglement

Qubit and Superposition The fundamental unit of quantum computation is the quantum version of a classical bit, known as a *qubit*. As previously discussed, a qubit is a physical object, but further in the text, we will present it using an abstract mathematical framework to generalize over many possible physical implementations. To describe a qubit, we use Dirac (Bra-Ket) notation [52]. A qubit "lives" in a two-dimensional complex vector space that has two basis states $|0\rangle$ and $|1\rangle$, corresponding to the classical bit values 0 and 1. The core difference between a classical bit and a qubit lies in the qubit's ability to exist in a *linear combination* of these basis states - a phenomenon known as *superposition*. This can be expressed as:

$$|\psi\rangle = \alpha|0\rangle + \beta|1\rangle, \tag{3.1}$$

where α and β are complex coefficients of basis states $|0\rangle$ and $|1\rangle$ correspondingly. These basis states are orthonormal and describe a vector space known as a *Hilbert space*. Another fundamental difference between classical bits and qubits lies in the observation, or *measurement*. When we observe a classical bit, we deterministically observe either value 0 or 1. On the other hand, the observation of a qubit is governed by the laws of quantum mechanics, specifically the *Born Rule*. This rule states that the probability of measuring a specific outcome is equal to the squared magnitude of its corresponding coefficient in the quantum state. For a qubit in the superposition Eq. (3.1), the probability of being in state $|0\rangle$ upon measurement is $|\alpha^2|$, and being in state $|1\rangle$ is $|\beta^2|$. Since these are probabilities, the normalization condition $|\alpha^2|+|\beta^2| = 1$ needs to be satisfied, which geometrically implies that the qubit is a unit vector in a two-dimensional complex vector space [53]. This shows an important quantum property: a qubit can exist in a superposition of states until it is measured, making the process of measurement inherently probabilistic. A useful visual representation of a qubit is given by a *Bloch sphere*, a three-dimensional sphere for visualizing one qubit state as a point on the surface. A qubit is represented by a quantum state as a complex wave function with complex coefficients α and β, which have both magnitude and a phase. These complex coefficients can be expressed using polar coordinates: $\alpha = |\alpha|e^{i\phi_\alpha}$ and $\beta = |\beta|e^{i\phi_\beta}$. Through the next mathematical procedure, we simplify the degrees of freedom and apply a global phase normalization, finally arriving to the representation:

$$
\begin{aligned}
|\psi\rangle &= |\alpha|e^{i\phi_\alpha}|0\rangle + |\beta|e^{i\phi_\beta}|1\rangle \\
e^{-i\phi_\alpha}|\phi\rangle &= e^{-i\phi_\alpha}(|\alpha|e^{i\phi_\alpha}|0\rangle + |\beta|e^{i\phi_\beta}|1\rangle) && \text{(multiply by } e^{-i\phi_\alpha}\text{)} \\
\cancel{e^{-i\phi_\alpha}}|\psi\rangle &= |\alpha||0\rangle + |\beta|e^{i(\phi_\beta-\phi_\alpha)}|1\rangle && \text{(global phase in } e^{-i\phi_\alpha} \text{ is not observable)} \\
|\psi\rangle &= |\alpha||0\rangle + |\beta|e^{i\phi}|1\rangle && \text{(substitute } \Delta \text{ with } \phi\text{)} \\
|\psi\rangle &= \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle && \text{(parametrize with } \theta \text{ and } \phi\text{)}
\end{aligned}
$$

This final expression can now be visualized at the *Bloch sphere* (see Fig. 3.1), where θ has values between $[0, \pi]$ positioning a qubit along the z-axis, and ϕ has values from $[0, 2\pi]$ determining the position of a qubit in x-y plane around z-axis.

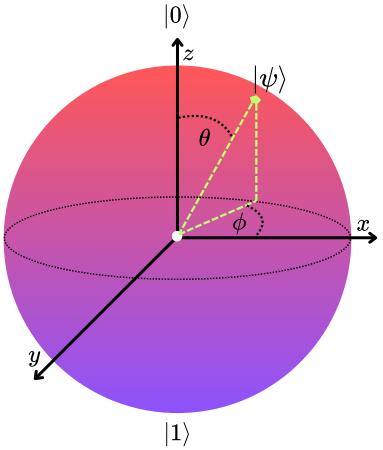


FIGURE 3.1: Representation of a Bloch Sphere for a quantum state $|\psi\rangle$ parametrized with θ and ϕ: $|\psi\rangle = \cos\left(\frac{\theta}{2}\right)|0\rangle + e^{i\phi}\sin\left(\frac{\theta}{2}\right)|1\rangle$.

What happens if we connect two or more qubits into a system? In this case, the Bloch sphere representation is not valid, but it is possible to describe it in an abstract mathematical formalism. Consider the case of having two classical bits, then the system can be in one of four possible states - 00, 01, 10, and 11. Similarly, a system of two qubits is described by four basis states - $|00\rangle, |01\rangle, |10\rangle$ and $|11\rangle$. A general quantum state of these qubits is a superposition of these basis states, each with a corresponding complex coefficient, or *amplitude*. This state can be written as:

$$|\psi\rangle = \alpha_{00}|00\rangle + \alpha_{01}|01\rangle + \alpha_{10}|10\rangle + \alpha_{11}|11\rangle. \tag{3.2}$$

Each basis state is measured with probability $|\alpha_x|^2$, where $x \in \{0,1\}^2$ stands for *"x is a string of length 2, each letter being either 0 or 1"* [53]. The probabilities need to satisfy the normality condition $\sum_{x\in\{0,1\}^2} |\alpha_x|^2 = 1$.

Entanglement Now that we understand how to express a linear combination for a two-qubit system in an abstract way, we can begin to explore the difference between *separable* and *non-separable* (or *entangled*) states. This leads us to another fundamental concept in quantum mechanics: *entanglement*. A quantum state is said to be *separable* if it can be expressed as a tensor product of two single-qubit states $|\phi\rangle$ and $|\psi\rangle$, which are

independent of one another. Mathematically, this is written as:

$$
\begin{aligned}
|\Phi\rangle &= |\phi\rangle \otimes |\psi\rangle \\
|\Phi\rangle &= (\alpha_0|0\rangle + \alpha_1|1\rangle) \otimes (\beta_0|0\rangle + \beta_1|1\rangle) \\
|\Phi\rangle &= \alpha_0\beta_0|00\rangle + \alpha_0\beta_1|01\rangle + \alpha_1\beta_0|10\rangle + \alpha_1\beta_1|11\rangle,
\end{aligned}
$$

where the final product is equivalent to the general two-qubit state defined in Eq. (3.2), but under the condition that the amplitudes factor into products of single-qubit coefficients. This separable structure describes no entanglement between the qubits.

On the other hand, an *entangled* state cannot be expressed simply as a tensor product of individual qubits. The most famous example of such a state is the *Bell state*, named after the physicist John Bell. There are four distinct Bell states, each describing a maximally entangled two-qubit state. For demonstration, we consider one of them to explain the concept of entanglement. Given the Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$, the measurement outcome of one qubit is determined by the measurement of the other. For instance, if the first qubit is measured and found in state $|0\rangle$, the second qubit will be measured in state $|0\rangle$ too. Analogously, if the first qubit is measured in $|1\rangle$, the post-measurement state of the system collapses to $|11\rangle$. This concept of inseparable measurement outcomes is the proof of the existence of quantum entanglement. To generalize this to a system of N qubits, the computational basis states take the form $|i_1 i_2 ... i_N\rangle$, and the full quantum state needs 2^N complex amplitudes to be described. This shows how information in quantum systems grows exponentially with the system size.

3.3 Quantum Gates

To develop a quantum algorithm, one must perform operations on quantum systems, either on a single qubit or simultaneously on multiple qubits. These operations are performed using *quantum gates*, which are assembled into a *quantum circuit*, similar to a classical electrical circuit composed of wires and logical gates. In a quantum circuit, each wire represents a qubit.

It is important to note that a quantum gate U acts *linearly* on a quantum state in a higher-dimensionl Hilbert space, which is imposed by the only constraint of being unitary, i.e., it satisfies $UU^\dagger = \mathbb{I}$ under ideal conditions[1].

As abstract mathematical objects, quantum gates are represented as matrices. A single-qubit gate acts on one qubit at a time and transforms its quantum state by performing rotation or reflection on the Bloch sphere. The most important single-qubit gates, presented in Table 3.1, include:

- Hadamard gate H - creates superposition,
- Pauli gates X, Y, and Z - perform rotations around the x, y and z axis,
- Phase gates S and T - apply controlled phase shifts.

To create entanglement, we extend these definitions of quantum gates to two-qubit gates that operate on pairs of qubits. Two important gates acting on two qubits, mentioned throughout this work, are presented in Table 3.2, along with their corresponding circuit diagrams and matrix representations. The essential two-qubit gate for creating entanglement is the *Controlled-NOT* (CNOT) gate. When combined with a complete set of single-qubit gates, it forms what is known as a *universal gate set*. This implies that any arbitrary multi-qubit quantum operation can be decomposed into a sequence of gates from this universal set. Why are such decompositions important? Because current quantum hardware is constrained to implementing only a limited set of single-qubit rotations and CNOT gates. Thus, it is essential to express complex quantum operations as combinations of these basic gates to be able to execute them on real quantum devices.

3.4 Quantum Circuits

Combining multiple qubits and quantum gates creates a *quantum circuit*. A simple example of such a circuit preparing a Bell state $|\Phi^+\rangle = \frac{1}{\sqrt{2}}(|00\rangle + |11\rangle)$ is shown in Fig. 3.2. This circuit consists of two qubits, one single-qubit gate and one two-qubit gate, where the operations are applied in parallel across all qubits (wires), reading from left to right.

[1] In non-ideal, practical conditions, quantum operations are affected by noise and decoherence.

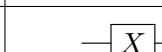
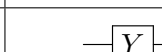
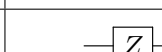
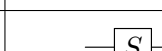
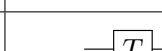
Gate	Circuit Symbol	Matrix
Hadamard (H)		$\frac{1}{\sqrt{2}}\begin{pmatrix}1 & 1\\ 1 & -1\end{pmatrix}$
Pauli-X (X)		$\begin{pmatrix}0 & 1\\ 1 & 0\end{pmatrix}$
Pauli-Y (Y)		$\begin{pmatrix}0 & -i\\ i & 0\end{pmatrix}$
Pauli-Z (Z)		$\begin{pmatrix}1 & 0\\ 0 & -1\end{pmatrix}$
Phase (S)		$\begin{pmatrix}1 & 0\\ 0 & i\end{pmatrix}$
Phase (T) ($\pi/8$ gate)		$\begin{pmatrix}1 & 0\\ 0 & e^{i\pi/4}\end{pmatrix}$

TABLE 3.1: Single qubit gates, their circuit symbols, and matrix representations.

Gate	Circuit Symbol	Matrix
CNOT		$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\\ 0 & 0 & 1 & 0\end{pmatrix}$
SWAP		$\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 0 & 1\end{pmatrix}$

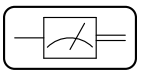
TABLE 3.2: Two-qubit gates, their circuit symbols, and matrix representations.

An important operation in quantum circuits is the *measurement*, an irreversible process that collapses a quantum state into one of its possible outcomes. Measurement is the only *non-linear* operation in a quantum circuit. For instance, when measuring in the computational basis, the quantum state $|\psi\rangle = \alpha|0\rangle + \beta|1\rangle$ collapses to $|0\rangle$ with probability $|\alpha^2|$ and $|1\rangle$ with probability $|\beta^2|$. Mathematically, measurement is described using a Hermitian operator M with a set of eigenstates $\{|k\rangle\}$ and corresponding projectors $\Pi_k = |k\rangle\langle k|$. Given that, the measurement projects a quantum state onto one of the eigenstates of the observable being measured, with the probability $P(k) = \langle\psi|\Pi_k|\psi\rangle$ to obtain the state:

$$|\psi\rangle \longrightarrow \frac{\Pi_k|\psi\rangle}{\sqrt{\langle\psi|\Pi_k|\psi\rangle}}.$$

For the measurement in the computational basis, these projectors are $\Pi_0 = |0\rangle\langle 0|$ and $\Pi_1 = |1\rangle\langle 1|$. We can generalize the measurements using a set of positive operators $\{E_k\}$, called Positive Operator-Valued Measures (POVMs), which satisfy conditions $E_k \geq 0, \sum_k E_k = \mathbb{I}$. They are more broadly used for applications in quantum error correction

and cryptography. The measurement symbol in a quantum circuit is given by:



Measurements are usually placed at the end of the quantum circuit, where they extract the final result of the algorithm. However, in the practical applications on hardware devices today, it is possible to place them in the middle of the circuit for conditional logic, error detection and correction, or hybrid quantum-classical processing. In Fig. 3.2, the measurement operator is measuring both qubits and storing the classical information into register c, where the possible measured states are 00 and 11.

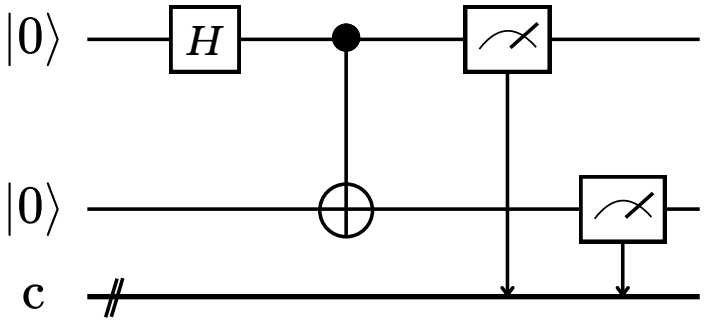


FIGURE 3.2: A quantum circuit representing a Bell state with two qubits prepared in state $|0\rangle$, a Hadamard gate H applied on the first qubit, and a Controlled-NOT gate applied on both qubits. The measurement operator measures the qubits into a classical register c consisting of two classical bits (two lines on the left crossing the lowest wire).

3.5 Quantum Algorithms as Machine Learning Models

Quantum Data Encoding When applying quantum algorithms to classical data, the most important step is to choose an appropriate encoding strategy which will describe the mapping of classical data to a quantum state. This process, known as a *state preparation*, is crucial for determining the expressive power and efficiency of the quantum algorithm. Everything else depends on the encoding step, and in many cases, it contributes significantly to problem-solving. The data encoding step can be interpreted as a *feature map*, similar to those used in classical kernel methods. It maps the data from a classical input

space $\mathcal{X}$ to a higher-dimensional Hilbert space $\mathcal{H}$, in which quantum operations are performed. If in this space exists a well-defined inner product, then the quantum encoding allows powerful methods such as kernel estimation and quantum similarity measures [33].

Let $x \in \mathcal{X}$ be a single data point and $\phi : x \mapsto |\phi(x)\rangle \in \mathcal{H}$ a quantum feature map. Then the similarity between two data points can be expressed as:

$$\langle \phi(x) | \phi(x') \rangle, \tag{3.3}$$

where $x' \in \mathcal{X}$ is the other data point. This similarity can be estimated with quantum inference (as explained in Sec. 8.4).

The most common strategies to encode a single data point x or the whole dataset $\mathbf{x}$ into an n-qubit system are:

- *Basis Encoding* encodes binary data directly into computational basis states. It is only efficient for binary data, as it requires one qubit per bit (e.g., 101 becomes $|101\rangle$).
- *Amplitude Encoding* encodes a normalized vector $x \in \mathbb{R}^{2^n}$ into the amplitudes of a quantum state, resulting in a compact representation using n qubits (e.g., $|\psi\rangle = \sum_{i=0}^{2^n-1} x_i |i\rangle$).
- *Angle or Rotation Encoding* encodes features of input vector x as rotation angles in quantum gates (e.g., $R_y(x_i)$ applied to i^{th} qubit), requiring one qubit per feature.
- *Hamiltonian Encoding* encodes classical data into the parameters of a Hamiltonian $H(\mathbf{x})$ with quantum evolution defined by $e^{-iH(\mathbf{x})t}$, which is best suited for physics-based models and to represent complex correlations. Here, $\mathbf{x}$ represents the whole dataset matrix $N \times M$ [33].

Although quantum data encoding strategies are context-dependent, tied to the nature of the data and the amount of accessible data, they can also bring an enhancement to the classical ML algorithm, in terms of computational efficiency and overall performance [54, 55]. In this work, we explored a version of *amplitude encoding*, explained in the latter Sec. 8.4.1.

QML models We already know that an ML model can be defined either as an architecture with learnable parameters or as a structure that is learned from data itself. To translate this concept into the domain of quantum computing, we introduce two primary types of Quantum Machine Learning models: (1) parametrized models, and (2) data-dependent models.

Parametrized Quantum Circuits (PQCs) refer to quantum circuits that contain gates with tunable parameters, for example, rotation angles. PQCs are trained to minimize a loss function, similarly to classical ML models, and the optimization is usually performed in a variational hybrid fashion, incorporating both quantum and classical computational components. When this kind of training loop is used, the PQCs are called *Variational Quantum Circuits* (VQCs). These can be used in several QML architectures, such as:

- Variational Quantum Classifiers, for supervised learning;
- Quantum Neural Networks (QNNs), similarly built to classical NNs;
- Quantum Generative Models, such as Quantum GANs.

In mathematical terms, a VQC implements a function $f_\theta(x)$ where x is a data sample encoded into a quantum state via a chosen quantum encoding procedure, and θ are the tunable parameters of the quantum gates. The output of the function is obtained via a measurement process, which results in a classical output. This output is further passed to classical optimizers, either gradient or non-gradient-based methods, which update the parameters θ. This process is repeated iteratively until it converges.

Data-dependent quantum algorithms are learning from data through classical post-processing, which include: quantum kernel methods (or quantum support vector machine), quantum k-nearest neighbors, quantum clustering (explained in later Sec. 8.4), and amplitude encoding-based models. These non-parametric data-driven methods are often more robust on current quantum hardware as they don't depend on a noisy hybrid optimization process.

3.6 Quantum Simulation

Quantum Simulation refers to a process of modeling complex quantum systems using either controllable quantum computers or efficient classical approximations [56]. In this section, we focus on discussing the latter approach, commonly referred to as *classical quantum simulation*.

Classical Quantum Simulation exploits advanced numerical methods to efficiently represent and analyze quantum systems on classical hardware. This task is very challenging due to the exponential increase of the Hilbert space with the number of particles or qubits. However, some approaches exploit the physical structure of quantum systems, such as entanglement and local interactions, and produce successful simulations. The two most common strategies include: (1) *full state-vector simulation*, an exact method that represents the entire state in memory, hence scales exponentially, and (2) approximate methods that represent a quantum state in a compact representation, thus enabling good scalability. The latter category contains methods like:

- *Tensor Networks*: Efficiently decompose large quantum states into networks of smaller tensors, focusing on low entanglement structures.
- *Neural Quantum States*: Represent quantum wavefunction using neural networks.
- *Quantum Monte Carlo*: Use probabilistic sampling from the distribution of the quantum system to determine physical observables.
- *Classical Shadows*: A method for learning the properties of a system by estimating observables through efficiently recording measurements into classical "snapshots".

In this work, we focus on *Tensor Networks*, which are powerful for simulating ground states of local Hamiltonians[2], lower-dimensional systems, and weakly entangled states. Classical quantum simulation is very important for validating quantum algorithms, the effectiveness of quantum hardware, and for researching new physical models. It is often used as a "sanity check" to evaluate a quantum algorithm, because if there is an effective classical simulation of it, you need to prove the worth of using a real quantum device to implement it.

[2] A Hamiltonian is a Hermitian operator that describes the total energy of the system, both kinematic and potential.

3.7 Challenges of Quantum Computing and Quantum Machine Learning

Although quantum computing and quantum machine learning are advancing rapidly, many practical challenges still limit their useful real-world applications. One of the biggest issues comes from qubits being very unstable and sensitive systems. Any noise coming from the environment can disturb them, introducing hardware noise that leads to imprecise qubit operations. This phenomenon is known as decoherence, which refers to the loss of quantum information due to interaction with the environment and unavoidable collapse to a classical system. Different physical implementations of qubits introduce varying fidelities, which are still far from good-quality qubits. As a result, today, quantum devices are called Noisy Intermediate-Scale Quantum (NISQ) machines and have a restricted number of qubits (up to hundreds) with short coherence times. One proposed solution to overcome these limitations is to operate qubits in highly controlled environments and to systematically correct this noise through *quantum error correction*. This has led to the development of fault-tolerant architectures that encode a single logical qubit into many physical qubits. This requirement of additional qubits makes it still very difficult to build large-scale quantum computers. For example, recent estimates show that factoring a 2048-bit RSA number using a quantum algorithm, known as Shor's algorithm, would require several thousand logical qubits and close to one million physical qubits [57]. Another realistic topic for optimizing the implementation on hardware is *quantum compilation*, which focuses on minimizing the gate count and circuit depth to avoid accumulating noise, which causes unnecessary errors. On the algorithmic side of quantum machine learning, the approaches are still largely theoretical. Many proposed algorithms rely on ideal assumptions and do not yet reflect real-world scenarios. The community is still actively exploring the real purpose of QML [58], and how to steer its development in the right direction, where it can offer genuine advantages. A known obstacle with the optimization of VQCs is the phenomenon of *barren plateaus*, which explains how the flatness of an exponential landscape can limit the optimization process. Overcoming this issue is still an active area of research [59]. Finally, it is still needed to establish robust benchmarks for QML algorithms and demonstrate clearly that they outperform their classical counterparts [60].

In summary, significant theoretical and experimental progress has been made, but overcoming these challenges still requires an interdisciplinary effort of engineering scalable hardware, developing practical algorithms, and refining theoretical studies. All this could lead to scalable, robust, and practical implementations of quantum computing and QML algorithms.

Chapter 4

Tensor Network Methods

E. Puljak

It's just a network of tensors. Simple as that. Or much more complex?

In this chapter, we introduce the theoretical foundations of Tensor Networks (TNs), beginning with the description of tensors and the graphical notation commonly used to represent them, followed by the most essential tensor operations. We then briefly discuss the origins of TNs in condensed matter physics, and provide an overview of different types of network structures. Specific focus is given to linear TN structures, known as Matrix Product States (MPS), as they are thoroughly studied for ML applications throughout this thesis. We also highlight particularly useful features of TNs - gauge freedom, canonical form, differentiation, normalization - which play an important part in achieving stable and efficient optimization. Then, the Density Matrix Renormalization Group (DMRG) algorithm, originally developed to search for ground states of quantum systems represented as TNs, is here described in the context of optimizing TNs for ML tasks. Lastly, we introduce the decomposition procedure of a quantum state or a large tensor to an MPS, and how TNs are related to quantum circuits.

4.1 Tensors and Graphical Notation

There exist many definitions of tensors, but here we adopt the one we find most intuitive, followed by a brief overview of commonly used tensor operations. A *tensor* can be interpreted either as a multidimensional array or as a multilinear map. In graphical notation, it is represented as a node with a number of edges (indices) corresponding to its rank. This notation is widely used in tensor network theory, as it enhances both visualization and interpretability. For instance, a scalar T is a rank-0 tensor, a vector T_i is a rank-1 tensor, a matrix T_{ij} is a rank-2 tensor, and so on (see Fig. 4.1).

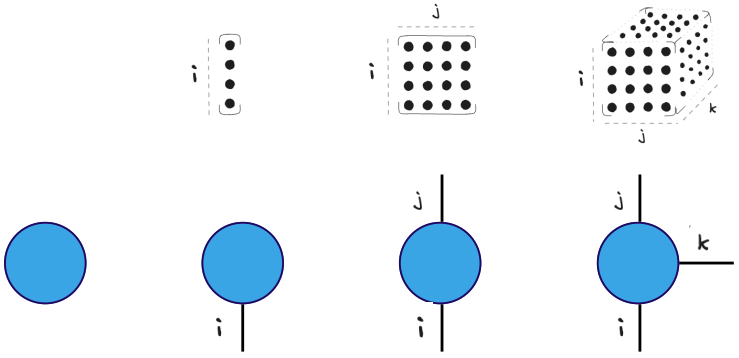


FIGURE 4.1: Graphical representation of a scalar (0-rank tensor) as circle with no indices; a vector (1-rank tensor) as a circle with one index i; a matrix (2-rank tensor) as a circle with two indices i and j; and a 3-rank tensor as a circle with indices i, j and k.

4.1.1 Tensor Operations

In this section, we describe key tensor operations used to manipulate TN objects, including index fusion and splitting, contractions, inner product, singular value decomposition, and other tensor decomposition techniques. These operations are fundamental for understanding algorithms modeled within the Tensor Network framework.

Index Fusion and Splitting Tensors organize information in structured, multidimensional forms, and the way this structure is chosen directly affects computation efficiency. *Index fusion* is a process of combining multiple indices of a tensor into a single index with a larger dimension. This reshaping results in a tensor of lower rank and can simplify operations such as matrix decompositions (e.g., Singular Value Decomposition discussed later). In contrast, *index splitting* is the inverse operation that decomposes a single index into multiple sub-indices, resulting in a higher rank tensor. These operations

are particularly useful for reducing computational costs in certain cases and for organizing data in a more interpretable form. They are both illustrated in Fig. 4.2.

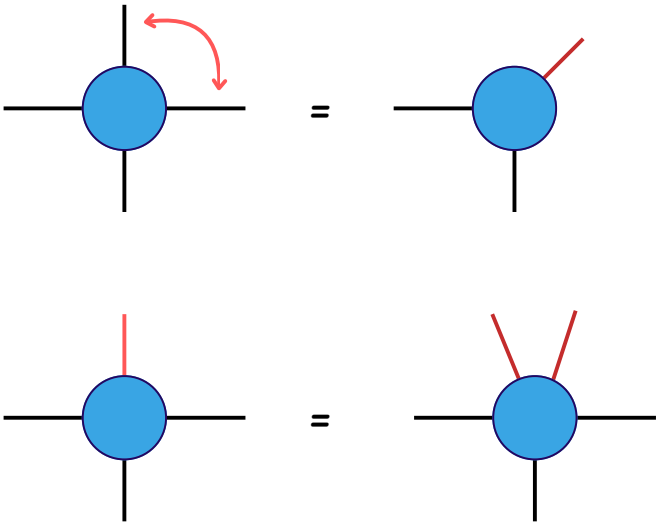


FIGURE 4.2: Graphical representation of index fusion (upper) when two indices are combined into one, and index splitting (lower) as a reverse operation of splitting one index into two.

Contraction When two tensors share a common index, the operation of summing over that index is called a *contraction*. This operation produces a lower-rank tensor by reducing the total number of indices. The simplest example of this operation is a matrix multiplication, which corresponds to a contraction of two rank-2 tensors:

$$C_{ik} = \sum_j A_{ij} B_{jk}, \tag{4.1}$$

as visualized in Fig. 4.3, where a new tensor is a matrix C_{ik}.

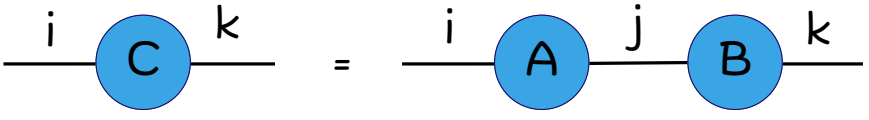


FIGURE 4.3: Graphical representation of a matrix-matrix multiplication - contraction of two rank-2 tensors A_{ij} and B_{jk} by index j resulting in tensor C_{ik}.

Fig. 4.3 is also an example of a simple Tensor Network, having only two tensors connected.

Inner Product An additional useful operation is the *inner product* of tensors, also called the Frobenius inner product, calculated as:

$$\langle A, B\rangle_F = \sum_i \sum_j A^*_{ij} B_{ij}, \tag{4.2}$$

where $A, B \in \mathbb{C}^{m\times n}$ and $\langle A, B\rangle_F$ is a scalar. This is used for computing overlaps of quantum states, calculating the norm of a tensor, or computing loss functions for machine learning tasks.

Singular Value Decomposition A method in linear algebra used to decompose a matrix into a network of three lower-rank matrices is called a *Singular Value Decomposition* (SVD). Given a real matrix A, the SVD creates a product of matrices $A = USV^T$ where U and V are orthonormal matrices and S is a diagonal matrix with real positive elements λ_i on the diagonal (i.e., $S_{ij} = \lambda_i > 0$ and $S_{ij} = 0$ for $i \neq j$). These elements λ_i are called singular values. When matrix A is complex, as it is in quantum information-related computations, then the factorization is $USV^\dagger$ with matrices U and V being unitary[1]. This can be described with mathematical notation:

$$A_{ij} = \sum_k U_{ik} S_{kk} V^\star_{kj}, \tag{4.3}$$

and illustrated in Fig. 4.4, where $\star$ labels $\top$ and $\dagger$ operations for real and complex matrices, respectively. Singular values can be ordered, most commonly in descending order, and the smallest values close to zero can be discarded. This is analogous to Principal Component Analysis (PCA) [61], where the principal directions with small singular values (lowest variance) are removed. Truncation of matrices U and V based on retained singular values introduces a reduced form of a tensor A_{ij} with corresponding error ϵ calculated as:

$$\left\| A_{i,j} - \sum_{k=1}^{m} U_{i,k}\lambda_k V^\star_{k,j} \right\| = \left\| \sum_{k=m+1}^{r} U_{i,k}\lambda_k V^\star_{k,j} \right\| < \sum_{k=m+1}^{r} \lambda_k < \epsilon C, \tag{4.4}$$

where m is the number of retained singular values, r is the rank of A tensor and C is a finite constant. If error ϵ is zero or close to zero (depending on precision), then we can represent a tensor efficiently with a computationally less complex structure [62]. Practical usage of SVD is described in one of the following sections 4.3.

[1] A unitary matrix $U \in \mathbb{C}^{n\times n}$ satisfies: $U^\dagger U = UU^\dagger = I$, where I is an identity matrix and $\dagger$ is conjugate transpose.

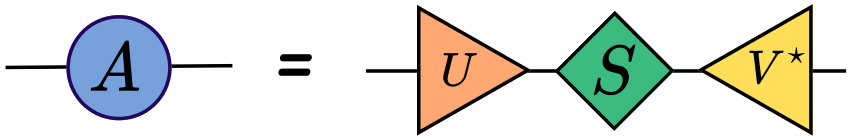


FIGURE 4.4: Graphical representation of singular value decomposition of matrix A into $USV^\star$ where $\star \in \{\dagger, T\}$.

Tensor Decompositions The concept of decomposition using SVD extends to higher-order tensors, leading to *tensor decompositions* that aim to decompose a tensor into simpler structures, while capturing its higher-order interactions and latent components. One common generalization is the *Tucker decomposition*, which represents a tensor $A \in \mathbb{R}^{I_1 \times I_2 \times \ldots \times I_N}$ as a multi-linear product of a smaller core tensor $\mathcal{G}$ and a set of orthonormal factor matrices $U^{(n)} \in \mathbb{R}^{I_n \times R_n}$ along each mode:

$$\mathcal{A}_{i_1 i_2 \cdots i_N} = \sum_{j_1=1}^{R_1} \cdots \sum_{j_N=1}^{R_N} \mathcal{G}_{j_1 \cdots j_N} \prod_{n=1}^{N} U^{(n)}_{i_n j_n}, \tag{4.5}$$

where R_n is the rank along mode n.

Another used form is the *Canonical Polyadic (CP) decomposition*, which represents a tensor as a sum of rank-one tensors:

$$\mathcal{A}_{i_1 i_2 \cdots i_N} \approx \sum_{r=1}^{R} \lambda_r \, u^{(1)}_{i_1 r} \, u^{(2)}_{i_2 r} \cdots u^{(N)}_{i_N r}, \tag{4.6}$$

where λ_r are scalar weights, $u^{(n)}_{i_n r}$ are components of factor matrices along each mode n, and R is the CP rank, i.e., number of components.

These factorizations are often truncated to retain only the most significant components, offering compact representations of high-dimensional data and more efficient computations.

4.2 Tensor Networks

Tensor Networks originate from condensed matter physics, where they were developed to efficiently describe the exponentially large vector space associated with quantum many-body systems. Researchers discovered that many physically relevant states (e.g., ground

states of local Hamiltonians) have limited entanglement and can be accurately approximated using TN-based structures. These methods factorize a global state of the system into a structure of smaller, interconnected tensors to capture local interactions.

In general, a *Tensor Network* is a graph-based structure composed of tensors connected via shared indices. The topology of the network reflects both dependencies in the data and the complexity of performing operations on the network. Linear, one-dimensional (1D) systems are described with structures known as Matrix Product States (MPS) [13, 14] and Matrix Product Operators (MPO) [14, 15]. These are thoroughly studied in this work and are discussed in detail below. Other widely studied TN architectures include a two-dimensional (2D) Projected Entangled Pair State (PEPS) [17], a Tree Tensor Networks (TTNs) with a tree-like structure, and a Multiscale Entanglement Renormalization Ansatz (MERA) [18, 63] architecture. Fig. 4.5 provides illustrative representations of these different TN topologies. In this work, we concentrate on the one-dimensional TN structure and explore its characteristics in depth.

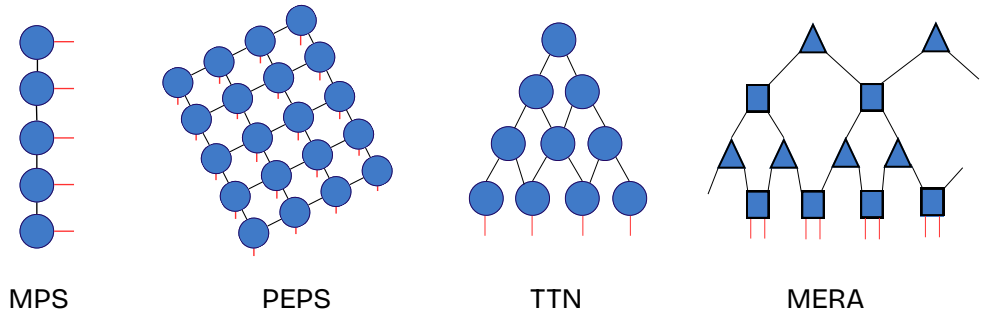


FIGURE 4.5: Graphical representation of the four types of Tensor Networks: (1) Matrix Product State (MPS), (2) Projected Entangled Pair States (PEPS), (3) Tree Tensor Network (TTN) and (4) Multiscale Entanglement Renormalization Ansatz (MERA).

4.2.1 One-Dimensional Structures

In the context of machine learning, 1D TN structures such as the MPS and the MPO are among the most widely used and studied. These two architectures are the foundation of the methods explored in this work. A version of an MPO, Spaced Matrix Product Operator (SMPO), is discussed in this work in the context of anomaly detection.

A *Matrix Product State* is a chain-like, compressed structure used to represent a high-order N-dimensional tensor. In quantum physics, this is mostly used to describe quantum states [13, 14]. Instead of storing a full tensor, MPS factories it into a sequence (or a loop,

depending on the boundary conditions[2]) of at most rank-3 tensors [7, 13]. Graphically, an MPS is typically represented as in Fig. 4.6(1), where each tensor at site k is connected with two virtual bonds (indices) $(D_{\mathrm{k}-1}, D_{\mathrm{k}})$ to its near-neighbor and contains an upper physical index i_{k}. The size of virtual bonds determines how much correlation the MPS can capture between parts of the system, influencing the model's expressive power.

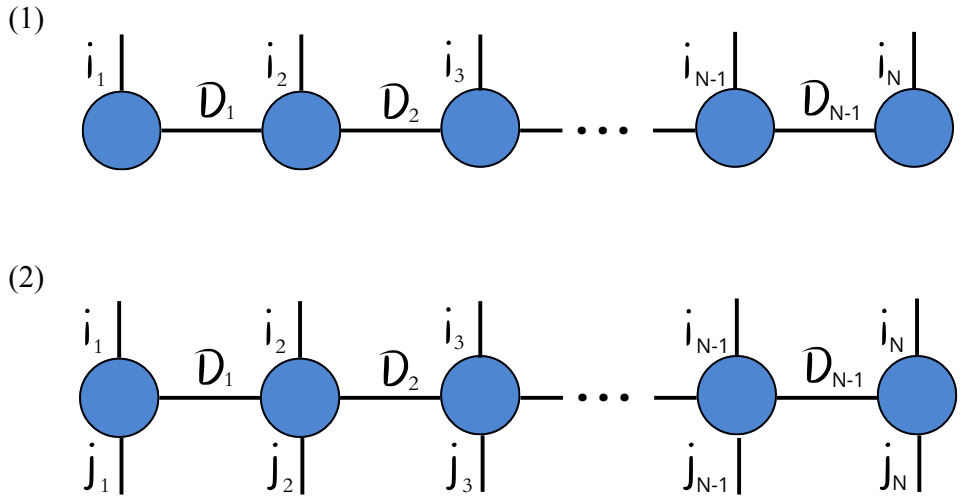


FIGURE 4.6: Graphical representation of (1) MPS with upper physical indices i_k and virtual (bond) indices D_k, and (2) MPO with upper, lower j_k physical and bond indices. Dimensions of all indices are chosen based on some heuristic or physical state of the system.

Formally, a quantum state $|\psi\rangle$ can be represented in MPS form as:

$$|\psi\rangle = \sum_{\substack{i_1,i_2,\ldots,i_{\mathrm{N}} \\ D_1,D_2,\ldots,D_{\mathrm{N}-1}}} A^{i_1}_{D_1} A^{i_2}_{D_1,D_2} \cdots A^{i_{\mathrm{N}}}_{D_{\mathrm{N}-1}} |i_1 i_2 \cdots i_{\mathrm{N}}\rangle. \tag{4.7}$$

Each matrix $A^{i_{\mathrm{k}}}_{D_{\mathrm{k}-1},D_{\mathrm{k}}}$ in the MPS represents a physical basis state $|i_{\mathrm{k}}\rangle$ containing virtual bonds of dimensions $D_{\mathrm{k}-1}$ and D_{k}. This representation is highly efficient, significantly reducing the memory requirements from exponential $\mathcal{O}(d^{\mathrm{N}})$ to polynomial dependence on N - $\mathcal{O}(\mathrm{N}dD^2)$ - where D is the maximum bond dimension across the MPS. When D remains reasonably small, MPS models provide a significant memory advantage. Although an MPS can exactly represent any tensor if the bond dimension is sufficiently large, order of $\mathcal{O}(d^{\mathrm{N}/2})$, in practice, D is typically kept as small as possible to capture only relevant correlations. In quantum terms, this means that the MPS captures the dominant parts of

[2]Open Boundary Conditions (OBC) are more commonly used in ML for modeling finite systems without periodic structure. Periodic Boundary Conditions (PBC), where first and last tensors are connected with a virtual bond, are used for modeling periodic systems.

the entanglement structure. Quantum systems with low entanglement are efficiently approximated using low-bond-dimension MPS, while high-entangled systems require larger bond dimensions. A very important feature of MPS models is that they naturally satisfy *area law* for entanglement entropy, which states that for two parts of the system, the entanglement scaling law depends on the boundary (e.g., the surface area) between them, not the volume[3]. For 1D structures, this boundary is a constant, further supporting the efficiency of MPS representations.

A *Matrix Product Operator* generalizes the concept of an MPS to represent operators acting on high-dimensional spaces [15]. Conceptually, an MPO is a factorized, chain-like representation of a large operator tensor, which has N input and N output indices. Each local tensor in the MPO has up to four indices: two physical (input and output) and two virtual (bonds). Input and output dimensions do not need to be the same. As shown in Fig. 4.6(2), the MPO visually looks similar to an MPS but includes an additional set of indices j_{k} that span the output space. Using mathematical notation, the MPO is expressed as:

$$\hat{O} = \sum_{i,j,\mathrm{D},\mathrm{D}'} \prod_{s} A^{(s)}_{i,j,\mathrm{D},\mathrm{D}'} |i_1, \cdots, i_{\mathrm{N}}\rangle\langle j_1 \cdots j_{\mathrm{N}}|, \tag{4.8}$$

where s marks the site, i and j are input and output physical indices, and D and D$'$ are left and right bond dimensions.

A *Spaced Matrix Product Operator* represents an asymmetric version of an MPO [16], as shown in Fig. 4.7, such that it maps from an input space V described with N upper indices i_{k} to a lower-dimensional output space W with M indices j_{k}, where N $>$ M. Similarly to an MPO, the input and output indices can have different dimensions. Depending on the chosen *spacing* parameter $S \in \mathbb{N}$, which controls the separation between output indices along the chain, the tensors in the SMPO can either have rank-3 or rank-4. The parameter S can be fixed or variable across the network, enabling the free allocation of output indices.

4.2.2 Contraction Paths and Complexity

Contraction path refers to a sequence of contractions over shared indices. When operating with TNs, finding the most efficient contraction path is crucial, as it significantly reduces

[3]The area law property is especially efficient for representing ground states of local gapped Hamiltonians, as explained in Ref.[64, 65]

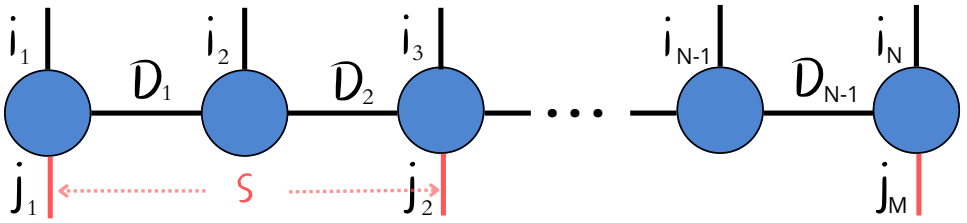


FIGURE 4.7: Graphical representation of an SMPO with upper physical indices i_k, bond indices D_k and lower physical indices j_k. The number of lower indices is determined by the spacing parameter S, which can vary from one index to another.

the computational cost of the operations. In general, for an arbitrary TN structure, determining the optimal contraction path is computationally hard, and it scales exponentially with the network's size. The complexity depends on the type of TN topology, the dimensions of all indices, and the structure of correlations - including entanglement in a quantum context. The *contraction complexity* is given as the product of contracted dimensions, such as the dimensions of physical indices d and the virtual ones D [62]. However, for one-dimensional TNs like MPS or MPO with open boundary conditions, there exists an efficient, sequential contraction order with a cost of $\mathcal{O}(d\mathrm{N}D^2)$ or $\mathcal{O}(d^2\mathrm{N}D^2)$, where N is the number of tensors. This polynomial scaling is one of the reasons the one-dimensional structures are most commonly used across quantum many-body physics and ML applications.

4.2.3 Differentiation

Tensor Network differentiation refers to the process of computing the gradient of a scalar (e.g., an optimization metric) with respect to one or more tensors within the TN. The selected tensor(s) are treated as differentiable variables, while the remaining tensors are fixed during the computation. Visually, this corresponds to removing the target tensors from the network, as illustrated in Fig. 4.8, where gradients are computed with respect to two tensors. In practice, modern implementations rely on automatic differentiation and allow ML frameworks (e.g., `Pytorch`, `JAX`, `Tensorflow`) to handle this process by tracing the computational graph and applying the chain rule throughout the network [66]. During the forward pass, the contraction of the network is recorded, which allows the gradient to be propagated backward through the structure during the training procedure. This is important in settings of variational learning. For differentiable TNs, the backward pass includes computing the contracted environment of each tensor, often with efficient update

schemes (i.e., sweeping optimization). This ensures that gradients respect the multilinear structure of the network and can be used with gradient-based optimizers (e.g., Adam, SGD). Incorporating differentiation within TNs allows one to use TNs for tasks such as classification, generative modeling, and physics-informed learning, which is further detailed in Chapter 5.

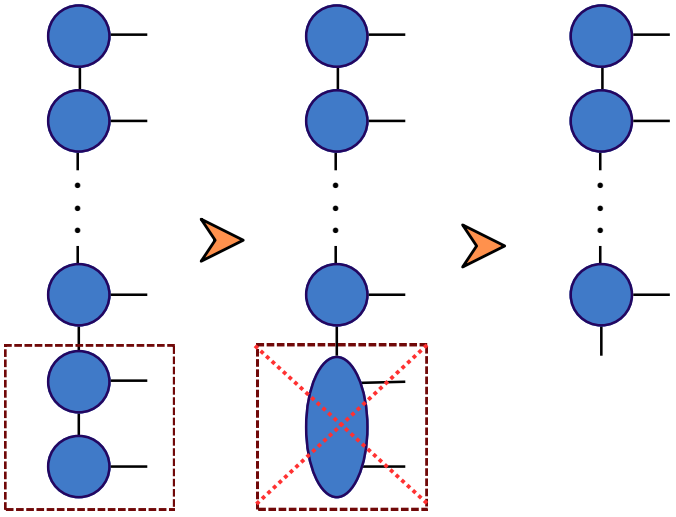


FIGURE 4.8: Graphical representation of a gradient (right) of the Matrix Product State network with respect to two lower tensors.

4.2.4 Gauge Freedom

A useful feature of an MPS is its invariance under local transformations applied to individual tensors, which leave the overall physical state unchanged. This implies that not all MPS forms correspond to distinct physical states. These local transformations are known as *gauge transformations*, and the flexibility to choose different MPS representations of the same physical state is called *gauge freedom*. If the global physical state remains unchanged, the transformation must happen at the virtual level. In particular, inserting a basis transformation V together with its inverse V^{-1} (or Hermitian conjugate †) along a bond index results in the two different MPS representations of the same state - $|\psi_A\rangle$ and $|\psi_B\rangle$. The tensors of state $|\psi_B\rangle$ are obtained by absorbing V into the right virtual index of one tensor and V^{-1} into the left virtual bond of the neighboring tensor (see Fig. 4.9). The only freedom allowing different MPS representations of the same physical state is given by this gauge transformation. This concept is known as the *fundamental theorem of MPS*, which states that two translationally invariant MPS describe the same state if

and only if they are related by a gauge transformation [67]. This naturally leads us to the canonical forms of an MPS.

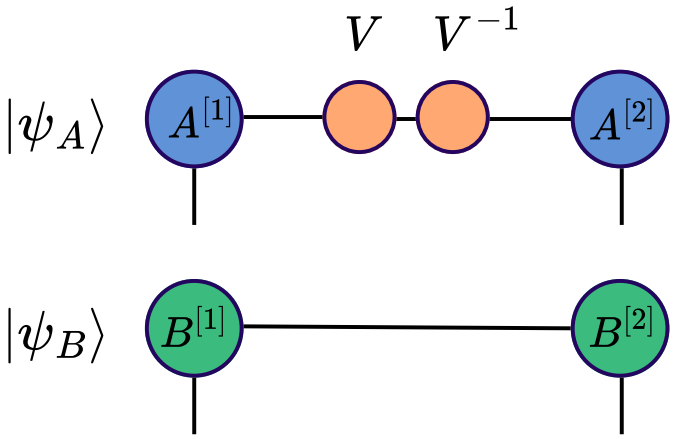


FIGURE 4.9: Graphical representation of a gauge transformation on the bond index between two tensors. Inserting matrix V and its inverse V^{-1}, the tensor $A^{[1]}$ absorbs matrix V, and tensor $A^{[2]}$ absorbs the V^{-1}. This results in two identical states $|\psi_A\rangle = |\psi_B\rangle$.

4.2.5 Canonical Form

A *canonical form* of an MPS is a version of the MPS under gauge transformation that arranges the tensors to satisfy certain orthonormality conditions. Examples of canonical forms are left-canonical, right-canonical, or mixed-canonical forms. In the *left-canonical* form, each tensor A is an isometry and it satisfies:

$$\sum_{i_k} (A^{i_k})^\dagger A^{i_k} = \mathbb{I}, \tag{4.9}$$

which means that when contracting over a physical index k, the result is an identity on the right bond index. The orthogonality center, or coefficient matrix, C, containing entanglement information, is pushed to the right, and all tensors left from it are isometries A_i. The left-canonical MPS form is expressed as:

$$|\psi\rangle = \sum_{\substack{i_1,i_2,\ldots,i_N \\ D_1,D_2,\ldots,D_{N-1}}} A^{i_1}_{1,D_1} A^{i_2}_{D_1,D_2} \cdots A^{i_{N-1}}_{D_{N-2},D_{N-1}} C^{i_N}_{D_{N-1}} |i_1 i_2 \cdots i_N\rangle, \tag{4.10}$$

assuming bond dimensions $D_0 = D_N = 1$.

In the *right-canonical* form, the reverse condition is satisfied:

$$\sum_{i_k} A^{i_k} (A^{i_k})^\dagger = \mathbb{I}, \tag{4.11}$$

and the contraction over k gives the identity on the left bond. The orthogonality center is to the left, and all other tensors to the right are isometries. The mathematical expression follows as:

$$|\psi\rangle = \sum_{\substack{i_1,i_2,\ldots,i_N \\ D_1,D_2,\ldots,D_{N-1}}} C^{i_1}_{D_1} A^{i_2}_{D_1,D_2} A^{i_3}_{D_2,D_3} \cdots A^{i_N}_{D_{N-1},1} |i_1 i_2 \cdots i_N\rangle \tag{4.12}$$

Transforming an MPS into a *mixed-canonical* form involves selecting a specific site k in the chain, such that the tensor at that site is surrounded by left-canonical tensors on the left and right-canonical tensors on the right. This configuration captures entanglement across the chosen bipartition. Fig. 4.10 illustrates the left-, right- and mixed-canonical form of an MPS. Canonical forms are very useful for ensuring numerical stability during tensor operations and optimization procedures, as they simplify norm calculation (explained in the next section). They contribute to the efficient calculation of expectation values and allow efficient access to the entanglement entropy of the system.

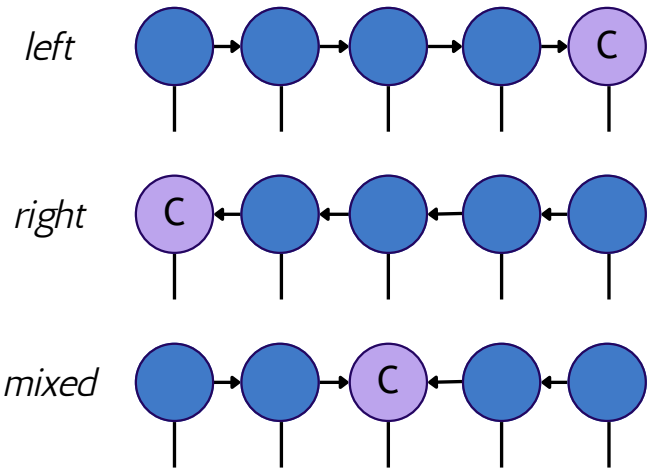


FIGURE 4.10: Graphical representation of canonical forms (left, right, mixed) of the Matrix Product State, where C is the canonical center.

4.2.6 Normalization

Normalization of a TN ensures that the overall output of the network, e.g., a scalar representing a probability, keeps numerical stability during training and inference. For models where the contraction of many tensors can lead to exponential growth or the vanishing of numerical values, such as MPS, normalization is very important. Normalization can be imposed through canonical forms (e.g., left- or right-canonical MPS), explicit normalization (e.g., dividing by the norm of TN), or initializations that put specific constraints on the tensors, such as unitary or orthogonal procedures. The Frobenius norm of an MPS state $|\Psi\rangle$ is defined as:

$$||\Psi||_F = \sqrt{\langle\Psi|\Psi\rangle}, \tag{4.13}$$

where $\langle\Psi|$ is a complex conjugate of $|\Psi\rangle$. Fig. 4.11 visualizes the contraction process of computing the Frobenius norm of a TN.

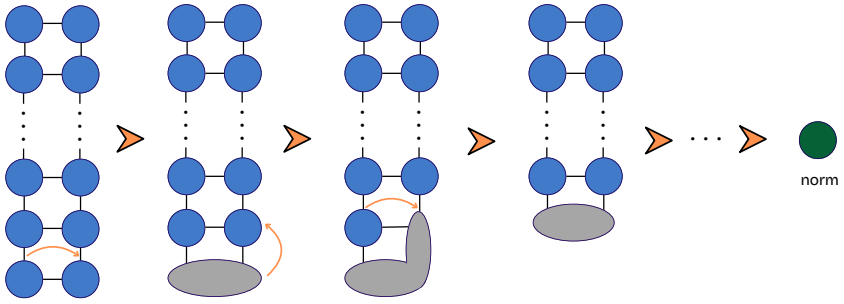


FIGURE 4.11: Schematic graphical representation of the calculation of the Frobenius norm of a Matrix Product State model. Arrows indicate the contraction path between tensors.

If an MPS is in a canonical form, for instance left-, we can compute the norm more efficiently because now the contraction of tensors A^{i_k} yields an identity, as expressed in:

$$\langle\Psi|\Psi\rangle = \sum_{\substack{i_1,\ldots,i_N \\ D_1,\ldots,D_{N-1}}} \left(A^{i_1}_{1,D_1}\cdots A^{i_{N-1}}_{D_{N-2},D_{N-1}} C^{i_N}_{D_{N-1}}\right)^* \left(A^{i_1}_{1,D_1}\cdots A^{i_{N-1}}_{D_{N-2},D_{N-1}} C^{i_N}_{D_{N-1}}\right) \tag{4.14}$$

$$= \sum_{\substack{i_N,i'_N \\ D_{N-1},D'_{N-1}}} C^{i_N *}_{D_{N-1}} \left(\sum_{\substack{i_1,\ldots,i_{N-1} \\ D_1,\ldots,D_{N-2}}} \left(A^{i_1}\cdots A^{i_{N-1}}\right)^*_{D_{N-1}} \left(A^{i_1}\cdots A^{i_{N-1}}\right)_{D'_{N-1}} \right) C^{i'_N}_{D'_{N-1}} \tag{4.15}$$

$$= \sum_{\substack{i_N,i'_N \\ D_{N-1},D'_{N-1}}} C^{i_N *}_{D_{N-1}} \rho_{D_{N-1},D'_{N-1}} \delta_{i_N,i'_N} C^{i'_N}_{D'_{N-1}} \tag{4.16}$$

$$= \sum_{i_N,D_{N-1}} \left| C^{i_N}_{D_{N-1}} \right|^2, \tag{4.17}$$

and illustrated schematically in Fig. 4.12. The contracted environment ρ is defined as:

$$\rho_{D_{N-1},D'_{N-1}} = \sum_{\substack{i_1,\ldots,i_{N-1} \\ D_1,\ldots,D_{N-2}}} \left(A^{i_1}_{1,D_1}\cdots A^{i_{N-1}}_{D_{N-2},D_{N-1}}\right)^* \left(A^{i_1}_{1,D_1}\cdots A^{i_{N-1}}_{D_{N-2},D'_{N-1}}\right), \tag{4.18}$$

which yields in $\rho = \delta$, where δ is the Kronecker delta, acting as identity.

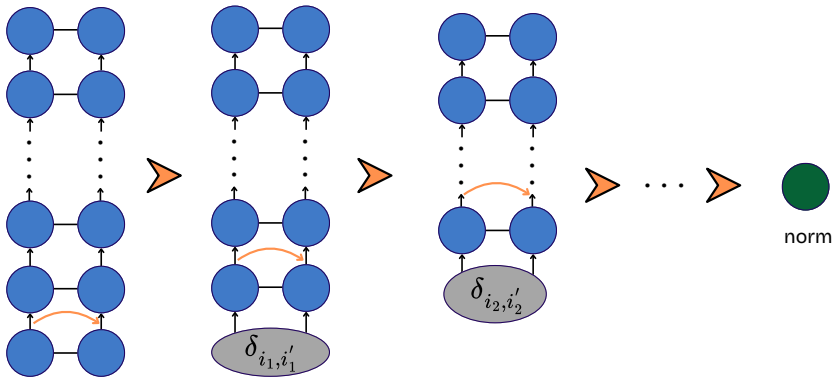


FIGURE 4.12: Schematic graphical representation of the calculation of the Frobenius norm of a Matrix Product State model when MPS is in left-canonical form. Arrows indicate the contraction path between tensors. $\delta_{i,i'}$ is the Kronecker delta for site i.

4.2.7 Density Matrix Renormalization Group

One of the most powerful methods currently available for simulating strongly correlated quantum-many-body systems is the *Density Matrix Renormalization Group* (DMRG) algorithm [68]. In quantum physics, DMRG is a variational algorithm used to approximate ground states of local Hamiltonians of quantum systems in the low-dimensional space. In this work, we describe it as a numerical optimization method applied to TN structures, with a focus on its utility for machine learning model optimization. For one-dimensional TNs, such as MPS [69], we optimize the tensors locally, sweeping back and forth along the chain. At each step, one or more tensors are treated as variational parameters and updated, while other tensors are held fixed. This process often offers the possibility of keeping the most significant eigenvectors to ensure computation efficiency. The optimization converges after a full sweep in both directions, resulting in a minimal change of the target metric. The process is more visually explained in Sec. 6.2.3, when we discuss the practical implementation in the `tn4ml` library.

4.3 Decomposition to Matrix Product State

An exponentially large tensor $|\psi\rangle$, which can represent a quantum state in $(\mathbb{C}^d)^{\otimes N}$, can be decomposed into an MPS. The process begins either on the left or right side, with the reshaping of a full rank tensor d^N into $d \times d^{N-1}$ to prepare $|\psi\rangle$ for the first SVD. The SVD splits the matrix into a left unitary (the first tensor of the MPS $A^{[1]}$), a diagonal matrix of singular values S, and a reshaped right tensor $V^\dagger$. The right tensor $V^\dagger$ is sequentially reshaped and decomposed at each step k, where left indices have dimensionality of a product of physical dimension d and the previous bond dimension D_{k-1}, and the right indices d^{N-k}. At each step, the SVD creates a new rank-3 tensor $A^{[k]} \in \mathbb{C}^{D_{k-1} \times d \times D_k}$, where the bond dimension D_k corresponds to the number of singular values. This process continues until reaching the final state of linearly connected low-rank tensors with a total parameter count of $\mathcal{O}(ND^2d)$.

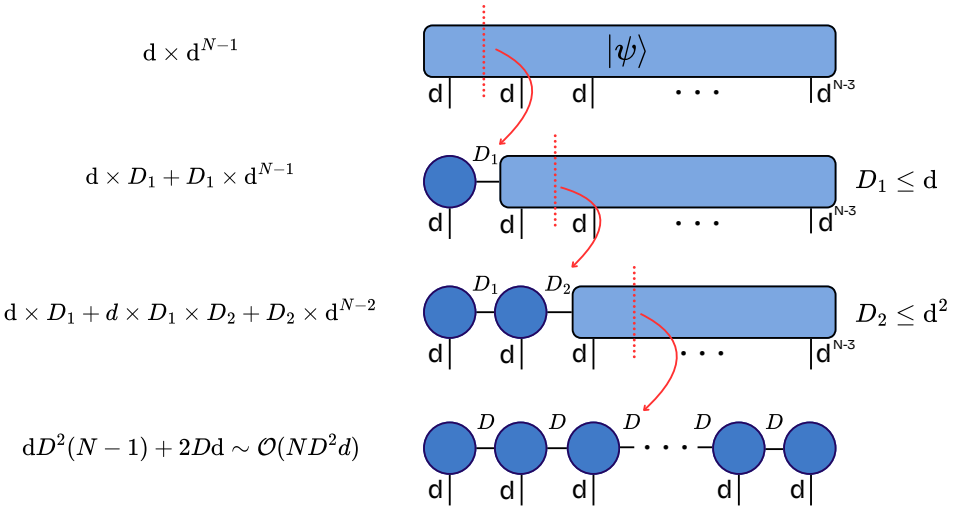


FIGURE 4.13: Schematic graphical representation of the decomposition of an exponentially large tensor $|\psi\rangle$ of dimension d^N into a Matrix Product State with total complexity NDd^2. The decomposition proceeds via a sequence of singular value decompositions, applied iteratively to reshape and factor $|\psi\rangle$. At each step, a high-rank tensor is split into two smaller tensors, and the bond dimension D controls the truncation and entanglement. The process results in a chain-like structure of tensors with each with one physical index d and two bond indices D. Here, the dimensions d and D are kept the same for each split.

4.4 Tensor Network ↔ Quantum Circuits

Every quantum circuit can be represented as a TN. To connect these two concepts, it is useful to remember the following definitions:

- Quantum gates can be rank-2 tensors with indices corresponding to input and output qubits (wires);
- Quantum states are tensors (e.g, $|0\rangle^{\otimes n}$)
- Measurements are contractions of tensors that create projections onto possible outcomes.

The connectivity of these tensors defines a circuit structure and the way entanglement is spread throughout the circuit. When contracting the full circuit (TN), the result can be: (1) a quantum state, (2) a scalar (probability amplitude) or (3) an expectation of an observable if we contract it with a measurement operator.

We will give an example of an exact and approximate mapping of Matrix Product State to a quantum circuit, following the explanation from Ref. [70, 71]. If MPS has a bond

dimension $\chi_{\max} = 2$ then it can be exactly mapped to a quantum circuit, as represented in Fig. 4.14(1), by building a ladder of two-qubit unitaries. This means that two tensors connected by a $\chi = 2$ are actually representing a two-qubit quantum gate. To extend this to higher bond dimensions, we map a $\chi_{\max} > 2$ to a multi-qubit gate acting on $\lceil \log_2(\chi) \rceil + 1$ qubits. This is not efficient when ported to a real quantum device, as multi-qubit gates need to be decomposed to smaller structures, thus, Ref. [71] studies how to decompose an MPS to a multi-layered ladder structure of two-qubit gates (see Fig. 4.14(2)), and uses an analytical and optimization-based approach (refer to these procedures in Ref. [71]).

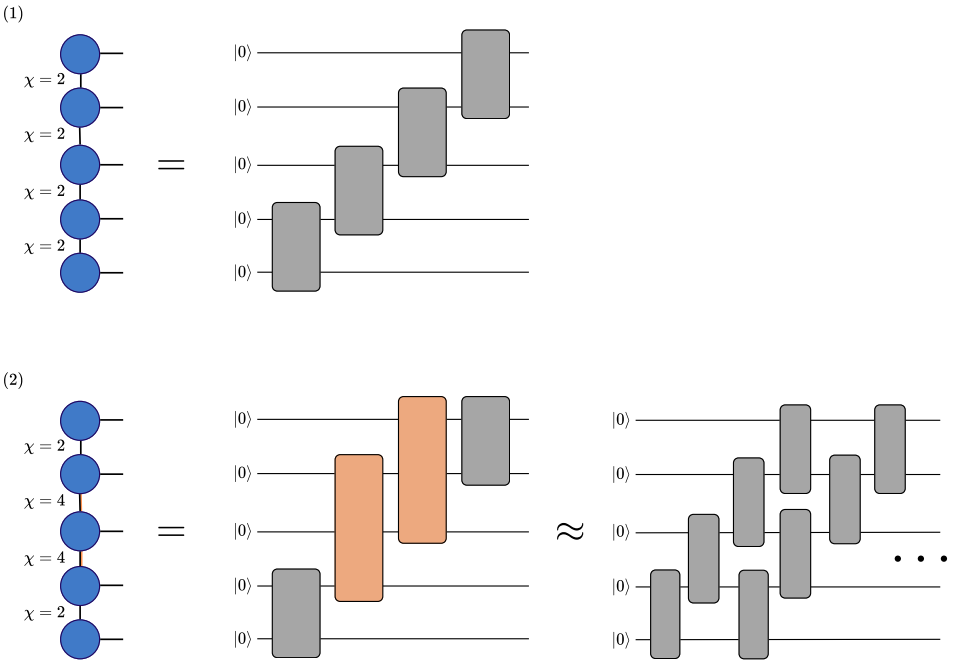


FIGURE 4.14: Illustrative representation of mapping of Matrix Product State to a quantum circuit. (1) Exact mapping is possible with bond dimension $\chi = 2$ to a ladder structure of two-qubit gates. (2) Approximate mapping is given for a higher bond dimension, where each bond dimension χ can be mapped to a multi-qubit gate acting on $\lceil \log_2(\chi) \rceil + 1$ qubits, which can be further decomposed to a multi-layered ladder structure.

This is important for many applications in quantum physics and quantum machine learning such as: encoding classical data in a quantum circuit [72], pre-training of a PQC with an MPS [70, 73], or simulating a quantum algorithm with an TN method and obtaining more accurate results than using noisy quantum hardware [74–77].

Chapter 5

Tensor Networks for Machine Learning

E. Puljak

I view every machine learning problem as a tensor decomposition waiting to happen.

In recent years, TNs have become a powerful tool of interest for machine learning, studied both in theoretical settings and practically for real-life applications. The goal of this thesis is to contribute to the practical side and to prove the usage of theoretical TN methods. Given their ability to efficiently represent large tensors, TNs offer a natural framework for compressing neural network (NN) layers, leading to a reduction in the number of parameters and computational resources [78–81]. Recent work has explored the integration of TNs into standard NN architectures [81, 82]. There is growing interest in exploiting this approach as an alternative to conventional network compression techniques such as layer fusion, quantization, and pruning. Due to their structured decompositions of quantum or data states, modular design, explicit control over entanglement, and physical motivation of the architectures, TNs prove strong analytical interpretability [83]. This property has been leveraged in various ML models such as Born machines [84–86], restricted Boltzmann machines [87], autoregressive networks [88], and privacy-preserving models [12]. They are also analyzed in many different learning setups, such as supervised learning [9, 10, 89–92]; unsupervised learning [85, 88] (e.g., anomaly detection [3, 16]), reinforcement learning [93, 94], and online learning [95]. Through intentional modeling of data correlations, we can introduce inductive bias to the TN design and create flexible structures that can enhance generative power and increase model efficiency.

Tensor Networks can be used in the context of Quantum Machine Learning, as discussed in Sec. 4.4. TNs can help more efficiently simulate quantum systems, serve as a starting point in quantum circuit optimization [70, 73], encode data into quantum circuits [72, 96] and offer quantum hardware-efficient structure [97–99].

Like classical ML models, TNs must learn from data through an appropriate optimization scheme. A natural approach for optimizing TN parameters is the DMRG-based technique, originating from condensed-matter physics (see Sec. 4.2.7). In ML context, this is often called the *Sweeping algorithm* [2, 9] and it is described in detail in the latter Sec. 6.2.3). In addition to the sweeping-based optimization, TN models can also be trained with *gradient-based* methods by differentiating through the TN structures. Both optimization strategies offer benefits and limitations, and the choice of method depends on the TN structure, the structure-dependent parameters and the specifics of the learning task.

In this chapter, we introduce the TN-based model for supervised learning, as well as probabilistic modeling with TNs, both of which have been applied to real-world use cases presented in this work.

5.1 Supervised Learning with Tensor Networks

In this section, we discuss a one-dimensional TN structure used for multi-classification as a form of supervised learning model. The implementation leverages the ability of TNs to represent a high-dimensional tensor (e.g., weight tensor) as a low-rank representation. In particular, the TN-based classifier is expressed as:

$$f_c(x) = \langle \Psi_c | \Phi(x) \rangle, \tag{5.1}$$

where Ψ_c is a TN (e.g., MPS) serving as a weight tensor, and $\Phi(x)$ is a nonlinear data embedding of input x in a higher-dimensional space. $\Phi(x)$ can embed the input data into a product space or another TN with bond dimension $\chi > 1$.

For a *product state embedding* [2], each feature of the input space x_i is separately mapped with a local feature map function ϕ_i to a local feature space with dimension d

(*physical dimension*). This yields a global feature map expressed as a tensor product:

$$\Phi(x) = \bigotimes_{i=1}^{n} \phi_i(x_i), \tag{5.2}$$

and visualized in Fig. 5.1.

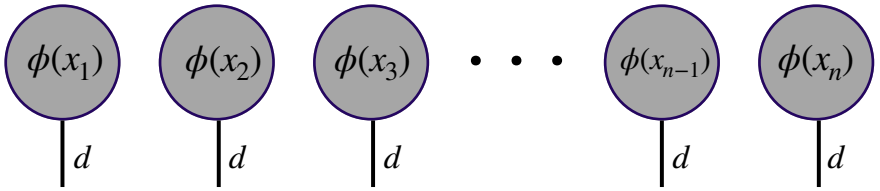


FIGURE 5.1: Graphical representation of a product state used for embedding input data with global feature map Φ, composed of a tensor product of local feature maps ϕ, which embed each input feature x_i into a vector with dimension d.

Each local feature map product needs to be normalized to unit norm to ensure the stability of the global mapping. The choice of local feature map is typically guided by the characteristics of the data domain or heuristic consideration, and it may be either identical or different for each individual feature x_i. These can be functions: the Fourier transform, Gaussian Radial Basis function (RBF), Polynomial function, etc. This embedding parallels the approach of kernel functions in support vector machines (SVMs), where nonlinear feature maps implicitly define inner products in high-dimensional spaces.

Another form of embedding creates an entangled state $\Phi(x) = |\Phi(x)\rangle$, expressed as a superposition of basis states with corresponding coefficients. This quantum state can be decomposed into an MPS, as defined in Sec. 4.4. Both types of embeddings are described in detail in the following chapter, where we present the `tn4ml` Python library.

Finally, to obtain class probabilities as outputs, the TN model Ψ_c is designed to have one output index with dimensionality corresponding to the number of classes N_C, and it is contracted along the physical indices with embedded data $\Phi(x)$. The resulting MPS is further contracted up to a vector of size N_C. This process is visualized in Fig. 5.2 for both types of data embedding, while both embeddings are closely explained in Fig. 6.2. The real-valued vector (red in Fig. 5.2) is passed through a softmax function to obtain the class probabilities.

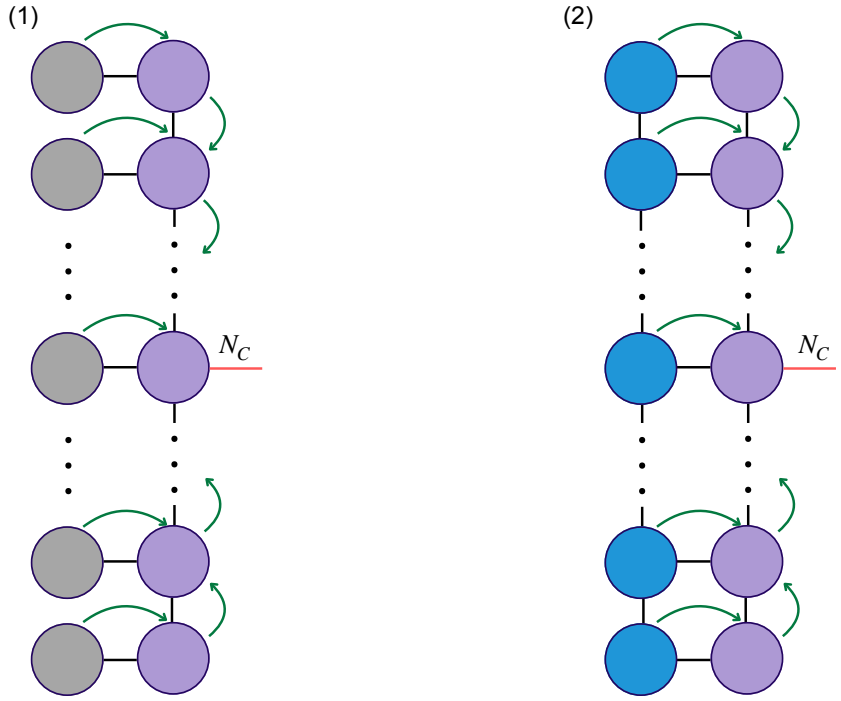


FIGURE 5.2: Graphical scheme of the contraction sequence for a N_C-class classification task using product state embedding (gray) or entangled state embedding (blue) and a parametrized Matrix Product state (purple) resulting in a real-valued vector with size corresponding to the number of classes (red).

5.2 Born Machine and Probabilistic Modeling with TNs

Born machine is a generative model using the laws of quantum mechanics to represent probability distributions, where the probability of observing an outcome x is given by the squared amplitude of that outcome from a quantum state representation of the whole data distribution $|\Psi(x)\rangle$ (i.e., *Born rule*). First, we introduce the approach applied to discrete data, then we extend it to the continuous data regime.

5.2.1 Discrete Data Regime

Given a discrete data setting, where each data point x is described with N binary or categorical random variables, we can represent it with a quantum state [3, 85]:

$$|\Psi\rangle = \sum_{x \in \tau} \Psi(x)|x\rangle, \tag{5.3}$$

where τ is a set of unique combinations of N random variables. $\Psi(x)$ is a coefficient corresponding to a unique configuration (outcome), and $|x\rangle$ is a basis state of the spanned Hilbert space. When we measure $|\Psi\rangle$, the outcome is defined as a tuple of observed values $x = (x_1, ..., x_N)$, where each x_i is the realization of the i-th random variable, with a probability given by the Born rule:

$$P(x) = \frac{1}{Z}|\Psi(x)|^2. \tag{5.4}$$

With this expression, the non-negativity condition $P(x) \geq 0$ is ensured. Z is a normalization constant, called also a *partition function*, that ensures the summation of probabilities to 1; $\sum_x P(x) = 1$. This said, we define the constant Z as:

$$Z = \sum_x |\Psi(x)|^2. \tag{5.5}$$

The computation of Z in a discrete regime is hard but possible exactly, as all unique possibilities of x are known. On the other hand, in the continuous regime, approximation techniques such as Monte Carlo sampling are needed.

To translate this to TNs, we can represent the amplitude (coefficient) of an outcome $\Psi(x)$, a rank-N tensor, using an MPS with N sites [2, 84, 100]. The decomposition through SVD is given by:

$$\Psi(x) = A^{(1)x_1} A^{(2)x_2} \cdots A^{(N)x_N}, \tag{5.6}$$

where each site $A^{(k)x_k}$ is a rank-3 tensor with two bond indices and one physical index with dimension d_k, except on the first and last sites where boundary conditions remove bond dimensions to the left (first) and right (last). For $\Psi(x)$ to represent a probability (scalar), we need to compute its norm, as previously explained in Sec. 4.2.6. An efficient representation with MPS makes it possible to reduce the exponential complexity of Z summation to $\mathcal{O}(Nd\chi^3)$, and even further if MPS is in a canonical form - $\mathcal{O}(N\chi^2)$. χ is the bond dimension and $d = \max_i d_i$ is the biggest physical dimension of the MPS.

Discrete-based probabilistic modeling with TNs enables efficient sampling, likelihood estimation and marginalization, with reduced computational complexity and scalable architectures. With a low-rank structure, these models can represent joint probability distributions without the need to store a full table.

5.2.2 Continuous Data Regime

To translate this framework to continuous data, we need to map each continuous random variable x_i to a discrete domain ($i \in [1, N]$). The approach for this is proposed by Ref. [86] and contains two parts:

1. **Global Feature Mapping**: A product state $\Phi(x)$ built as a tensor product of local feature maps $\phi_i(x_i)$ that project each feature from a continuous to a discrete D-dimensional vector space ($\phi_i : \mathbb{R} \to \mathbb{K}^D$).

2. **Matrix Product State**: A parametrized TN model $|\Psi\rangle$ with N sites, with unit norm, learning the probability distribution.

To build a span of the space $\mathbb{K}^D$, we need to define a set of D feature functions for each ϕ_i such that each $\{f_j\}_{j=1}^{D}$ is a projection $f_j : \mathbb{R} \to \mathbb{K}$ and orthonormal basis of space $\mathbb{K}^D$. The important requirement of these feature maps is an *isometric*[1] condition, which keeps the same distance between points in different vector spaces, or in other words, preserves inner products between vectors during the transformations from one space to another.

Ref. [86] presents a proof of universal approximation of any probability density function (that ensures the convergence of expansion) through the continuous-valued MPS model, visualized in Fig. 5.3

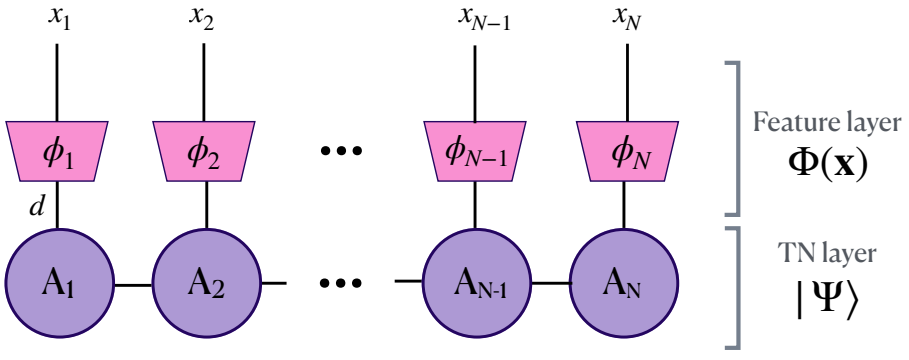


FIGURE 5.3: **Continuous-valued MPS model** as a density estimator consisting of an embedded product state $|\Phi(x)\rangle$ (feature layer) and an MPS model $|\Psi\rangle$ with trainable parameters A_k (TN layer) that learn a probability distribution.

We can express the amplitude of the probability density function as:

$$\theta(\mathbf{x}) = \langle \Phi(\mathbf{x}) | \Psi \rangle. \tag{5.7}$$

[1] $\langle \phi(x), \phi(y) \rangle = \langle x, y \rangle$ for all $x, y \in \mathbb{R}^n$

To obtain the probability distribution, we calculate the squared magnitude of $|\theta(\mathbf{x})|^2$:

$$P(x) = |\langle \Phi(\mathbf{x}) | \Psi \rangle|^2. \tag{5.8}$$

From this expression, we satisfied the non-negativity and the normalization condition for PDFs ($\int_{-\infty}^{\infty} P(\mathbf{x})dx = 1$), which is satisfied from the isometric condition. It should be noted that for PDFs, it is possible that for some $x \in \mathbb{R}$ to have $P(x) > 1$.

5.3 Explainability of One-Dimensional Structures

Matrix Product States are known by their transparent and explainable modeling framework that comes from their quantum-inspired structure. Namely, entanglement entropy, such as the von Neumann entropy, is a natural measure of information flow and correlation structure within the high-dimensional data [101]. Opposed to traditional NNs, which are often called "black-box" models, the MPS structure provides direct access to mutual information, feature-wise probabilities, and interpretable entropy-related metrics, offering a transparent view into a model's learning process. Another important aspect of the model's architecture is its expressivity, modeled by the bond dimension, where higher values can capture richer correlation patterns at the cost of higher computational complexity. Additionally, this tradeoff can be moderated by the area-law scaling of the entanglement entropy, which can be useful for many practical applications as an implicit regularizer [102]. Recent theoretical works have demonstrated the universality of MPS form, proving they can approximate arbitrary Boolean functions and form a dense subset of continuous functions under specific conditions [86, 87]. This further confirms their versatility across classical and quantum applications.

Chapter 6

`tn4ml`: A Python Software to Train and Customize Tensor Networks for Machine Learning Applications

The recent development of Tensor Networks as machine learning models has motivated their practical implementation and the creation of a user-friendly, efficient, and computationally lightweight framework. To research the applicability of TNs for a given ML task, we need to build a training-evaluation pipeline and conduct a thorough benchmarking procedure as one would do in any existing framework for conventional ML models. Motivated by this, we developed a Python library, `tn4ml` (**T**ensor **N**etworks for **M**achine **L**earning), that creates a TN-based ML pipeline with a user-friendly interface, easily accessible to any student, researcher, or ML practitioner. The library contains modules for data embedding, TN architectures (currently supporting 1D structures), initialization functions, optimization strategies, optimization metrics, and evaluation methods. Similar to other ML frameworks, the library supports a training and evaluation pipeline structure. To showcase the library's potential and usage, we present two examples, for supervised and unsupervised learning on different types of data. The library is also described in Ref. [2] and available at github.com/bsc-quantic/tn4ml.

6.1 Design Philosophy and Goals

The goal of the tn4ml library is to smoothly implement an ML pipeline for TN-based models applied to arbitrary learning problems. The ML pipeline consists of the following steps:

1. **Data Embedding**: A step involving a transformation of input data samples into a TN-based format, either product state or entangled state, resulting in an embedded data structure Φ.

2. **Model Choice and Initialization**: This step is driven by the definition of the learning model, which influences the choice of TN architecture and initialization technique.

3. **Optimization**: This phase begins with defining an objective function $\mathcal{L}$ and continues by choosing an optimization strategy to steer the learning process.

4. **Evaluation**: The last step of the pipeline involves selecting the proper evaluation functions and the techniques to visually present the results.

Fig. 6.1 illustrates the TN-based ML pipeline, visualizing each part discussed above.

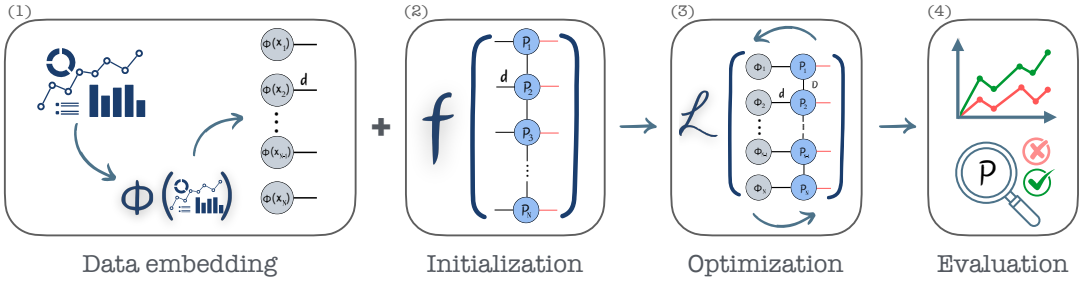


FIGURE 6.1: The ML pipeline for a parametrized TN-based model P includes: (1) data embedding process, with Φ as embedding function; (2) TN structure design and choice of initialization technique f; (3) optimization procedure with objective function $\mathcal{L}$ and (4) evaluation process.

In the following sections, we describe each part in more detail and provide supporting motivation. Features of the library that are not explained here are available in the library's documentation tn4ml.readthedocs.io.

6.2 Modules: Description and Code Implementation

The library is built using `quimb` [27] for Tensor Network structure representation and functionalities, and `JAX` [103] for optimization scheme using automatic differentiation and just-in-time compilation.

6.2.1 Data Embeddings

A general idea of the Tensor Network model for an ML task models a function:

$$f(x) = W\Phi(x), \tag{6.1}$$

where W is a low-rank representation of a high-dimensional weight tensor, and $\Phi(x)$ is a data embedding procedure. The only nonlinearity comes from the embedding $\Phi(x)$, which is essential for uncovering complex correlations. The data embedding procedure is important for creating a bridge between input data and the TN model. It keeps the important properties of the data and transforms it into a regularized vector space, to a format suitable for further effective analysis and exploration by the TN-based model. The choice of the embedding procedure is motivated by the learning problem and the overall desired outcome. This part of the pipeline is the most important step, as it drives every following choice in pipeline design and steers the optimization in a specific direction. In our library, there are two choices of embedding procedures: *Product State Embedding* - creates a tensor product of local feature maps; and *Entangled State Embedding* - creates a high-order tensor decomposed to an MPS.

Product State Embedding

As already described in Sec. 5.1, each feature x_i from input sample $x = [x_1, x_2, ..., x_n]$ is mapped with a local feature map $\phi_i(x_i)$ into a high-dimensional space, creating a tensor product $\Phi(x)$, noted as a global feature map $\Phi(x)$ and expressed in Eq. (5.2). In Fig. 6.2(1), we illustrate this embedding visually, where each local tensor has an embedding dimension $d \in \mathbb{N}$ and no connecting indices between neighbors - known as a *product*

state. Numerical stability of the whole mapping is ensured by the unit norm of ϕ_i. Our library supports several functions that can be used as local feature maps[1]:

- **Trigonometric** function of dimensionality $2k$ transforms a real-valued feature into a $2k$-dimensional vector using sine and cosine functions, mapping $\mathbb{R} \to \mathbb{R}^{2k}$. The expressiveness of the feature map is controlled by the parameter k.

$$\phi(x) = \frac{1}{\sqrt{k}} \left(\cos\left(\frac{\pi}{2}x\right), \sin\left(\frac{\pi}{2}x\right), \ldots, \cos\left(\frac{\pi}{2^k}x\right), \sin\left(\frac{\pi}{2^k}x\right) \right)^\top. \tag{6.2}$$

  This embedding introduces periodic components, thus capturing cyclic patterns in the data and enhancing orthogonality among features [9].

- **Fourier** function creates a discrete periodic delta function with a finite Fourier series, with period $\frac{p}{p-1}$ and mapping $\mathbb{R} \to \mathbb{R}^p$ to a one-hot-like vector.

$$\phi(x_j) = \frac{1}{p} \left| \sum_{k=0}^{p-1} \exp^{2\pi i k \left(\frac{p-1}{p}x - \frac{j}{p}\right)} \right|. \tag{6.3}$$

  This approach can capture more complex periodic structures. Its implementation was taken from Ref. [16]:

- **Gaussian Radial Basis Function**(RBF) is commonly used to encode non-periodic features due to its ability to create a smooth interpolation across the feature space. This non-linear transformation captures the similarity of data points. Its mathematical expression is:

$$\phi(x) = \exp^{\gamma ||x - x_c||^2}, \tag{6.4}$$

  where $||\cdot||$ is the Euclidian (L2) distance, x_c represents the center of the basis function, and γ is the scaling parameter. The values of x_c and γ are either chosen based on a specific heuristic, or motivated by the distribution and scale of input features. The resulting embedding dimensionality d is determined by the number of centers.

- **Polynomial** function projects input features into a higher-dimensional space by computing polynomials up to a degree d. Optionally, a bias term C (typically set to 1.0) can be included. This transformation is very effective when modeling non-linear

[1]For all feature functions, x denotes a single input feature, except for the Fourier function, where j is the index of the feature in the input vector $\vec{x}$, thus $x \to x_j$.

relationships and uncovering complex relations among features.

$$\phi(x) = \left(C, x, x^2, \ldots, x^d\right)^\top \tag{6.5}$$

- **Legendre polynomial** function describes polynomials of increasing degree $n \in \mathbb{N}_0$ defined on the interval $[-1, 1]$ which ensures orthogonality and stable numerical representations. When isometrized (normalized or scaled to form an orthonormal basis), they are proportional to the classical Legendre polynomials [86].

$$P_n(x) = \frac{1}{2^n n!} \frac{d^n}{dx^n} \left((x^2 - 1)^n\right) \tag{6.6}$$

- **Laguerre polynomial** function describes polynomials of increasing degree $n \in \mathbb{N}_0$ multiplied by an exponential factor $e^{-x/2}$ and defined on interval $x \geq 0$ for $x \in \mathbb{R}$.

$$L_n(x) = e^{-x/2} \sum_{k=0}^{n} \frac{(-1)^k}{k!} \binom{n}{k} x^k, \tag{6.7}$$

- **Hermite polynomial** function creates polynomials of increasing degree $n \in \mathbb{N}_0$ multiplied by the Gaussian function $e^{-x^2/2}$ and defined on all $\mathbb{R}$. Isometrization is ensured by the weight $e^{-x^2/2}$.

$$H_n(x) = (-1)^n e^{x^2} \frac{d^n}{dx^n} \left(e^{-x^2}\right) \tag{6.8}$$

The library also supports the formation of a *Complex Embedding* class, which allows the use of custom embedding functions for each feature.

Entangled State Embedding

To define an *Entangled State* embedding, all features x are mapped to a quantum state $|\Phi(x)\rangle$, and embedding is defined as:

$$\Phi(x) = |\Psi(x)\rangle \tag{6.9}$$

$$= \sum_{i_1, i_2, \ldots, i_n} C_{i_1, i_2, \ldots, i_n}(x) \, |i_1\rangle \otimes |i_2\rangle \otimes \cdots \otimes |i_n\rangle, \tag{6.10}$$

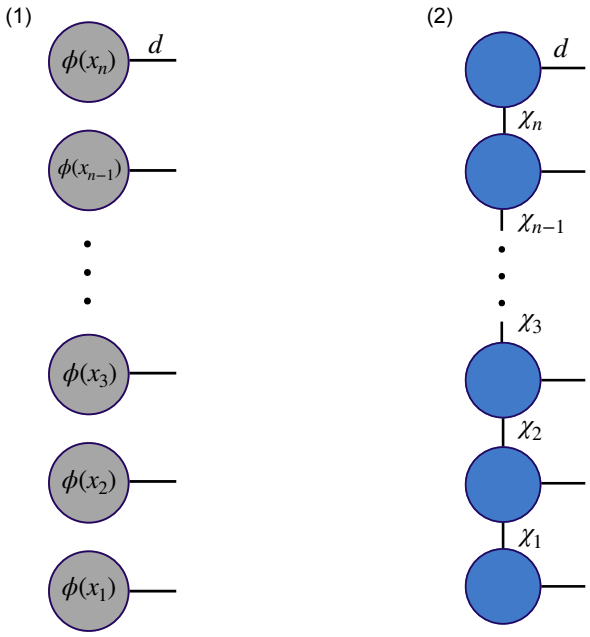


FIGURE 6.2: Format of data embedding final state after applying (1) Product State or (2) Entangled State Embedding. $\phi(x_i)$ is the feature map transforming a component of the input feature x. d is the physical dimension of the embedding, which can be different for each tensor. χ_i is the bond dimension between two neighboring tensors. n is the number of tensors in a product state or a matrix product state representation of the embedded input.

where coefficients $C_i(x)$ depend on input x. This definition is followed by a decomposition into an MPS representation [71, 104], where users can define their own decomposition strategy, or use the default one implemented in the library. The final entangled MPS is visually presented in Fig. 6.2(2).

The entangled embedding from Ref. [96], called *Patch Embedding*, is currently implemented in the library, and it serves as a quantum encoding of classical images using the flexible representation of quantum images (FRQI). The quantum state of the entire image is expressed as:

$$|\psi\rangle = \frac{1}{\sqrt{N}} \sum_{i=0}^{N-1} |i\rangle \left(\cos\left(\frac{\pi p_i}{2}\right) |0\rangle + \sin\left(\frac{\pi p_i}{2}\right) |1\rangle \right), \tag{6.11}$$

where N is the total number of pixels, $|i\rangle$ denotes computational basis states represented as binary strings that indicate positions of pixels in the flattened N-dimensional image array, and p_i is a pixel value at position i.

6.2.2 Model Choice and Initialization

Our library currently supports 1D structures, such as MPS, MPO, and SMPO, theoretically explained in Sec. 4.2.1. While multidimensional structures are not yet supported, the restriction to 1D structures is not a significant limitation, as 1D TNs are the most extensively studied and widely applied, and many practical problems can be transformed into a 1D structure. We describe which parameters are needed to define such models in practice. First, we introduce supported initialization methods, followed by descriptions of individual parameters to the TN object class.

Initialization

Sometimes the TN optimization, stability and efficiency can be directly influenced by the choice of initialization technique (see the presented results in Sec. 6.4.2 and Chapter 9). Issues like slow convergence, vanishing, or exploding gradients can be caused by ineffective initialization methods. Our library supports any functions from `jax.nn.initializers` and implements the following methods:

- **Gram-Schmidt orthogonalization** [105] is a known method for establishing orthonormal basis sets in vector spaces. Here, we explain its implementation applied to an MPS. Each tensor for an MPS typically contains up to three dimensions. By fixing the first dimension of a tensor as a principal index, and collapsing the remaining dimensions to a single, combined index, we effectively create a matrix representation, where each row is a distinct vector. These row vectors are passed through the Gram-Schmidt orthogonalization algorithm, generating a set of orthogonal vectors, and are additionally normalized. The resulting data arrays are reshaped back to the original tensor dimensions. This method preserves the TN's norm, thus ensuring numerical stability of tensor operations and contractions. The initial configuration of tensor elements can either be randomly sampled from Gaussian or uniform probability distributions.

- **Random normal initialization** function defines tensor values as random variables drawn from a Gaussian distribution with mean μ and standard deviation σ. Optionally, a Gaussian noise can be added to the tensor values, with different μ_{noise} and σ_{noise}.

- **Unitary initialization** method is structuring tensor values as a collection of random unitary matrices, typically sampled according to the Haar measure to ensure uniformity across the unitary group [25, 106]. First, random values from a normal distribution are generated to form a matrix, followed by the QR decomposition and phase correction. By default, unitary matrices preserve vector norms, which supports stable and efficient training and reduces the risk of gradient problems.

- **Zeros or Ones** function is initializing tensor values either to all zeroes or all ones with the option of adding a Gaussian noise centered at 0 with standard deviation σ_{noise}.

- **Adding an identity matrix to a tensor's diagonal values** is an option to ensure higher stability in some cases, by keeping the tensor values close to identity (implementation from Ref. [25]).

Model class The parameters needed to instantiate TN model classes supported in the library, such as a general `TensorNetwork`, `MatrixProductState`, `MatrixProductOperator`, and `SpacedMatrixProductOperator`, are presented with names and a short description in Table 6.1.

An additional feature in the SMPO model, introduced in Ref. [16], is the ability to specify a varying spacing parameter S, which enables a free choice of output indices.
For the SMPO model, contraction paths are fixed for two contraction cases:

- **SMPO - SMPO/MPO**: contraction path is fixed by contracting tensors with the same name tags, adopted from `quimb` [27].

- **SMPO - MPS**: The operator (SMPO) and MPS are merged and contracted over shared physical indices (i.e., outer indices of the MPS), respecting spacing between outputs and ensuring a valid result state in the format of an MPS.

The definition of arbitrary `TensorNetwork` is currently only useful for defining an MPS state for classification, as the optimization of an arbitrary TN is not supported.

Parameter	Description
L	Number of tensors.
initializer	Type of tensor initialization function.
key	PRNG key used to generate random numbers for initialization.
dtype	Data type of the tensors (e.g., jax.numpy.float_).
bond_dim	Dimension of virtual indices between tensors.
phys_dim	Dimension of physical index for each tensor. A tuple for (input, output) indices if a model is an MPO/SMPO.
shape_method	Method to generate shapes for tensors. If value is *'even'* then values are generated equally for each tensor (bond_dim, bond_dim, *phys_dim) without no compression.
cyclic	Indicates if a model is cyclic.
add_identity	Whether to add identity to tensors' diagonal elements.
add_to_output	Whether to add identity to diagonal elements of tensors with output indices.
boundary	Boundary condition ('obc' = open, 'pbc' = periodic).
class_index	Index of tensor that serves as output node for class (classification tasks only, MPS only).
class_dim	Dimension of output node or number of classes for classification (classification tasks only, MPS only).
compress	Whether to truncate bond dimensions.
insert	Index of tensor used for norm division. If None, distribute norm across all tensors.
canonical_center	If not None, create canonical form centered around this index.
spacing	Spacing parameter, or space between output indices in number of sites. Only when spacing parameter is fixed (SMPO models only).
output_inds	Indexes of tensors which have output indices. From 0 to L. If spacing is varied across the network (SMPO only).

TABLE 6.1: List of parameters used for initializing a tensor network model.

6.2.3 Optimization

For a machine learning optimization problem, it is important to define an objective function describing the learning problem at hand and an optimization strategy to steer the optimization in the right direction. Firstly, we want to highlight the motivation behind using JAX as an optimization package in Python. JAX is known for its powerful features like automatic differentiation and just-in-time compilation, which ensure an efficient numerical backend. Libraries like Optax and Flax are sub-packages from JAX offering useful

functions like early stopping, different optimizers (e.g., Adam, AdaDelta, RMSProp) used for gradient updates, and gradient clipping. Additionally, it is possible to choose a device (CPU or GPU) for running the optimization pipeline.

Objective functions supported in the library are defined in the Table 6.2. Each function has to implement a set of TN contractions before any further calculation, to obtain either a scalar or vector value. For cases like classification, the result of a contraction is a real-valued vector, referred to as y_{pred}, which is further given to the softmax function to obtain class probabilities.

Name	**Formula**	**Typical Use**
LogQuadNorm	$\frac{1}{N}\sum_{i=1}^{N}\left(\log\|\|P\Phi(x_i)\|\|_2^2 - 1\right)^2$	Learning problems where the MPO or SMPO transforms embedded input $\Phi(x)$.
NegLogLikelihood	$-\frac{1}{N}\sum_{i=1}^{N}\log(\|P\Phi(x_i)\|^2)$	Probabilistic and generative modeling, unsupervised or semi-supervised learning.
Cross-Entropy Softmax	$-\sum_{i=1}^{N} y_{\text{true}_i}\log(\text{softmax}(y_{\text{pred}_i}))$	Classification tasks minimizing the difference between predicted and true class labels.
MeanSquaredError	$\frac{1}{N}\sum_{i=1}^{N}(y_{\text{pred},i} - y_{\text{true},i})^2$	Regression tasks minimizing the error between continuous outputs and true values.
OptaxWrapper	_____	Wraps TN contractions into inputs for `Optax` loss functions, for general-purpose use.
LogNorm	$\log(\|\|P\|\|_F^2)$ or $\text{ReLU}(\log(\|\|P\|\|_F^2))$	Regularization to prevent overfitting and improve generalization.

TABLE 6.2: Overview of the objective functions. N denotes number of samples in a dataset; P is a parametrized TN model; y_{true} and y_{pred} are true and predicted class labels; and F is a Frobenius norm.

Our library supports two types of **Training Strategies**, both gradient-based, suitable for optimizing TN structures. The choice of an appropriate technique is driven by many factors, and here we describe the benefits and caveats of each.

- **Sweeping method**, originated from the DMRG algorithm (see Sec. 4.2.7) and introduced for ML practices in Ref. [9], developed as an effective optimization method for 1D TN structures. The method includes contracting two neighboring tensors, computing the gradient with respect to the combined tensor, updating its value, and then performing an SVD to split it back into two tensors. This procedure is applied sequentially for each tensor along the chain in both the forward and backward directions. Gradients are computed using automatic differentiation, ensuring that the contraction path is fixed to provide a deterministic sequence of summations and multiplications. This deterministic structure makes it possible to track how the final output depends on each operation. As detailed in Sec. 4.2.3, the gradient is obtained by removing a specific tensor from the TN, and keeping its surrounding environment. One limitation of this method is the requirement of storing optimizer state for each pair of tensors that keeps track of the accumulating gradients. This results in $2N\times$ (batch size) memory consumption. However, this technique offers a dynamic choice of bond dimension at each SVD step and benefits from robust optimization that avoids issues such as exploding or vanishing gradients [107].

- **Mini-Batch Gradient Descent** is employed as a primary optimization strategy, due to wide adoption across machine learning applications. Our implementation uses automatic differentiation to determine the gradients for each tensor within a TN. Tensor updates are occurring after processing a mini-batch using a chosen optimizer from `JAX` in a backward phase. The training procedure can be accelerated using multithreaded CPU processes or through a GPU runtime.

To increase the training speed, we vectorize the loss calculation across batch size, using `vmap` function in `JAX`. To keep numerical stability during the training procedure, optionally, canonicalization or normalization can be performed at the beginning of the training or after each epoch.

6.2.4 Evaluation

To evaluate the model's performance and understand its results, we need to choose the correct evaluation metrics and methods. `tn4ml` offers multiple choices for evaluation metrics and functions for visually presenting their results. For supervised learning tasks,

such as classification, the performance is assessed by comparing predicted labels with true labeled data with metrics such as accuracy, precision, recall and the Receiver Operating Characteristic (ROC) curve. For unsupervised learning, one can analyze new, unseen patterns by comparing probability distributions, evaluate clustering quality by comparing distances to cluster centers, or use other domain-specific methods. Functions in our library can be used to replicate our results presented in the following Sec. 6.4, and in other chapters throughout this thesis. Additionally, a user can define their metrics and customize existing ones to cover the analysis of any ML task out of scope from our examples.

6.3 Comparison with Existing Libraries

At the intersection of Tensor Networks and Machine Learning, various libraries aim to develop ML pipelines for TN-based models and implement their optimization efficiently. Some libraries are built with `PyTorch`, a known ML framework, as backend, such as - `TorchMPS` [24], `tntorch` [108], `TensorKrowch` [25], while `TensorNetwork` [28] supports multiple backends. Other libraries are powerful tools for various TN-based algorithms and operations, but are not focused on ML applications, such as `quimb` [27], `TenPy` [26], `cirkit` [109] and `qtealeaves` [110]. Currently, a lot of development has been done in the Julia language, which offers libraries such as `Tenet.jl` [32], `ITensors.jl` [29, 30] and `TensorKit.jl` [31]. Each of these libraries has its own benefits and limitations, and together they are showing the promising progress of applications with TNs. The biggest shortcoming of Python libraries is on a practical aspect: they lack a user-friendly interface, similar to known ML frameworks such as `keras` or `JAX`. Additionally, we wanted to avoid reinventing a new interface for TN objects. Motivated by these reasons, the `tn4ml` is built on top of `quimb`, and leverages powerful just-in-time compilation in `JAX`.

6.4 Practical Examples

To showcase the features of the library, we collected dedicated examples and tutorials in our `GitHub` repository github.com/bsc-quantic/tn4ml. Here, we present two examples: supervised learning as a binary classification task, and unsupervised anomaly detection based on the task from Ref. [16]. For the classification task, we assess the TN performance

with accuracy and execution time on different devices. For the anomaly detection task, we consider three metrics derived from the ROC curve - the Area Under the ROC curve (AUC), True Positive Rate (TPR) at 1% False Positive Rate (FPR), FPR at 95% TPR.

6.4.1 Supervised Learning

Dataset Description

To showcase a supervised learning setup with TNs, specifically a binary classification problem, we use the *Breast Cancer* dataset available through Kaggle [111, 112], which contains numerical features obtained from digitized breast tumor biopsy images. Each data point is either malignant or benign, and it contains 30 features quantifying aspects of the cell nuclei visible under the microscope. These features include measurements such as radius, perimeter, area, texture, smoothness, symmetry, fractal dimension, compactness, concavity, and concave points. Each feature, scaled within the range $[0, 1]$ has three types of statistical categories: the mean, standard error, and the worst (maximum) recorded value. An imbalance in the training dataset leads to assigning a class weight to each sample. This is not used in validation/testing.

Implementation Pipeline

A Matrix Product State (MPS) model is used for the binary classification task. The parametrized MPS (P) has 30 tensors - one for each input feature - and an output index at the center of the chain with dimensionality corresponding to the number of target classes, in this case two. Model parameters are initialized using a Gaussian distribution with a standard deviation of 10^{-1}. The input features are embedded with a product state embedding and a polynomial feature function with a degree of two and a bias term of 1.0, resulting in a product state with physical dimension $d = 3$. Contraction of this embedded state with the parameterized MPS (as illustrated in Fig. 5.2(1)) produces a real-valued vector which is further passed through a softmax function to compute class probabilities.

Results

We study how model complexity, governed by the bond dimension, affects the classification accuracy and computational performance on the CPU/GPU platform. The primary goal of these experiments is not to achieve state-of-the-art accuracy, but to show efficiency

trade-offs and different evaluation metrics. The following experiments are conducted using a training set of 364 samples and a testing set of 91 samples, processed in a batch size of 16. The bond dimension ranges from 2 to 400. All experiments were carried out either on Intel Core i5-9600KF using six cores or on a single NVIDIA GeForce RTX 2080 Ti. The classification accuracy results can be compared with other benchmarks reported in the Kaggle competition. The computational efficiency on a device is assessed through the training and inference time, measured per vectorized batch of 100 samples, with time expressed in milliseconds (ms). Fig. 6.3 summarizes the execution time comparison between CPU and GPU devices, along with a small subplot showing classification accuracy. GPU runtimes remain nearly constant for both training and inference across increasing bond dimensions, while CPU execution time grows rapidly with bond dimension. The inference time plot highlights that GPUs become more efficient than CPUs at higher bond dimensions, while CPUs are preferable for lower bond dimensions. This trend is also observed in training time, where GPU execution is slower at lower model sizes.

In terms of classification performance, accuracy improves as the bond dimension increases from 2 to around 50. The highest observed accuracy is 97.3%, which follows results from traditional ML models on the same dataset ($\sim 98.5\%$). Accuracy plateaus or slightly decreases for larger bond dimensions, most likely due to overfitting. Interesting gap of less than 1% in performance between CPU and GPU at the bond dimension 400 might be the result of different tensor contraction implementations, but this was not investigated further. Overall, our results suggest that the bond dimensions between 20 and 50 yield the best result in terms of the tradeoff between accuracy and efficiency.

6.4.2 Unsupervised Learning

Dataset Description

To demonstrate unsupervised learning with TNs, we utilize the grayscale images of handwritten digits (0 to 9) from the Modified National Institute of Standards and Technology (MNIST) dataset [113]. Each event consists of an image of a resolution of 28×28 and a corresponding label. There are 60,000 train samples and 10,000 test samples. As our study follows the results from Ref. [16], the pixel values are scaled to range $[0, 1]$ and the resolution is reduced to 14×14 with the bilinear interpolation method.

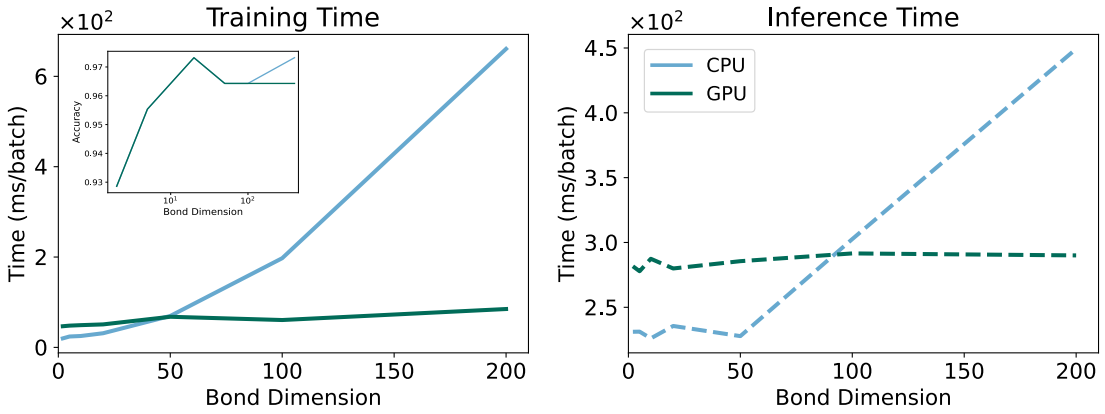


FIGURE 6.3: Comparison of training and inference execution times for each device (GPU, CPU) as a function of increasing bond dimension. Time is expressed in milliseconds (ms) per batch size of 100 samples. A small subplot inside the first plot demonstrates an increase in accuracy with the growth of bond dimension, with the best value of 0.973.

Implementation Pipeline In this study, our goal is to implement a simple task of anomaly detection, where the training process happens on one class of images, called *background*, and is tested on all others. To flatten the image to a one-dimensional vector, pixels are ordered in a *zig-zag* fashion, as illustrated in Fig. 6.4. The first row is mapped to the first X elements, the second row to the second X elements, etc. The 1D input x is embedded into a product state $\Phi(x)$ with physical dimension $d = 2$ using the trigonometric feature function.

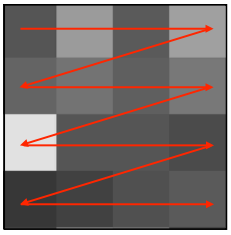


FIGURE 6.4: The *zig-zag* flattening operation of the image to a 1D vector. As indicated by the red arrows, the first row with 4 pixels is positioned at the beginning of the 1D vector. Then the second row is mapped to the next 4 vector elements, etc.

The SMPO (see Sec. 4.2.1) is used for anomaly detection, with the goal of learning to project anomalies close to the origin of the hypersphere in the operating space, while keeping normal instances closer to the surface. The design of the objective function is introduced in Ref. [16] and defined as the norm of the transformed input vector $D(x) = ||P|\Phi(x)\rangle||_2^2$, where P is the SMPO model (see Fig. 6.5). The model is trained with the mini-batch gradient descent method.

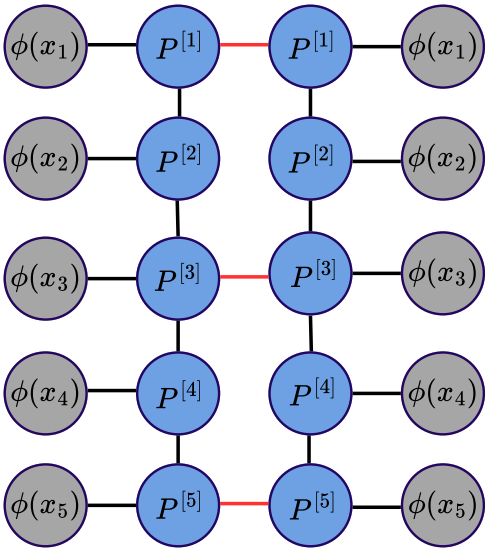


FIGURE 6.5: Graphical representation of the objective function for anomaly detection assuming the number of tensors is five. Embedded input vector $\Phi(x) = \bigotimes_{i=1}^{n} \phi_i(x_i)$ is transformed with the Spaced Matrix Product Operator model (P). The scalar value of the loss is obtained by full contraction of these tensor network structures.

Results We conduct an analysis of performance under varying hyperparameter configurations: (1) bond dimension $\chi \in \{5,\ 10,\ 30,\ 50\}$, (2) spacing parameter $S \in \{4,\ 8,\ 16,\ 32,\ 64\}$, and (3) the initialization method. Different initialization methods we study are: Glorot normal, He or Kaiming normal, orthogonal (from `JAX`), Gram-Schmidt orthogonalization, and random normal initialization. We choose to individually study classes 0, 3, and 4 as normal samples, while others are considered anomalies. We use metrics such as AUC, TPR at 1% FPR and FPR at fixed 95% TPR. In terms of the AUC and TPR, the higher value is better, but for the FPR, the lower value yields to better performance. First, we study the performance under a change of bond dimension for each initialization technique, while fixing the spacing parameter $S = 16$. As demonstrated in Fig. 6.6, the choice of initialization technique can sometimes influence the performance. Some initialization techniques need a larger bond dimension to achieve better results, while others overfit with a larger space. Second, we analyze the performance when changing the spacing parameter, while fixing the bond dimension to 10. Fig. 6.7. Across all metrics, we observe there is a *sweet-spot* in performance, with values $S = [16,\ 32]$ as best performing for all three classes. With these results, the goal is not to find the best technique, but to show how varying the hyperparameters can impact the overall performance.

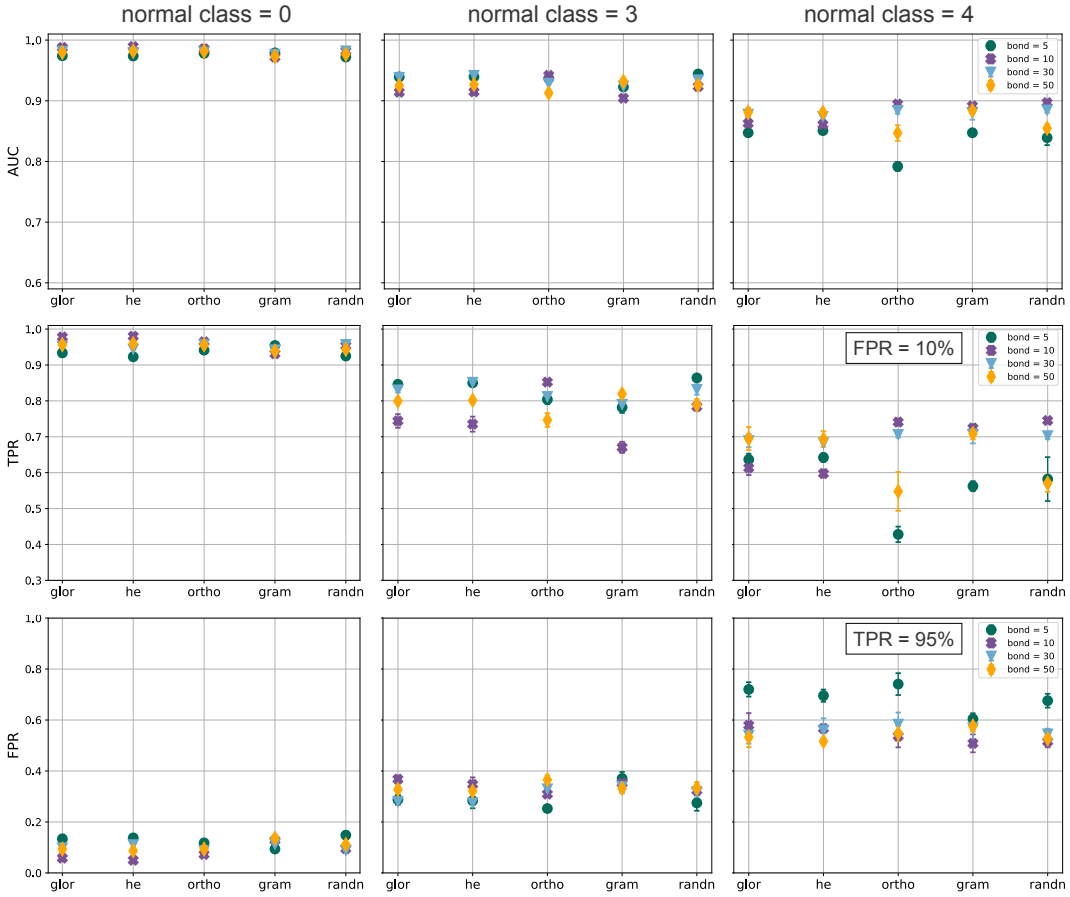


FIGURE 6.6: Anomaly detection task on MNIST dataset across (top) AUC, (middle) TPR at a fixed FPR (10%), and (bottom) FPR at a fixed TPR (95%). Classes 0, 3, or 4, are used for training as normal samples, while all other classes are used as anomalous instances for testing. Hyperparameters used to compare performances are: (1) bond dimension $\chi \in \{5, 10, 30, 50\}$) and (2) initialization technique (*glor* - Glorot normal, *he* - He normal, *orto* - orthogonal, *gram* - Gram-Schmidt orthogonalization, *randn* - random normal).

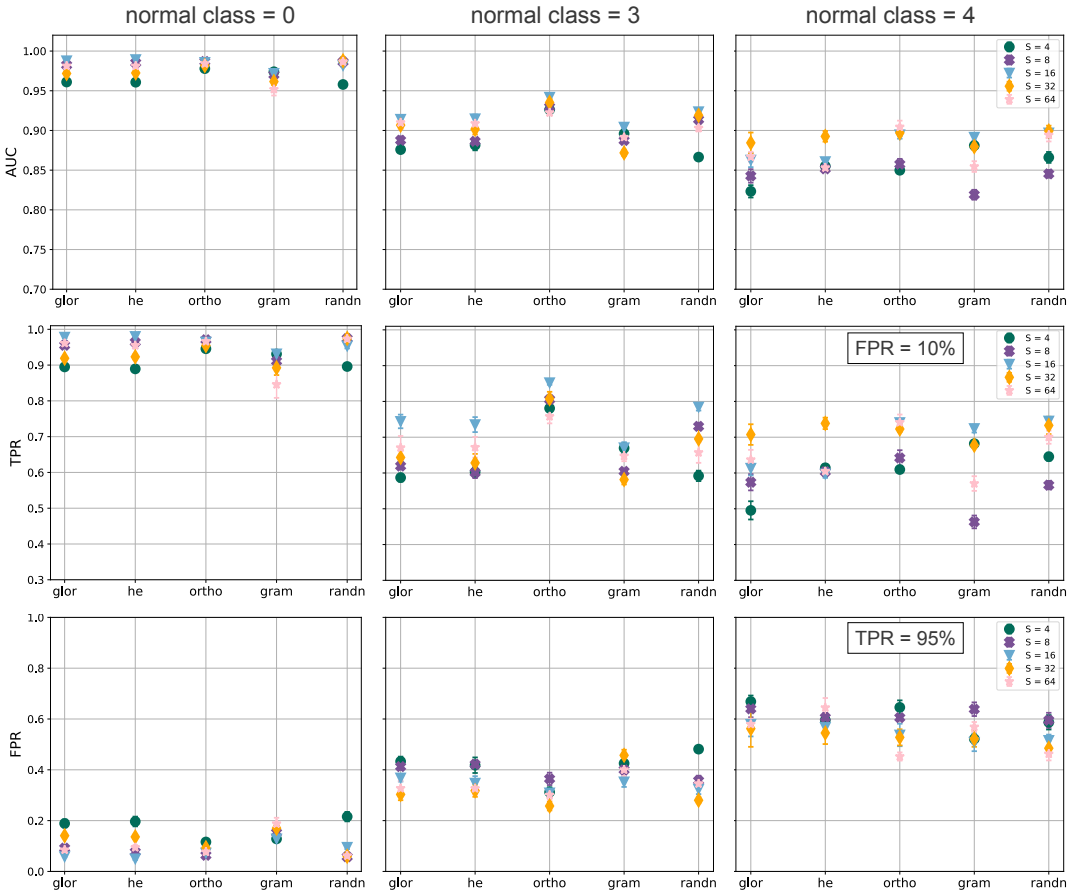


FIGURE 6.7: Anomaly detection task on MNIST dataset across (top) AUC, (middle) TPR at a fixed FPR (10%), and (bottom) FPR at a fixed TPR (95%). The training dataset, defining the normal samples, is obtained by combining "0", "3" and "4" classes. All other classes are used as anomalous instances for testing. Hyperparameters used to compare performances are: (1) spacing parameter $S \in \{4,\ 8,\ 16,\ 32,\ 64\}$) and (2) initialization technique (*glor* - Glorot normal, *he* - He normal, *orto* - orthogonal, *gram* - Gram-Schmidt orthogonalization, *randn* - random normal).

Chapter 7

Jet Datasets

Jets are collimated sprays of particles originating from the decay and hadronization process of quarks and gluons, produced directly in the LHC collisions or in the decay of heavier particles.

In this chapter, we describe the datasets used in the studies presented throughout this work. Two high-energy physics (HEP) datasets support the research aimed at improving measurement precision at the LHC, enabling the discovery of new-physics processes and developing novel ML pipelines for physics analysis. Both physics datasets consist of jet events - sprays of particles resulting from high-energy collisions. The structure and characteristics of these jet events are explained in detail in the following section.

These datasets serve multiple purposes throughout this work. The *Jet Dataset for Anomaly Detection* is used to develop quantum-classical and quantum-inspired anomaly detection pipelines aimed at enhancing new physics discovery, as discussed in Chapters 8 and 9. The *Jet Dataset for Tagging*, explored in Chapter 10, is used to develop a Tensor Network model as a potential quantum-inspired candidate for deployment in the real-time LHC trigger system. The choice of these particular datasets reflects the current priority analysis in the LHC physics, where jet-based studies play an important role. A variety of jet types and production mechanisms are present in these datasets, ensuring the relevance of our approaches for ongoing and future LHC analyses.

7.1 Jet Substructure

When colliding in the LHC, protons produce quarks q and gluons g, collectively called partons. These partons reach the detector system after going through the following transformations. First, high-energy quarks and gluons release other particles in a multiple *parton showers*, which at the end create a cascade of lower-energy particles. This process continues until the energy of individual smaller particles reaches $\sim 1\,\text{GeV}$. At this point, quarks and gluons cannot exist freely in nature due to a phenomenon from Quantum Chromodynamics (QCD) called *confinement* - a feature of the strong force that keeps quarks together, and it gets stronger as quarks move farther apart (in contrast to the electromagnetic force). When the distance between quarks increases, the energy gets big enough to generate new quark-antiquark pairs. This suggests that isolated quarks are never found in nature. Instead, they are always combined into particles called hadrons. This process, called hadronization, transforms showers of quarks and gluons into sprays of particles that, when they are stable enough, reach the detector system. These cone-shaped sprays appear as **jets**. Researchers use sophisticated reconstruction algorithms to gather results from various detector signals, with the goal of understanding the original high-energy collision by working backwards. In particular, these algorithms combine information from tracking systems, calorimeters and muon detectors to identify individual particles and cluster them into jets [114], which can then describe the original dynamics of quarks and gluons, and help in identifying the kind of particle that originated the jet.

One of the key challenges in particle physics is the problem of determining the type of particle that produces a particular jet. This process is called **jet tagging**. Traditionally, one is interested in identifying gluon jets or various kinds of quark jets. However, due to the high energy at the LHC, heavy particles such as W, Z, and Higgs bosons or top quarks t, can be produced with large momentum. In this case, decays of these particles are compressed into a small area, forming a single large jet (*boosted jet*) that contains all the decay fragments of the individual heavy particle. For bosons, W, Z or Higgs, this results in two overlapping quark jets, and for top quarks t, three quark jets merge into one. These different structures are illustrated in Fig. 7.1 for q, g, W, Z, H and t jets, where one prong describes one quark/gluon jet [115].

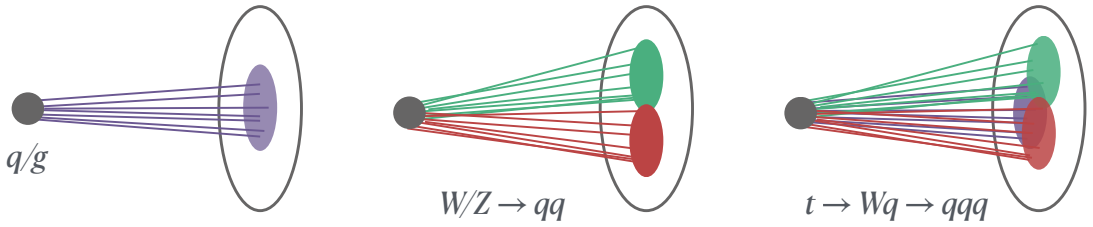


FIGURE 7.1: Schematic representation of different jet flavors. Quark and gluon jets are producing one cluster of particles, cone-shaped (left); heavy bosons W and Z are decaying to quarks, creating two clusters of particles in a single jet (center); top quark decay creates a jet composed of three clusters of particles (right).

7.2 Dataset for Jet Tagging

In this thesis, we study `hls4ml` jet dataset for classification of jet substructure, publicly available on `Zenodo` [116].

7.2.1 Composition

This dataset consists of simulated jets from $\sqrt{s} = 13\,\text{TeV}$ proton-proton collisions, with an energy of $p_\text{T} \approx 1\,\text{GeV}$ originating from: (1) quarks q, (2) gluons g, (3) W bosons W, (4) Z bosons Z and (5) top quarks t. Each jet is represented using three different data formats: high-level features capturing global jet properties, two-dimensional image representation preserving spatial relations, and low-level particle-based representation that contains kinematic information of each particle in a jet. Here, we focus on the low-level representation of a jet, describing it as a list of up to 150 particles (constituents). This representation offers the maximum amount of information about jet structure while still being computationally tractable for ML applications. Each particle is characterized by up to 16 features, derived from its *four-momenta* vector (E, p_x, p_y, p_z), including transverse momentum p_T, pseudorapidity η, azimuthal angle ϕ and various derived quantities. Table 7.1 details 12 features relevant to our analysis, while Fig. 7.2 presents their statistical distributions across different jet flavors, demonstrating the discrimination of each observable. The choice of 150 particles follows the tradeoff between the need to capture the majority of important jet information and computational efficiency. Jets containing fewer than 150 constituents are zero-padded. In contrast, if a jet has more than 150 particles, it is truncated to keep only the most relevant, first 150 particles, ranked by decreasing transverse momentum p_T. The dataset described in more detail can be found in Ref. [117].

Feature	Description
p_x	x-component of the particle momentum (Cartesian coordinate)
p_y	y-component of the particle momentum (Cartesian coordinate)
p_z	z-component of the particle momentum (Cartesian coordinate)
E	Absolute energy of the particle
p_T	Transverse momentum $\rightarrow \sqrt{p_x^2 + p_y^2}$
η	Pseudorapidity of the particle
ϕ	Azimuthal angle of the particle
ΔR	Distance from the jet center in (η, ϕ) space $\rightarrow \sqrt{\Delta\eta^2 + \Delta\phi^2}$
E^{rel}	Relative energy: $E^{\text{particle}}/E^{\text{jet}}$
p_T^{rel}	Relative transverse momentum: $p_T^{\text{particle}}/p_T^{\text{jet}}$
η^{rel}	Relative pseudorapidity: $\eta^{\text{particle}} - \eta^{\text{jet}}$
ϕ^{rel}	Relative azimuthal angle: $\phi^{\text{particle}} - \phi^{\text{jet}}$

TABLE 7.1: A list of 12 features representing each particle in the jet constituent list for different jet flavors: quarks q, gluons g, W bosons W, Z bosons Z, and top quarks t.

7.3 Dataset for Anomaly Detection

7.3.1 Description

A second jet dataset was specifically designed to test large-scale anomaly detection techniques. The dataset was created as a simulation of proton-proton collision events at the center-of-mass energy $\sqrt{s} = 13\,\text{TeV}$, produced with PYTHIA library [118]. Events are further processed with DELPHES framework [119] to simulate the CMS detector resolution and efficiency effects. Additional proton interactions (in-time pileup) are included in the simulation, modeled by incorporating low-energy events into the primary collision, and the number of pileup interactions is drawn from the Poisson distribution with a mean of 40 (parameter representing LHC conditions in 2018). Events are reconstructed using the particle-flow algorithm [120] from DELPHES, and natural units ($c = \hbar = 1$) are used to express all physical quantities.

The dataset includes both background and anomalous (signal) events. The Standard Model (SM) background is primarily composed of Quantum Chromodynamics (QCD) multijet events, the most frequent process at the LHC in fully hadronic final states. As

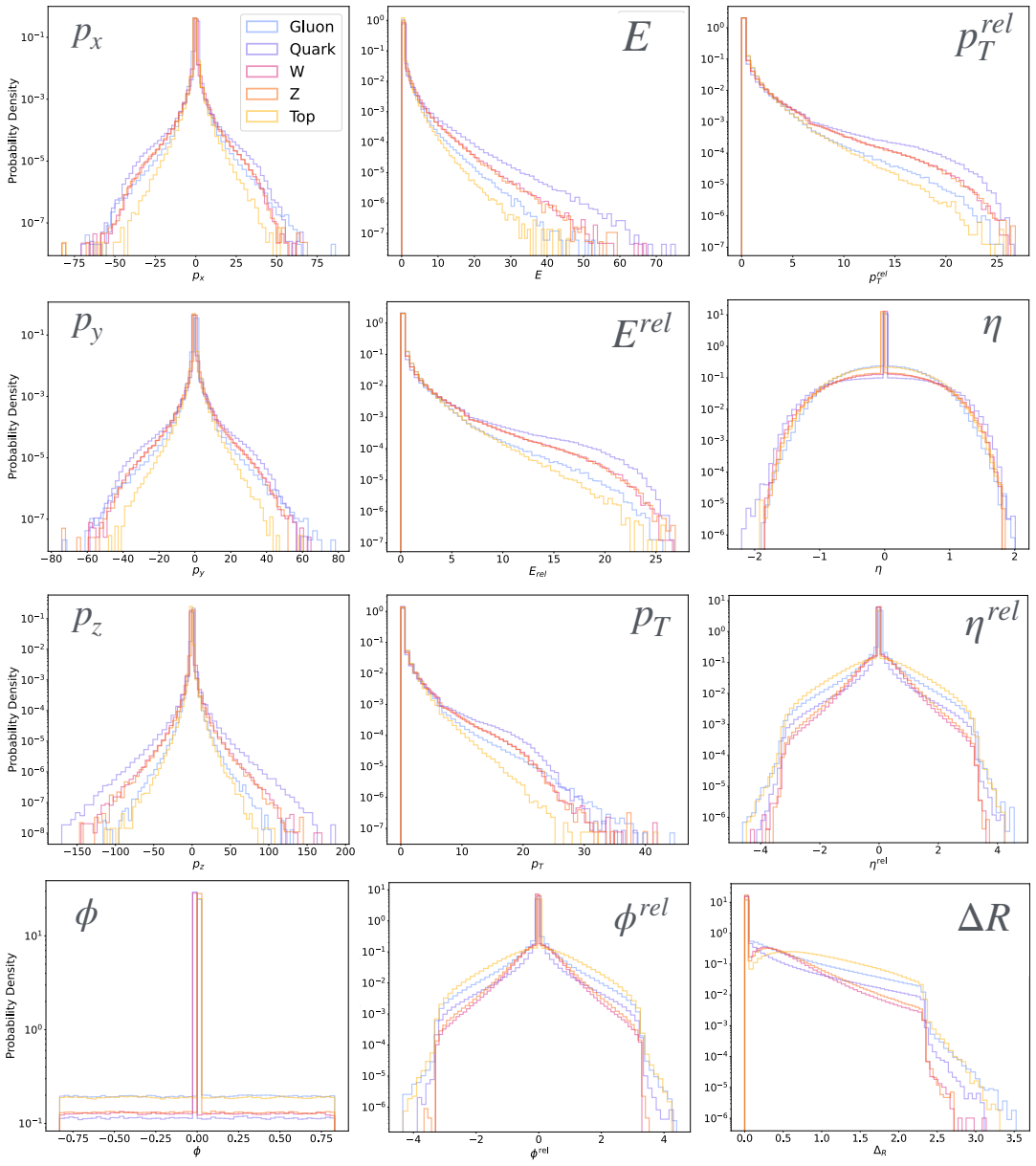


FIGURE 7.2: Distribution of 12 features describing each particle of a jet, for all five types of jets in this dataset.

potential Beyond the Standard Model (BSM) scenarios, three representative processes are selected:

- (Narrow $G \to WW$): production of narrow Randall-Sandrum (RS) [121] gravitons decaying to W boson pairs;

- (Broad $G \to WW$): production of broad RS gravitons decaying to W boson pairs;

- $(A \to HZ \to ZZZ)$: production of scalar bosons A decaying into Higgs (H) and Z bosons, where Higgs further decays into ZZ pair, resulting in a ZZZ final state.

By forcing W and Z bosons to decay into quarks, which rapidly hadronize, the final state of the event contains only jets. This event topology is usually referred to as all-jet events.

7.3.2 Selection and Preprocessing

Jets are formed from reconstructed particles using the anti-k_{T} clustering algorithm [114] with a radius parameter of jet cone $R = 0.8$, as implemented in the FastJet [122] package. Events are selected with the requirement of selecting two leading jets with transverse momentum $p_{\mathrm{T}} > 100$ GeV and pseudorapidity $\eta < 2.4$, with invariant mass $m_{\mathrm{jj}} > 1260$ GeV to satisfy online selection thresholds. m_{jj} is the invariant mass of a system composed of two jets, calculated as $m_{jj} = \sqrt{E_{jj}^2 - |\vec{p}_{jj}|^2}$, where $E_{jj} = E_{j_1} + E_{j_2}$ is the sum of jet energies, and $\vec{p}_{jj} = \vec{p}_{j1} + \vec{p}_{j2}$ is the vector sum of their momenta. The squared magnitude of the total momentum is given by $|\vec{p}_{jj}|^2 = (p_{x_1} + p_{x_2})^2 + (p_{y_1} + p_{y_2})^2 + (p_{z_1} + p_{z_2})^2$.

Each jet in an event is represented by 100 particles with the highest p_{T} within a cone of $\Delta R < 0.8$, where all constituents are ordered by decreasing p_{T}. Each particle is described by its p_{T}, and angular distance from the jet axis ($\Delta\eta$ and $\Delta\phi$), which results in a 100×3 matrix representing one jet. If fewer than 100 particles are available, zero padding is applied. The final all-jet events are represented in matrix format in Fig. 7.3.

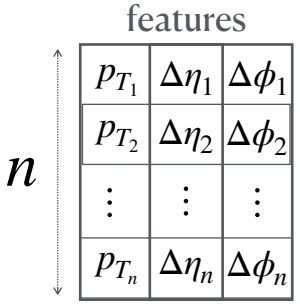


Figure 7.3: Matrix representation of a jet event represented with n particles each with three features $p_{\mathrm{T}}, \Delta\eta, \Delta\phi$.

The width of the RS graviton is controlled via the Pythia parameter $\kappa \times m_G$, set to 0.01 for the narrow case, and to 2.5 for the broad case, resulting in a resonance with a width

approximately 35% of the mean reconstructed dijet mass. Masses for BSM resonances are varied from 1500 to 4500 GeV in 1000 GeV steps.

QCD data is divided into two kinematic regions of interest defined by the pseudorapidity separation between the two highest-p_{T} jets. The first one is called a *sideband* region, which has $|\Delta\eta_{jj}| > 1.4$, and is minimally contaminated with BSM signals. The other region is defined with the requirement of $|\Delta\eta_{jj}| \leq 1.4$. It is called a *signal* region, since potential anomalies are more likely to appear in it. Fig. 7.4 shows the distribution of parameters $\Delta\eta_{jj}$, $\Delta\phi_{jj}$ and p_{T} of constituent particles, distinguishing the sideband and signal regions. Fig. 7.5 shows the distribution of parameters $\Delta\eta_{jj}$, $\Delta\phi_{jj}$ and p_{T} of two jets, also for the QCD sideband and signal regions. Fig. 7.6 shows m_{jj} distributions for QCD sideband and signal regions and for each signal scenario with varied masses.

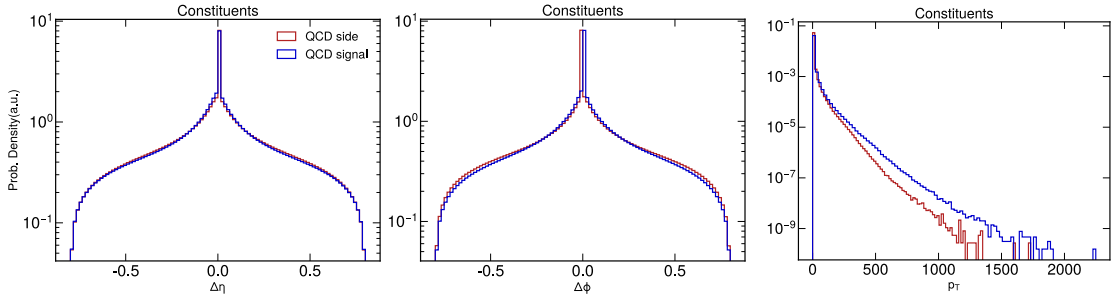


FIGURE 7.4: Distribution of particle features $p_{\mathrm{T}}, \Delta\eta, \Delta\phi$ for sideband (red) and signal (blue) QCD regions.

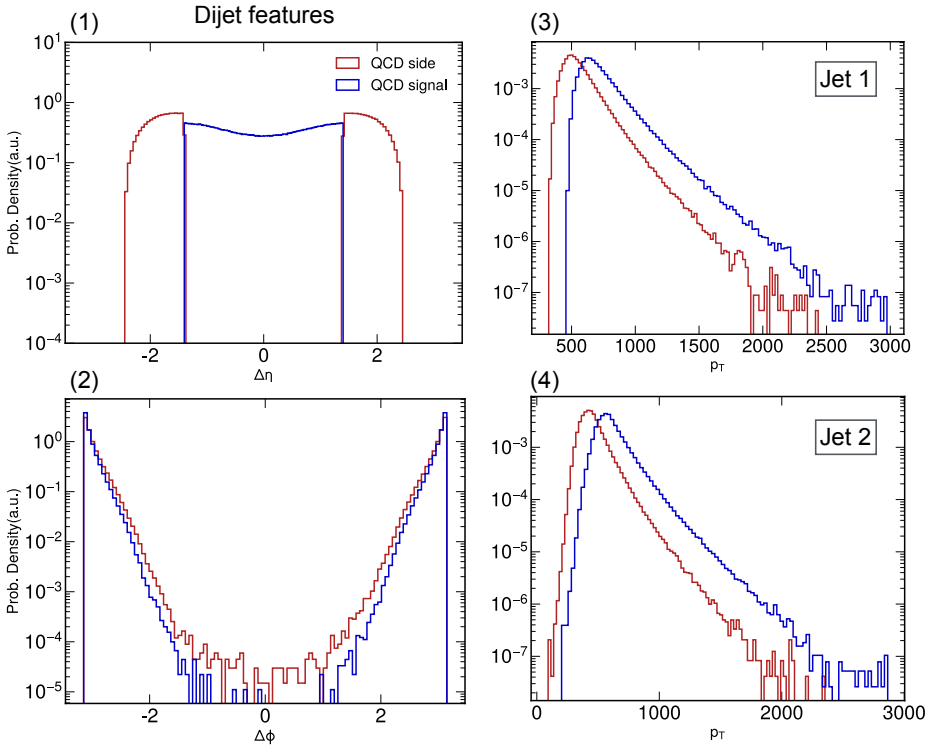


FIGURE 7.5: Distribution of features p_{T} for each jet and dijet features $\Delta\eta, \Delta\phi$ for sideband (red) and signal (blue) QCD regions after the event selection.

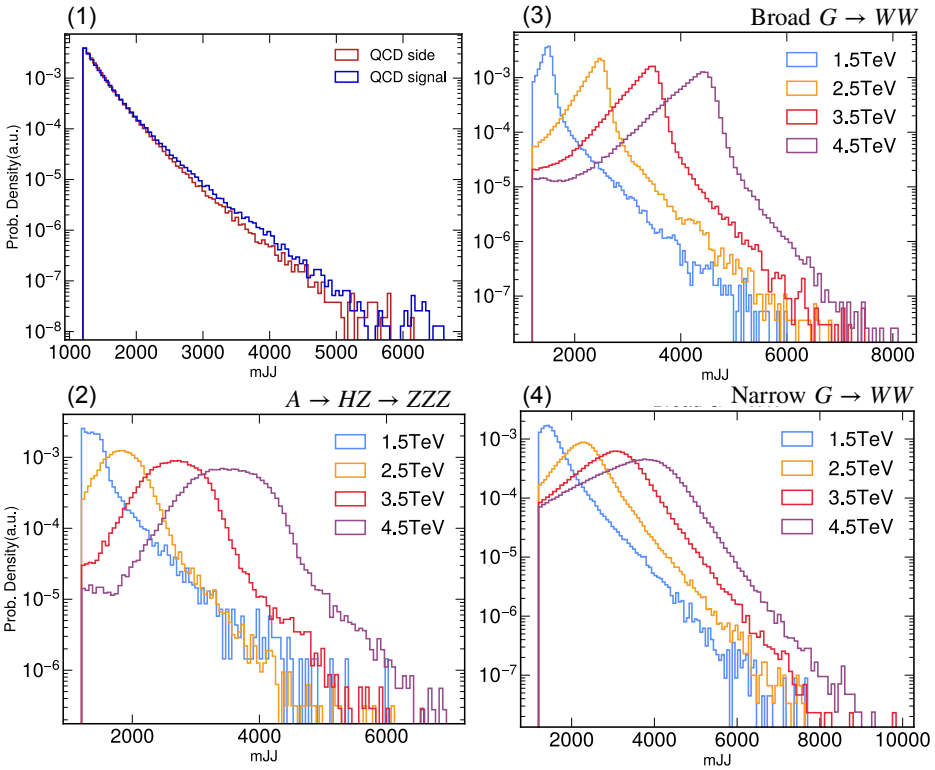


FIGURE 7.6: Distribution of m_{jj} for sideband (red) and signal (blue) QCD regions, and for each signal process with different center-of-mass energies in TeV.

Chapter 8

Quantum-Classical Clustering for Anomaly Detection at the LHC

In this chapter, we present an anomaly detection pipeline that combines a pre-trained encoder for dimensionality reduction with a quantum-classical k-medians clustering algorithm applied in the latent space of proton-proton collision events at the LHC. This pipeline is a part of the broader systematic study for anomaly detection introduced in Ref. [1], which compares a kernel method and two clustering approaches (k-means and k-medians) for detecting anomalies in the latent representations of collision events. The study further involves comparing quantum versus classical versions of the clustering method to assess the actual contribution of quantum components. Each element of this hybrid pipeline, illustrated in Fig. 8.1, is designed heuristically, with its motivation demonstrated sequentially in the result section. Overall, the final performance highlights the potential of the quantum approach as a viable alternative to its classical counterpart, showing comparable or slightly improved performance in various regions of the evaluation metrics. Notable results are proving the superior performance of the quantum algorithm in regimes with limited training data.

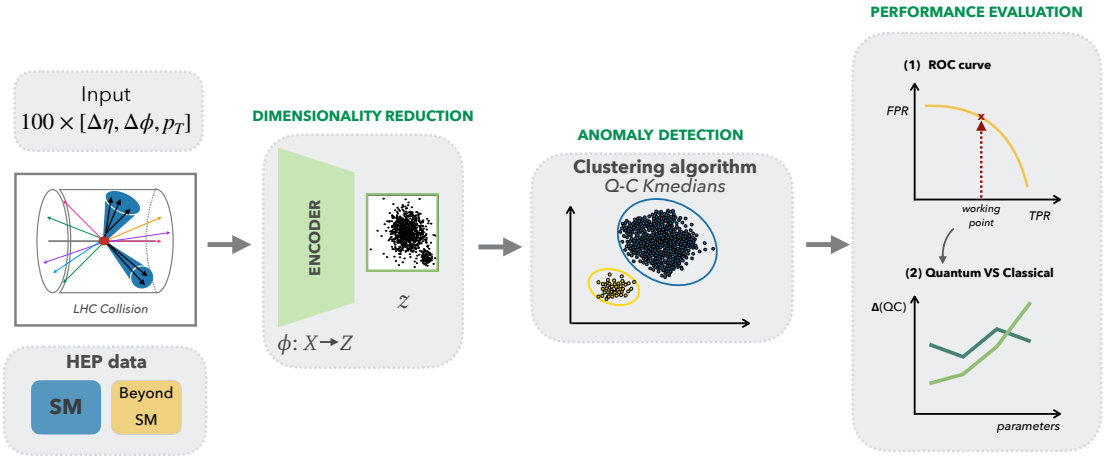


FIGURE 8.1: **Anomaly Detection Pipeline** Dijet events produced in proton-proton collisions at the LHC are passed through an autoencoder for dimensionality reduction, followed by a quantum-classical k-medians clustering algorithm for anomaly detection. Each jet is represented by up to 100 particles, each having three features, resulting in the input matrix format $100 \times [\Delta\eta, \Delta\phi, p_T]$. Standard events are used to train the pipeline to detect future unseen samples, possibly originating from previously unobserved processes - so-called new physics. The pipeline is evaluated by calculating the ROC curve and supporting metrics, and the quantum-classical k-medians model is compared to its pure classical counterpart.

8.1 Motivation for (Quantum) Anomaly Detection in HEP

As discussed in Sec. 2.3 and further motivated by the review article in Ref. [44], ML-based anomaly detection has become increasingly important in HEP for the discovery of new-physics scenarios. Unlike traditional model-dependent searches, self-supervised or unsupervised anomaly detection models - trained only on background samples - do not rely on any prior signal hypothesis, making them more robust and versatile for uncovering unknown signals. These models are mostly based on autoencoders [123], clustering algorithms [39, 123], or Gaussian Mixture Models [124]. In general, clustering algorithms detect the appearance of anomalous clusters in the presence of an unknown signal, especially when these differ significantly from the learned feature space. Autoencoders aim to learn the difference between real and reconstructed data and use the reconstruction error, or deviations in the latent space, as an anomaly score. The ATLAS collaboration at CERN proposed such models in anomaly detection [125, 126] and a weak-supervised [40] setting. Similarly, the CMS collaboration investigated the integration into a real-life event selection system at the CMS detector in earlier studies [127, 128], and has since developed and tested model-agnostic approaches for deployment at the trigger level [45, 129]. The

anomaly detection task discussed in this chapter is framed as an *out-of-distribution* problem, benefiting from training solely on known background - unlabeled Standard Model events - with the aim of enhancing sensitivity to previously unseen signal processes.

Recently, the field of quantum machine learning (QML) has contributed significantly to the development of anomaly detection techniques. These include Quantum Kernel Methods (discussed later in this work, see Ref. [1]), Quantum Autoencoder [130], Quantum Support Vector Classifier [131], and Quantum Generative Adversarial Network [132], which have demonstrated advantages over classical counterparts in certain data regimes, or have shown better convergence and the ability to train effectively with smaller datasets.

This study implements an anomaly detection pipeline (illustrated in Fig. 8.1) that combines an autoencoder model for dimensionality reduction - commonly used to reduce data complexity [127, 133] - with a quantum-classical clustering algorithm. The goal is to leverage both the quantum nature of the physics data (described in Sec. 7.3) and the efficiency of classical optimization techniques for finding the nearest cluster. The systematic study compares the quantum-classical model with its pure classical counterpart to understand the potential of quantum parts.

8.2 Autoencoder-Based Dimensionality Reduction

The primary motivation for reducing the dimensionality of the data in this study comes from the limited number of qubits in current quantum hardware, which makes it infeasible to process the full input size of the dataset directly. While traditional methods like Principal Component Analysis (PCA) offer a linear approach to dimensionality reduction, their ability remains limited for modeling complex, nonlinear relationships among features, which is a common characteristic of HEP data. In contrast, an autoencoder (AE) is a multi-layered neural network that learns the compressed representation of the data by capturing nonlinear correlations. This way, AE preserves the important features of the data in a lower-dimensional latent space. Additionally, the architecture of the AE is more flexible and offers control when studying the trade-off between the amount of compression and performance, when compared to the PCA method.

To construct a training dataset for the AE, we utilize a *sideband* region of QCD-background data (as defined in Sec. 7.3), without relying on any true labels. The input to

AE is structured as a matrix where each sample corresponds to a single jet, represented by 100 particles, each described by three features: p_{T}, $\Delta\eta$, and $\Delta\phi$. The AE is designed to operate at the *per-particle* level and consists of convolutional layers, with a final activation function of *tanh*, positioning latent features in the range $[-1,\ 1]$. The AE consists of a series of consecutive convolutional and dense layers in both the encoder and decoder parts. The full architecture of the AE is described in detail in Ref. [1] and [134], and visualized in Fig. 8.2. Once trained, the AE is used to process QCD events from *signal* regions, producing latent representations of individual jets, denoted as vector z with dimension l. Figure 8.3 shows the latent space representations of single jet events from the QCD background and the Narrow $G \to WW$ signal with a mass of 3.5 TeV. Notably, a degree of separation between background and signal events is already achieved by dimensionality reduction, a point that is further discussed in the results section. These latent features serve as input for training the clustering algorithm, where each event is formed as a concatenation of two latent vectors, with length $2l$, resulting in di-jet samples. Both background and anomalous events used for evaluating the clustering performance are selected from the same signal region.

The same input dataset is used in Ref. [134], where a particle-based variational autoencoder (p-VAE) is employed for anomaly detection, using the same architecture as the AE applied here for dimensionality reduction. The only difference lies in the variational component of the latent space and the choice of latent dimension: the best results in Ref. [134] are achieved with a latent space of size 12. The findings from that study are referenced in the discussion of results.

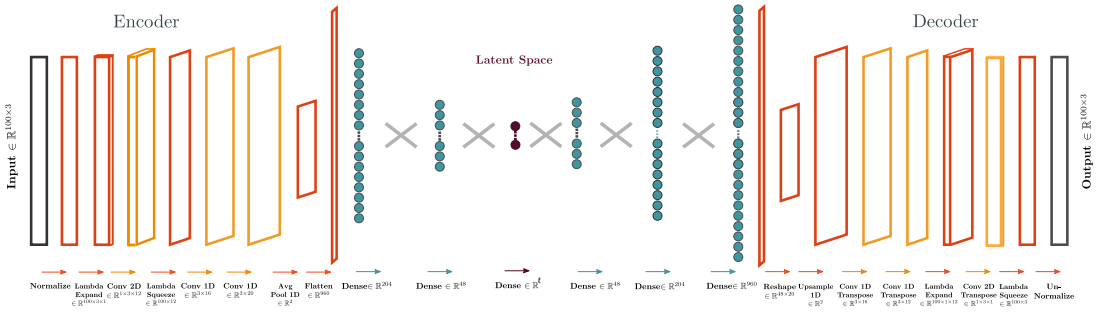


FIGURE 8.2: The architecture of the autoencoder used to reduce the dimensionality of the dataset from 100×3 dimensions per jet to a latent space dimension of l. Both the encoder and decoder parts consist of three convolutional layers and three dense layers, designed to act on *per-particle* level. The output of the encoder is used as input to the anomaly detection studies. Figure is taken from Ref. [134].

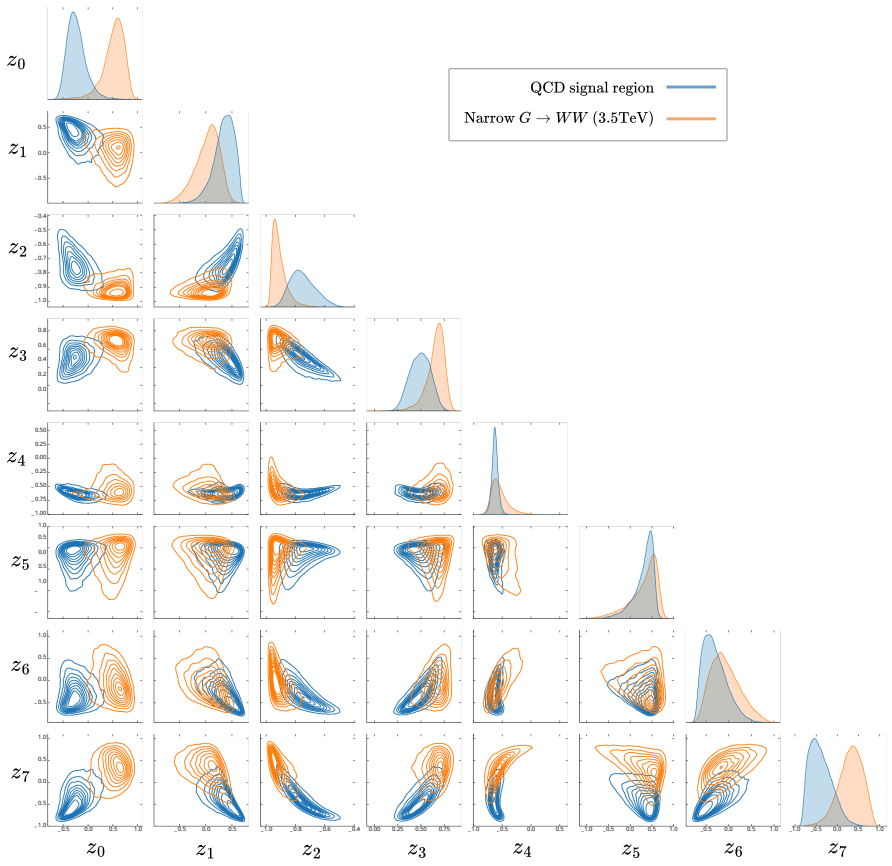


FIGURE 8.3: Latent space representation z with dimension $l = 8$ of the QCD signal region (blue) and the Narrow $G \to WW$ signal (orange) with a mass of 3.5 TeV.

8.3 Classical k-Medians Clustering

The k-medians clustering algorithm is a variation of the k-means algorithm that uses median calculation instead of the mean to update cluster centroids. Similar to k-means, the k-medians aims to partition the dataset into k clusters such that it learns to position a data point in the cluster with the nearest centroid. The distance to the cluster centroid is calculated using $L2$ distance. The algorithm begins by randomly selecting k initial centroids from the dataset. Then, for each data point x, the distance to all current centroids is computed, and the point is assigned to the nearest cluster. After all assignments, each cluster's centroid is updated by calculating the *median* of the points assigned to that cluster. The algorithm repeats iteratively until centroids converge, meaning the change for update of the next centroid is below a specified threshold ϵ. The pseudocode of the algorithm is presented in Algorithm 1.

Algorithm 1 Classical k-medians clustering algorithm.

Input: $\mathcal{D}$: `array` with N points, k: `int`
centroids $\leftarrow \mathcal{D}$[random_indices(0, N, k)]
repeat
  for each data point x in $\mathcal{D}$ **do**
    $d \leftarrow$ L2DISTANCES(x, centroids)
    $c \leftarrow \arg\min d$
    Assign x to cluster c

  for each cluster c **do**
    centroid$_c \leftarrow$ median of points assigned to c

until cluster assignments stable (change $< \epsilon$)

function L2DISTANCES(x, centroids)
  for each centroid c in centroids **do**
    $d[c] \leftarrow \sqrt{\sum_{i=1}^{n}(x_i - c_i)^2}$

  return d

8.4 Quantum-Classical k-Medians Clustering

The final heuristic structure of the algorithm is based on Ref. [135], with additional adaptations made to each subroutine within the algorithm. A *geometric median* is defined as an existing point in the vector space with the minimum distance to all other points. This avoids the problem commonly found in k-means algorithm, where the *mean* can sometimes fall outside the region spanned by the data [135].

The quantum-classical k-medians clustering algorithm has two subroutines:

1. Quantum circuit DISTCALC for distance calculation using destructive interference probabilities (from Ref. [136]);

2. Quantum-classical subroutine for finding cluster medians (centroids) MEDIANCALC (adopted from Ref. [137]), using DISTCALC quantum circuit for distance calculation.

The full pseudocode for the quantum-classical k-medians clustering algorithm goes as follows:

Algorithm 2 Quantum-classical k-medians clustering algorithm.

Input: $\mathcal{D}$: `array` with N points, k: `int`
centroids $\leftarrow \mathcal{D}$[random_indices(0, N, k)]
repeat
  for each x in $\mathcal{D}$ **do**
    $d \leftarrow$ DistCalc(x, centroids)
    $c \leftarrow \arg\min(d)$
    assign x to cluster c

  for each cluster c **do**
    centroid$_c \leftarrow$ MedianCalc(points in c)

until cluster assignments stable (change $< \epsilon$)

8.4.1 Quantum Distance Calculation

The standard quantum ansatz used for distance calculation (explained in Ref. [1]) consists of preparing two quantum states and a swap gate, which is very costly. In this work, we use a shallower quantum circuit for distance calculation with state preparation of one quantum state and one Hadamard gate, referred to as DistCalc.

For n-dimensional data samples, given two points in vector space $(x_1, ..., x_n)$ and $(y_1, ..., y_n)$, a quantum state is prepared by first normalizing the vectors with a normalization factor $\mathcal{N}$:

$$\mathcal{N} = \sqrt{\sum_{i=1}^{n} x_i^2 + \sum_{i=1}^{n} y_i^2} \tag{8.1}$$

$$x_i' = \frac{x_i}{\mathcal{N}};\ y_i' = \frac{y_i}{\mathcal{N}}. \tag{8.2}$$

Then, the vector $\vec{v}$ is defined as:

$$\vec{v} = [x_1', \ldots, x_n',\ y_1', \ldots, y_n']^T, \tag{8.3}$$

where each vector $\vec{x}$ and $\vec{y}$ is zero-padded, if necessary, so that the total length of $\vec{v}$ is a power of two, i.e., $2n = 2^m$ for some $m \in \mathbb{N}$. To represent this vector as a quantum state $|\phi\rangle$ we need $m = \log_2(2n)$ qubits. The final prepared state $|\psi\rangle$ is expressed as:

$$|\psi\rangle = \sum_{i=0}^{2n-1} v_i \, |i\rangle \,, \tag{8.4}$$

where v_i is a i^{th} component of the normalized vector $\vec{v}$. Using m qubits, we can rewrite the state with one *data* qubit and $m-1$ *index* qubits as:

$$|\psi\rangle = \sum_{i=0}^{n-1} x'_i |0\rangle |i\rangle + y'_i |1\rangle |i\rangle \tag{8.5}$$

After state preparation, a Hadamard gate (explained in Table 3.1) is applied to the most significant qubit (MSQB) - *data* qubit - which transforms the state to:

$$|\psi'\rangle = \frac{1}{\sqrt{2}} \sum_{i=0}^{n-1} \left[x'_i(|0\rangle + |1\rangle) |i\rangle + y'_i(|0\rangle - |1\rangle) |i\rangle \right] \tag{8.6}$$

$$= \frac{1}{\sqrt{2}} \sum_{i=0}^{n-1} \left[(x'_i + y'_i) |0\rangle |i\rangle + (x'_i - y'_i) |1\rangle |i\rangle \right] \tag{8.7}$$

By measuring the MSQB in the state $|1\rangle$ we obtain destructive inference probabilities $P(|1\rangle) = \frac{1}{2}\|\vec{x}' - \vec{y}'\|^2$ and calculate the distance as:

$$\text{dist}(\vec{x}, \vec{y}) = \mathcal{N} \cdot \sqrt{2 \cdot P(|1\rangle)} = \|\vec{x} - \vec{y}\| \tag{8.8}$$

Fig. 8.4 illustrates a scheme of the quantum circuit used for distance calculation.

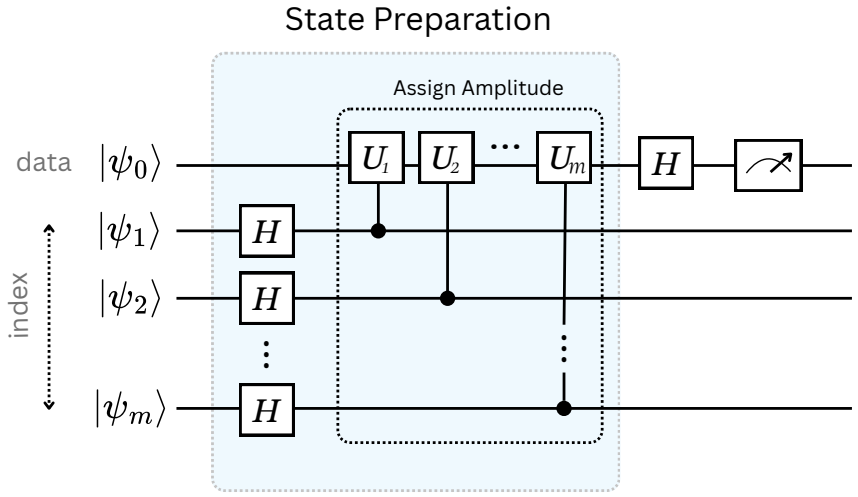


FIGURE 8.4: Quantum circuit for **distance calculation** using a state preparation routine followed by a Hadamard gate H on the most significant qubit (MSQB), referred to as the *data* qubit. The state preparation consists of two steps: first, H gates are applied to each *index* qubit to create a uniform superposition over basis states $|i\rangle$; second, for each $|i\rangle$, a unitary U_i is applied to the data qubit, encoding an amplitude $x'_i|0\rangle + y'_i|1\rangle$. The final H on the data qubit allows for interference between x'_i and y'_i components, resulting in final state expression $(x'_i + y'_i)|0\rangle + (x'_i - y'_i)|1\rangle$. This final state is used to compute the distances when measuring the data qubit in state $|1\rangle$.

8.4.2 Quantum-classical subroutine for finding centroids

The MedianCalc subroutine is using DistCalc to calculate the distance between a given centroid to all other points, and using a classical heuristics approach from Ref. [137] for finding the median as a point with minimum distance to all other points. The heuristic method tries to find the median by searching toward more concentrated points. Given a set of points $\mathcal{X} = \{x_1, x_2, \ldots, x_N\} \in \mathbb{R}^d$, the *geometric median* $m*$ is defined as:

$$m^* = \arg\min_{m \in \mathbb{R}^d} \sum_{i=1}^{N} \|x_i - m\|. \tag{8.9}$$

Step 1 For current median (centroid) $m^{(t)}$, compute:

$$d_i = ||x_i - m^{(t)}||, \tag{8.10}$$

using quantum subroutine DistCalc and only keep nonzero distances in a set: $\mathcal{I} = \{i \in \{1, ..., N\} : d_i \neq 0\}$.

Step 2 Compute the inverse-distance weights:

$$W_i = \frac{1/d_i}{\sum_{j \in \mathcal{I}} 1/d_i}, \tag{8.11}$$

and the weighted sum of the nonzero distance points:

$$\mathrm{T}_1 = \sum_{\substack{i \in \mathcal{I} \\ x_i \neq m^{(t)}}} W_i x_i, \tag{8.12}$$

which represents a weighted center based on inverse distances, giving higher influence to the points closer to the current median.

Step 3 Handle zero-distance cases by setting a rule:

$$y = \text{number of points with } d_i \text{ equal to } 0 = N - |\mathcal{I}|, \tag{8.13}$$

which serves to update the median. There exist three cases based on y:

1. $(y = 0)$ - there are no points exactly on current median $\rightarrow$ set $m^{(t+1)} = T_1$;
2. $(y = N)$ - all points exactly lie at the median $\rightarrow$ no update is needed $m^{(t+1)} = m^{(t)}$;

3. $(0 < y < N)$ - exists a partial overlap $\rightarrow$ update with convex combination by first calculating correction term:

$$R = (T_1 - m^{(t)}) \cdot \sum_{j \in \mathcal{I}} \frac{1}{d_j}, \tag{8.14}$$

and expressing parameters $r = ||R||$ and $\gamma = \min(1, \frac{y}{r})$.

Thus, updating the next median as:

$$m^{(t+1)} = (1 - \gamma)T_1 + \gamma m^{(t)}. \tag{8.15}$$

Step 4 After updating the median, compute the distance to the previous median:

$$||m^{(t+1)} - m^{(t)}|| < \epsilon, \tag{8.16}$$

where ϵ is the tolerance. If this condition is met, the algorithm has converged and the final geometric median is $m^{(t+1)}$.

This algorithm is beneficial for high-dimensional tasks, as it provides a more meaningful centroid for multivariate data by taking all features into account jointly, and it is robust to outliers by minimizing the sum of distances to all data points.

8.5 Anomaly Detection Pipeline

Two jet events, each represented with an input matrix of size 100×3, are first processed by the trained AE, producing a latent feature vector $z \in \mathbb{R}^l$ for each jet. For each latent vector, we calculate the distance to the k cluster centroids, which were found by a previous training of the clustering algorithm. The anomaly detection score is calculated first by computing the sum of squared distances to k cluster centroids:

$$s = \sum_k |z \ - \ c_k|^2, \tag{8.17}$$

and then measuring the minimum distance between the two jets.

8.6 Results and Discussion

The model's performance is evaluated by analyzing its ability to detect anomalous processes. It is important to note that the AE used for dimensionality reduction can be used for detecting anomalies by using reconstruction error as an anomaly score, where a higher error indicates a more anomalous event. The results from AE are used as a baseline for the following analysis. From the anomaly scores s obtained using clustering algorithm, we compute the ROC curve and calculate two metrics: (1) the Area Under the ROC curve (AUC), and (2) inverse False Positive Rate (FPR^{-1}) at a specific True Positive Rate (TPR), corresponding to a benchmark working-point threshold that one would apply on the score to use the algorithm in a real-life application. We follow the HEP language for addressing this last metric by noting FPR a background efficiency ε_b, and TPR a signal efficiency ε_s. Typical ε_s points important for analysis at the LHC are $0.6, 0.8$, what we use to compare ε_b^{-1}. All quantum circuits are evaluated in simulation.

We study the final performance across three data-dependent scenarios:

- Different new-physics processes (mentioned in Sec. 7.3);
- Latent space dimensions $l \in \{4, 8, 16, 32\}$ (sometimes including $\{24, 40\}$);
- Number of training samples $N_{\text{train}} \in \{10, 600, 6000\}$.

When it is not explicitly mentioned, then we set $N_{\text{train}} = 600$ and $l = 8$. After setting the final configuration for the quantum-clustering algorithm, the following evaluation is performed as 5-fold testing, using 10^{-5} samples (divided evenly into anomalies and SM samples).

Searching for correct setup Before showing the results of the final fixed algorithm setup, we first describe the reasons behind the choices made for the Algorithm 2. For this preliminary part, we focus on the signal scenario Narrow $G \to WW$ with mass 3.5 TeV.

Since the number of clusters k must be predefined, we investigate its impact on the algorithm's performance while keeping all other parameters consistent with Algorithm 2. As shown in Fig. 8.5(1), the best performance is achieved for $k = 2$, thus we set it accordingly.

Next, we analyze the effectiveness of the MedianCalc subroutine compared to the standard `numpy.median` for finding cluster centroids. Fig. 8.5(2) shows that MedianCalc outperforms the `numpy.median` for higher-dimensional latent spaces, while achieving similar performance for other values of l. Based on this, we select MedianCalc as the subroutine for computing cluster medians.

The last part of this preliminary study includes checking whether employing a quantum algorithm for finding the minimum distance - specifically, *Grover*'s search [51, 138] - offers any advantage in this setting. Since *Grover*'s algorithm provides a quadratic speedup $\mathcal{O}(\sqrt{k})$ only when k is large[1], its benefit is not important for two clusters $k = 2$. Fig. 8.5(3) confirms that the same performance is achieved across different latent spaces for both functions. Therefore, we choose to use the classical version for finding the minimum.

New-physics signal comparison Comparison of the algorithm's performance across different signals is shown in Fig. 8.6, with corresponding performance of the classical counterpart. Different signal scenarios yield different performances. This is justified by the theoretical explanation that some BSM processes are more similar to SM events, which makes them harder to detect (e.g., the broad Graviton process). Quantum-classical model shows competitive performance with its classical counterpart, especially for the signal process Narrow G $\to$ WW (3.5 TeV) with achieved AUC of 97.68%. For the $A \to HZ \to ZZZ$ process at center-of-mass energy 3.5 TeV, the classical version of the algorithm marginally outperforms the quantum one in the mid-FPR range, though the difference is within statistical uncertainty. The detection of Broad Broad G $\to$ WW 3.5 TeV is near-random both for classical and quantum versions, when considering AUC as evaluation metric. On the other hand, if a low-TPR (e.g., 0.1, 0.2) is considered as a working point, both versions of the algorithm are a factor of $>\sim 2$ better than a random solution. This is an example of why sometimes the AUC is not expressive enough to assess the effectiveness of a classifier. Notably, all quantum counterparts have lower uncertainty, which makes them more robust in generalization. When comparing with AE from Fig. 8.2, considering the AUC metric, the clustering method outperforms AE only for signal Narrow G $\to$ WW, while considering a more important metric ε_b^{-1} at fixed ε_s the clustering method shows improvement both for Narrow Graviton and $A \to HZ \to ZZZ$ signals. The p-VAE model from Ref. [134] has a higher latent dimension of 12, and outperforms both quantum and classical clustering, as well as the AE model. However, clustering algorithms are

[1]Grover's algorithm provides a speedup of $\mathcal{O}(\sqrt{n})$ when searching for a minimum in an array of size n.

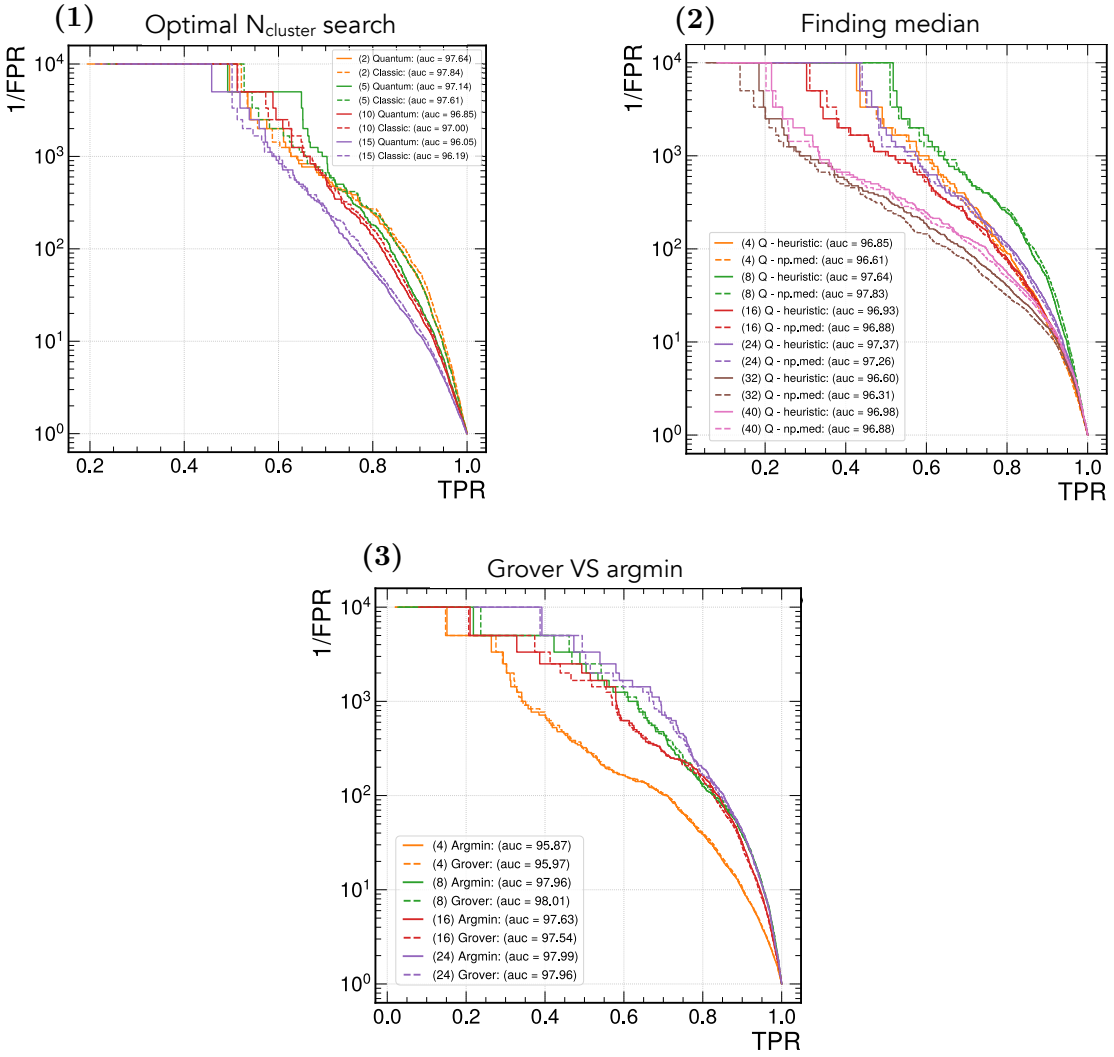


FIGURE 8.5: Preliminary study to find optimal setup for Algorithm 2 using ROC curve and corresponding AUC value to evaluate the performance. (1) Search for the optimal number of clusters k among values $\{2, 5, 10, 15\}$ for the quantum-classical and pure classical clustering; (2) Performance comparison for different median finding methods for the quantum-classical clustering algorithm and latent space $l \in \{4, 8, 16, 24, 32, 40\}$; (3) Analysis of performance advantage for Grover's algorithm for the quantum-classical clustering algorithm and latent space $l \in \{4, 8, 16, 24\}$.

trained on only 600 samples, whereas both AE and p-VAE are trained on up to 10,000 to 1,000,000 samples, which is up to four orders of magnitude more. This gap limits the fairness of direct comparison and highlights the potential for significant improvement in clustering-based models.

Latent dimension comparison To assess the impact of latent space dimensionality, we evaluate the algorithm's performance on a fixed signal scenario $A \to HZ \to ZZZ$. AE achieves an AUC value of $\sim 89.5\%$ for all latent spaces for this signal type, and the metric ε_b^{-1} at fixed $\varepsilon_s \in \{0.6,\ 0.8\}$ shows values of ~ 15.3 and ~ 6.1 correspondingly, also for all

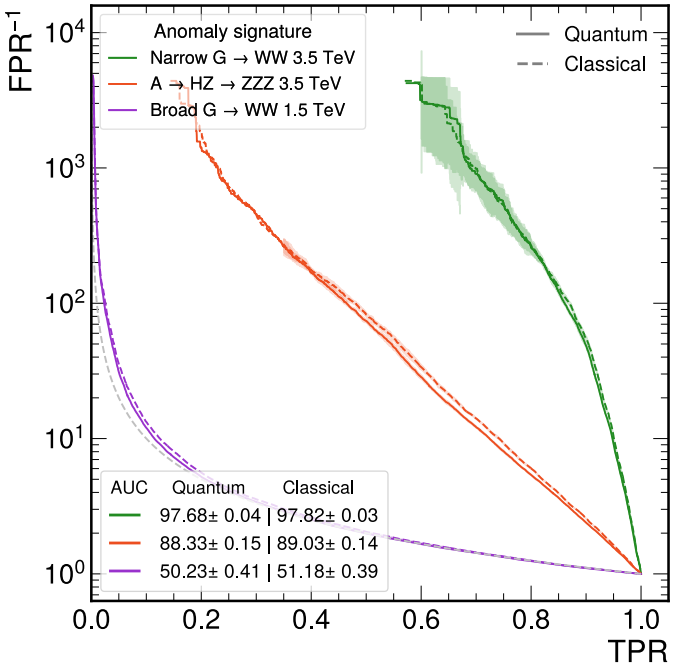


FIGURE 8.6: The ROC curve and the Area Under the ROC curve (AUC) evaluated on different new-physics signatures for quantum and classical clustering algorithms. Here, `Quantum` denotes the quantum-classical clustering method. Error bands are the result of the 5-fold testing procedure. The statistics are very low in the area of lower TPR values, leading to large errors, which are omitted from the plot. The gray dotted line corresponds to taking a random decision.

latent spaces. This result is used as a baseline for comparison for the following analysis. As shown in Fig. 8.7, reducing the dimensionality with an AE from $\mathbb{R}^{300}$ to $\mathbb{R}^4$ maintains satisfactory performance for both classical and quantum clustering algorithms. The best performance is achieved at a latent space size of $l = 8$, while for $l = 32$ the performance drops significantly. The ROC curves demonstrate that the quantum algorithm slightly outperforms the classical one across all latent dimensions except for $l = 8$, where the performance is comparable. Table 8.1 quantitatively supports these observations by reporting the inverse of background efficiency ε_b^{-1} at signal efficiency $\varepsilon_s \in \{0.6, 0.8\}$, where higher values indicate superior performance. These results confirm that quantum models can operate efficiently in lower latent dimensions, and their comparable performance to the classical baselines demonstrates their potential as promising alternatives. The best methods for both quantum and classical algorithms are shaded in gray. When comparing with AE models, results for the latent space $l = 8$ show better performance for all evaluation metrics, while for other dimensions, e.g., $l = 16$ and 32, the AE algorithm performs better as a standalone, suggesting either overfitting with an additional clustering algorithm or lack of training samples for higher dimensions.

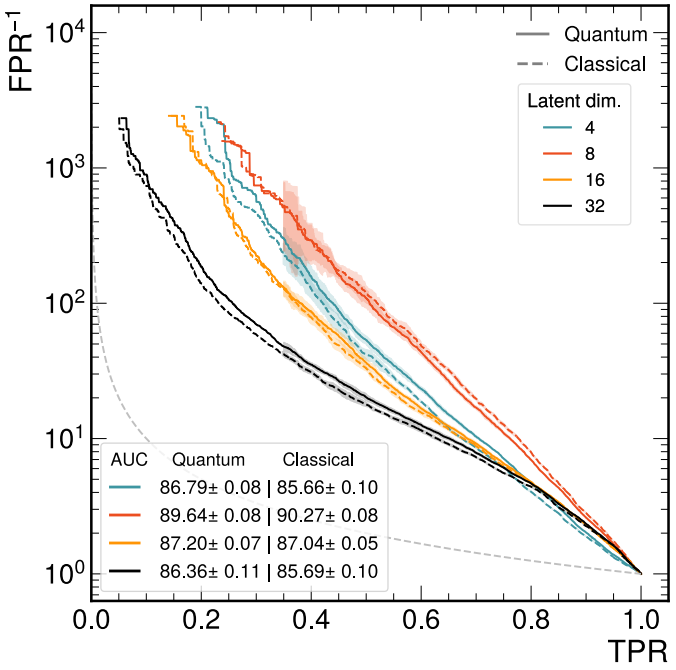


FIGURE 8.7: The ROC curve and the Area Under the ROC curve (AUC) evaluated on different latent space dimensions for quantum and classical clustering algorithms. Here, `Quantum` denotes the quantum-classical clustering method. Signal scenario is set to $A \to HZ \to ZZZ$ and $N_{\text{train}} = 600$. Error bands are the result of the 5-fold testing procedure. The statistics are very low in the area of lower TPR values, leading to large errors, which are omitted from the plot. The gray dotted line corresponds to taking a random decision.

Latent	Method	$\varepsilon_s = 0.6$	$\varepsilon_s = 0.8$
4	Q	23.29 ± 1.01	4.68 ± 0.09
	C	18.72 ± 0.37	4.01 ± 0.10
8	Q	43.48 ± 2.62	6.93 ± 0.13
	C	48.72 ± 3.20	7.83 ± 0.14
16	Q	16.76 ± 0.69	4.75 ± 0.03
	C	15.56 ± 0.68	4.63 ± 0.08
32	Q	12.59 ± 0.46	4.64 ± 0.06
	C	11.50 ± 0.48	4.38 ± 0.04

TABLE 8.1: ε_b^{-1} values (± standard deviation) at fixed ε_s levels, for varying latent space dimensions across quantum and classical algorithms for signal scenario $A \to HZ \to ZZZ$. The higher value indicates better performance. Standard deviations are the result of the 5-fold testing procedure.

Train size comparison To investigate the claim that quantum algorithms can outperform their classical counterparts when trained on smaller train sizes, we analyze the impact of N_{train} on the model's performance for a latent $l = 8$ and the signal process $A \to HZ \to ZZZ$. As shown in Fig. 8.8, performance improves with increasing training size, as expected. Notably, the results confirm the initial hypothesis: for $N_{\text{train}} = 10$ the quantum

model outperforms the classical one. Table 8.2 numerically supports these observations by reporting the inverse of background efficiency ε_b^{-1} at fixed signal efficiencies $\varepsilon_s \in \{0.6, 0.8\}$, where superior performance of the quantum counterpart is only observed for $N_{\text{train}} = 10$ (shaded in gray).

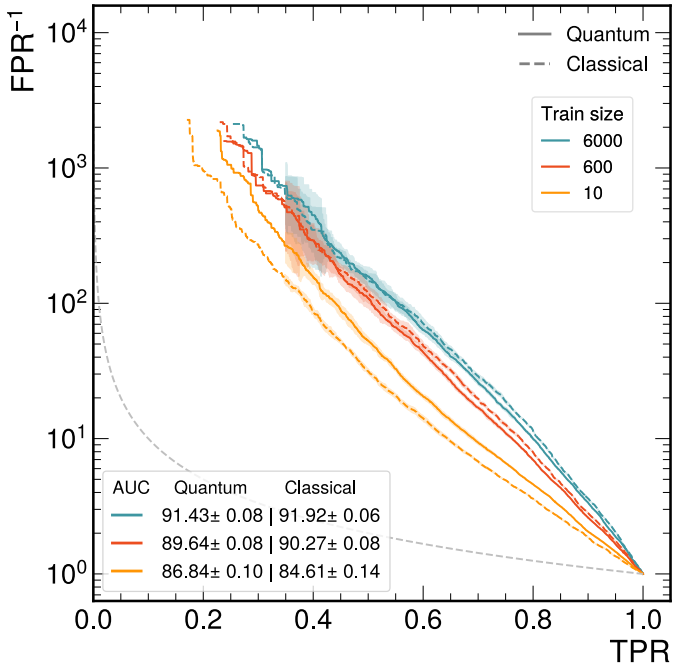


FIGURE 8.8: The ROC curve and the Area Under the ROC curve (AUC) evaluated on different numbers of training samples for quantum and classical clustering algorithms. Here, `Quantum` denotes the quantum-classical clustering method. Signal scenario is set to $A \to HZ \to ZZZ$ and latent space $l = 8$. Error bands are the result of the 5-fold testing procedure. The statistics are very low in the area of lower TPR values, leading to large errors, which are omitted from the plot. The gray dotted line corresponds to taking a random decision.

N_{train}	Method	$\varepsilon_s = 0.6$	$\varepsilon_s = 0.8$
10	Q	20.57 ± 0.97	4.58 ± 0.03
	C	14.12 ± 0.92	3.59 ± 0.01
600	Q	43.48 ± 2.62	6.93 ± 0.13
	C	48.72 ± 3.20	7.83 ± 0.14
6000	Q	63.71 ± 4.57	10.00 ± 0.28
	C	71.16 ± 7.23	11.23 ± 0.16

TABLE 8.2: ε_b^{-1} values (± standard deviation) at fixed ε_s levels, for varying training sizes across quantum and classical algorithms. The higher value indicates better performance. Standard deviations are the result of the 5-fold testing procedure. The shaded area highlights the better performance of the quantum algorithm for the lower training size.

8.7 Conclusion

In this chapter, we presented a realistic pipeline for anomaly detection in the latent space of proton-proton collisions at the LHC. Starting with an encoder for dimensionality reduction, we reduce the complexity of the data while retaining important information that enables the identification of anomalous processes. This pre-processing step also makes the problem more tractable for current quantum algorithms. The core of the pipeline is a k-medians clustering algorithm based on a quantum subroutine for distance computation with a classical optimization method for finding the cluster centroids as medians. Our results indicate that this hybrid approach performs comparably to its classical counterpart, with slight improvements in certain cases. This demonstrates the practical viability of hybrid quantum-classical methods in high-energy physics applications. Although the overall performance gain is modest, the results show that quantum components could be considered for existing machine learning applications, which presents an important step towards applied quantum algorithms for physics analysis. The future integration of quantum algorithms into the LHC real-time selection system still remains an open research question, with engineering and practical utility challenges that go beyond algorithmic performance.

Chapter 9

Matrix Product State for Anomaly Detection at the LHC

Recently, TNs have gained attention in HEP, with their main application surrounding quantum information methods, lattice gauge theories, and classification of b-jets [139–146]. However, TNs are mostly known for their efficiency in representing complex correlations within high-dimensional data. Thus, their growing success in a broader ML context [9, 11, 12, 83] highlights their potential as a powerful tool for HEP-related ML tasks. Recent research efforts have begun to explore tasks such as classification [144] and event reconstruction [147].

The motivation for anomaly detection at the LHC, discussed in the previous chapter, remains central in this study. Building on that, we extend the approach by replacing the quantum-classical clustering algorithm with a TN-based model - Matrix Product State - for anomaly detection in the latent space of dijet events. This TN leverages the Born rule of quantum mechanics to model probability distributions with extension to the continuous data regime [86, 148]. In addition to evaluating the overall performance of the TN model, our goal is to compare it to a quantum kernel method introduced in Ref. [1]. This work also explores the potential for deploying TN-based models in a real-time event selection system at the LHC. The selection system consists of two levels: the Level-1 Trigger (L1T), where selection algorithms are implemented on low-latency hardware such as Field Programmable Gate Arrays (FPGAs), and the High Level Trigger (HLT), which operates on a software-based computing farm. TN-based models designed for efficient inference

at the HLT are required to run within $\mathcal{O}(100)$ms [44]. In this study, we demonstrate that this latency requirement can be met in practice with the TN method. The following sections present the full anomaly detection pipeline, detail the evaluation procedure used to determine the optimal configuration, and discuss the key challenges related to online deployment. The work presented in this chapter is described in Ref. [3].

9.1 Anomaly Detection Pipeline

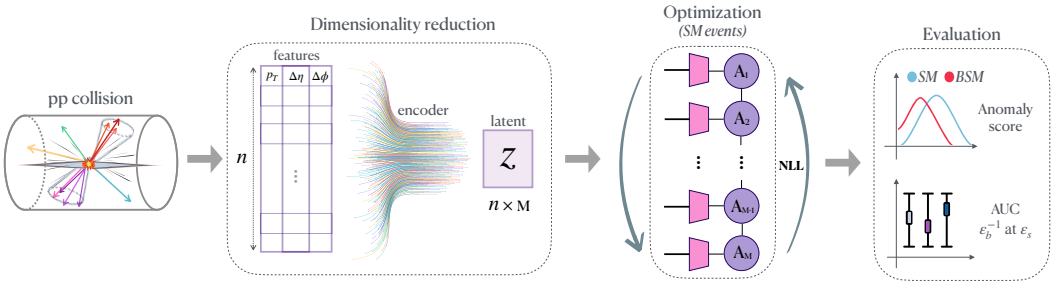


FIGURE 9.1: **TN-based Anomaly detection pipeline**: LHC simulated collision data in matrix format $n \times 3$, where three features $p_T, \Delta\eta, \Delta\phi$ are passed through the trained encoder, generating a latent space of dimensionality M. These latent features are further embedded by isometric feature functions (pink) into a product state. A parametrized MPS model with A_k parameters (purple circle) is contracted with an embedded state to obtain output probabilities. The MPS model is trained on background Standard Model (SM) QCD samples by minimizing the negative log-likelihood (NLL) loss. The output probabilities are serving as anomaly scores. The anomaly detection performance is assessed on a test dataset of standard events and a set of benchmark new-physics scenarios. Evaluation metrics are calculated from the Receiver Operating Characteristic (ROC) curve: Area Under the Curve (AUC) value and background efficiency, ε_b, at signal efficiency, ε_s.

In this section, we describe the anomaly detection pipeline illustrated in Fig. 9.1. The pipeline begins by obtaining low-dimensional latent features from the jet dataset (detailed in Sec. 7.3) using a convolutional autoencoder trained at the *per-particle* level (as discussed in the previous chapter and detailed in Ref. [1]). Latent space features of two jets are stacked together, creating a $2l$-dimensional input vector, further denoted as x. To embed this vector into a TN-based embedding, we employ a *Product State* embedding procedure (explained in Sec. 6.2.1) and apply a Laguerre polynomial feature map with degree n to each feature component x_i:

$$L_n(x_i) = e^{-x_i/2} \sum_{k=0}^{n} \frac{(-1)^k}{k!} \binom{n}{k} x_i^k. \tag{9.1}$$

The final embedded state is in the form of a product state with physical dimension t:

$$\Phi(x) = \bigotimes_{i=1}^{N} L_n(x_i). \tag{9.2}$$

We then introduce a parametrized Matrix Product State (MPS), denoted by:

$$|\psi\rangle = \sum_{\substack{t_1,t_2,\ldots,t_N \\ \chi_1,\chi_2,\cdots,\chi_{N-1}}} A^{t_1}_{\chi_1} A^{t_2}_{\chi_1,\chi_2} \cdots A^{t_N}_{\chi_{N-1}} |t_1 t_2 \cdots t_N\rangle, \tag{9.3}$$

and visualized in Fig. 9.2 whose parameters $A^{(k)^{t_k}}$ are optimized via NLL minimization.

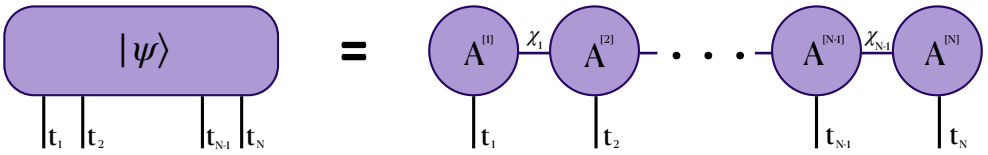


FIGURE 9.2: Matrix Product State model for anomaly detection in the latent space of proton collision events at the LHC. t_i is the physical dimension of i^{th} site. χ_i is the bond dimension between i^{th} and $(i+1)^{\text{th}}$ site.

The NLL loss, over the background Standard Model samples D, is written as:

$$L = -\frac{1}{|D|} \sum_{x \in D} \ln P(x), \tag{9.4}$$

where $P(x) = |\langle \Phi(x) | \Psi \rangle|^2$ is the probability of observing the input x. To ensure numerical stability during training, both the model $|\Psi\rangle$ and the embedded input state $\Phi(x)$ are normalized. The embedded input state and MPS model are visualized as continuous-valued MPS for anomaly detection in Fig. 5.3.

Anomalous events are assigned low probabilities by the trained model, reflecting deviations from the learned distribution of normal behaviour. On the other hand, normal samples have an expected distribution. This framework leads to a clear separation between normal and anomalous samples, allowing one to rank events according to the likelihood value computed under the normal-event hypothesis. Moreover, this method enables direct anomaly detection with fast inference suitable for real-time applications, without requiring any post-processing.

9.2 Training and Evaluation

We used the `tn4ml` library for training and evaluation, as fully described in Chapter 6 and Ref. [2]. Training is performed using mini-batch gradient descent with the Adam optimizer and a learning rate of 10^{-4}, with the training size of $N_{\text{train}} = 10^5$ and testing size of $N_{\text{test}} = 10^5$ per jet. To ensure statistical robustness, each configuration is trained and tested ten times with different random seeds and data shuffling. The final results focus on calculating the Receiver Operating Characteristic (ROC) curve and the corresponding Area Under the ROC Curve (AUC).

Before training, input data is passed through a pre-trained autoencoder, as explained in Chapter 8 and originally proposed in Ref. [1]. We discuss several **stages** important for the selection of the final hyperparameters:

Stage 1 In the first set of experiments, we evaluate anomaly detection performance of the model on BSM signature $A \to HZ \to ZZZ$, with the latent space dimension fixed at $l = 4$, resulting in an input vector of size $2l = 8$. As the autoencoder employs a *tanh* activation function in its final layer of the encoder, the latent features are bounded in the range $(-1, 1)$. Motivated by suggestions from Ref. [86] to use Legendre polynomials for inputs defined over this interval, we adopt this embedding and investigate how different initialization techniques affect the training. For the parametrized MPS model, we set a total of $L = 2l$ tensors and evaluate the performance across bond dimensions $\chi \in \{2, 4, 8, 16\}$. To understand how initialization influences the training stability, we track the NLL loss over time. Fig. 9.3 presents the training curves for four initialization strategies: (1) unitary function; (2) normally distributed random values with standard deviation $\sigma = (\chi_{\max} \cdot d)^{-1}$; (3) normally distributed random values with standard deviation $\sigma = 10^{-2}$; and (4) Gram-Schmidt orthogonalization with normal distribution $\sigma = (\chi_{\max} \cdot d)^{-1}$, where d is the embedding dimension. In Fig. 9.3, different color shades represent individual runs for each bond dimension, which allows us to characterize the variability when setting different random seeds. Models are trained for a maximum of 1000 epochs, with early stopping patience of 30 epochs. The figure reveals clear differences in training stability across various initializers, with larger standard deviations indicating training instabilities and slower convergence. The most stable method is the *unitary initializer*, which we select as the preferred method. This technique initializes the tensors as stacks of unitary matrices.

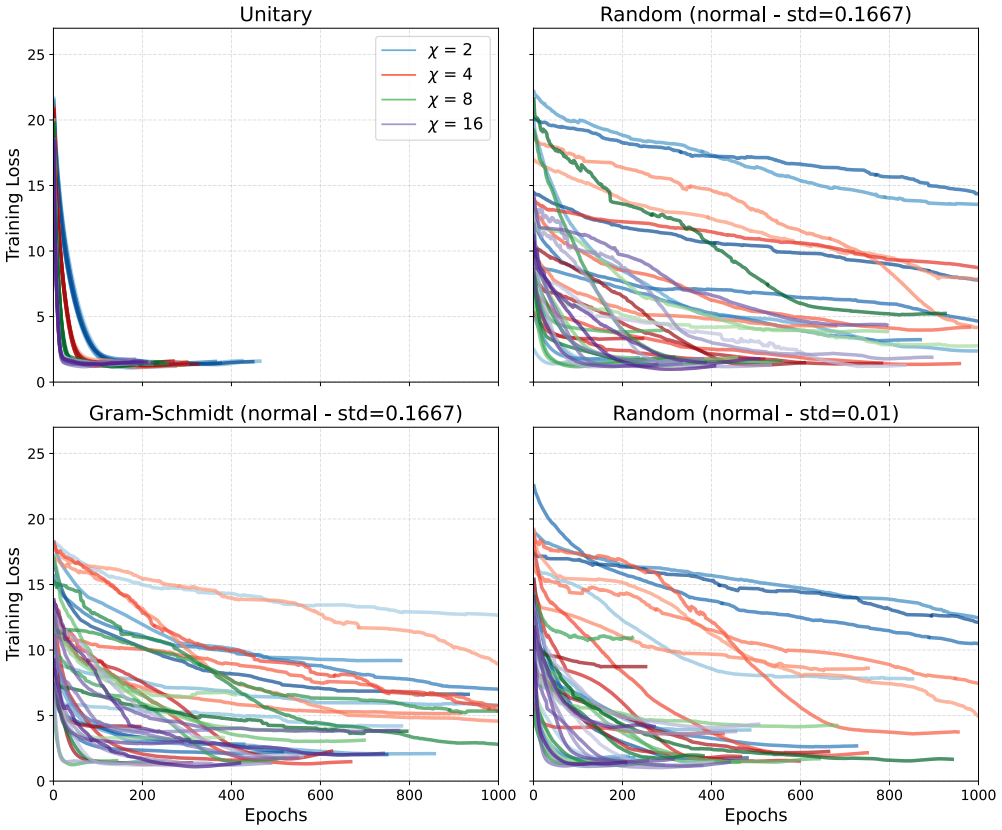


FIGURE 9.3: The distribution of Negative Log-Likelihood loss over number of epochs for each initializer technique and bond dimension χ using a Legendre feature map of degree two.

Stage 2 To validate our choice of embedding map, we also evaluate two alternative feature maps: Hermite polynomials (defined over the entire space of $\mathbb{R}$), and Laguerre polynomials (defined on the non-negative axis), both with the degree of expansion fixed at two. To make the inputs compatible with the Laguerre map, we rescale them to the interval $[0, 1]$. Anomaly detection performance is assessed by comparing the distributions of the output probabilities for Standard QCD events and Beyond Standard Model (BSM) signal events, as shown in Fig. 9.4, across different bond dimensions χ and embedding methods. The plots indicate that the *Laguerre* polynomial embedding achieves the best separation between signal and background processes.

Stage 3 As a final step in selecting the model's optimal hyperparameters, we compute the ROC curve to quantify the model's discriminative power. From the ROC curve we calculate corresponding AUC score and track HEP-relevant metrics - inverse background efficiency (ε_b^{-1}), or False Positive Rate (FPR) at the specific value of signal efficiency (ε_s), or True Positive Rate (TPR), $\varepsilon_s \in \{0.6, 0.8\}$, following the evaluation from previous

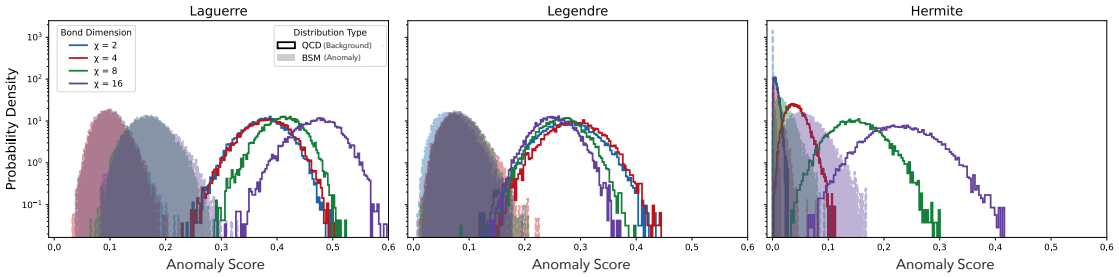


FIGURE 9.4: Histograms of anomaly score distributions for QCD (solid line) and BSM (filled histogram) across different embedding functions - Laguerre (left), Legendre (center), and Hermite (right). Each distribution shows performance for different bond dimensions $\chi \in 2, 4, 8, 16$.

chapter 8. From these results, shown in Fig. 9.5, we observe that the best performance for latent space $l = 4$ is obtained with a bond dimension of $\chi = 2$. Although larger bond dimensions theoretically offer greater expressive power of the model, our analysis demonstrates that this is necessarily the case when the number of tensors in a network is small. In such settings, models with higher χ tend to overfit on the training data, resulting in poorer generalization on the test dataset. In contrast, for higher latent space dimensions ($l = 8$ and 16), increasing bond dimension shows a positive trend up to a peak performance (in both cases for bond dimension $\chi = 32$). Additionally, we investigated the impact of increasing the polynomial degree and found no performance gain, thus, we decided to use degree two in the final configuration.

Final evaluation After fixing hyperparameters of the MPS model to $L = 2l = 8$, using the *unitary* initializer, the *Laguerre* polynomial feature map and a bond dimension of $\chi = 2$, we evaluate the model across various data-dependent factors: three new-physics (BSM) scenarios and multiple latent space dimensions. The goal is to understand how data compression affects the performance and how well the model performs on different types of anomalous signals. Since the AE can detect anomalies independently by evaluating reconstruction error, we use its performance as a baseline. In the following analyses, we compare the performance of the clustering algorithm to that obtained from the AE.

New-physics signal comparison Fig. 9.6 presents the ROC curves for three benchmark BSM signals, illustrating different performance across different signal types (introduced in Sec. 7.3). This is an inherent characteristic of the nature of each signal process - for instance, the broad Graviton signal with light center-of-mass energy is the most challenging to distinguish from the background due to its similarity to SM events, and the

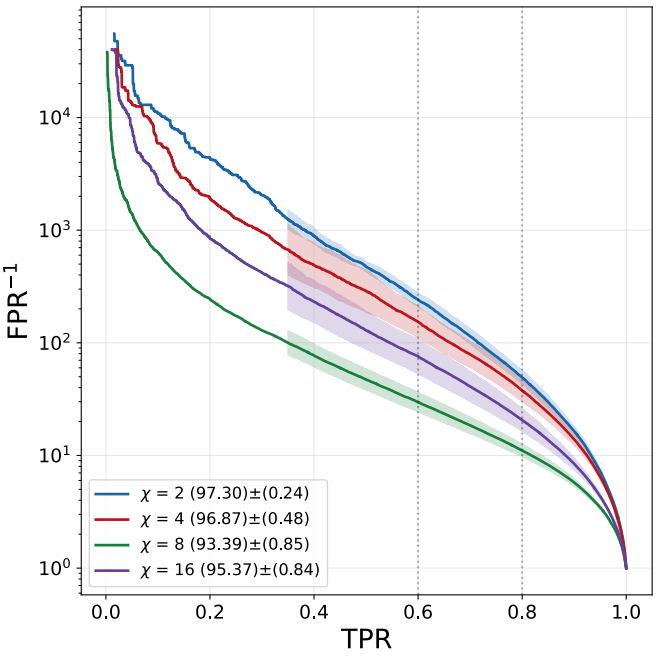


FIGURE 9.5: ROC curve for the MPS model with $L = 2l = 8$ tensors with *unitary* initializer, *Laguerre* polynomial feature map and bond dimensions $\chi \in \{2, 4, 8, 16\}$. Shaded regions around each ROC curve indicate repeated testing with different random seeds for initialization and data shuffling. In areas of low TPR values, large error bands are removed from the plot for smoother analysis.

narrow Graviton process is the most anomalous. An interesting observation is that, with a latent space of $l = 4$, our model achieves performance comparable to that of the quantum kernel method (QKM) in Ref. [1], which used a latent dimension of $l = 8$. The QKM employs $n_q = l$ qubits and implements three layers of data re-uploading with nearest-neighbor entanglement. This comparison highlights that our model achieves competitive or improved results with lower input dimensionality and a simpler model architecture. Additionally, our model outperforms the quantum-classical clustering algorithm from the previous Chapter 8. Particularly, the notable result comes from the performance on the signal Broad $G \to WW(1.5\,\text{TeV})$, where our model reaches an AUC of $69.49 \pm 0.85\%$, outperforming the QKM's $47.62 \pm 0.52\%$ AUC for $l = 8$, corresponding to an absolute gain of 21.87 percentage points. When comparing with AE results, in both AUC value and ε_b^{-1} at $\varepsilon_s \in \{0.6, 0.8\}$, the results show improvement for each BSM scenario when using the TN-based model to detect anomalies.

Latent dimension comparison The performance of the model on different latent spaces is evaluated for the signal scenario $A \to HZ \to ZZZ(3.5\,\text{TeV})$. Each latent space dimension performs the best for different values of bond dimension, choosing for

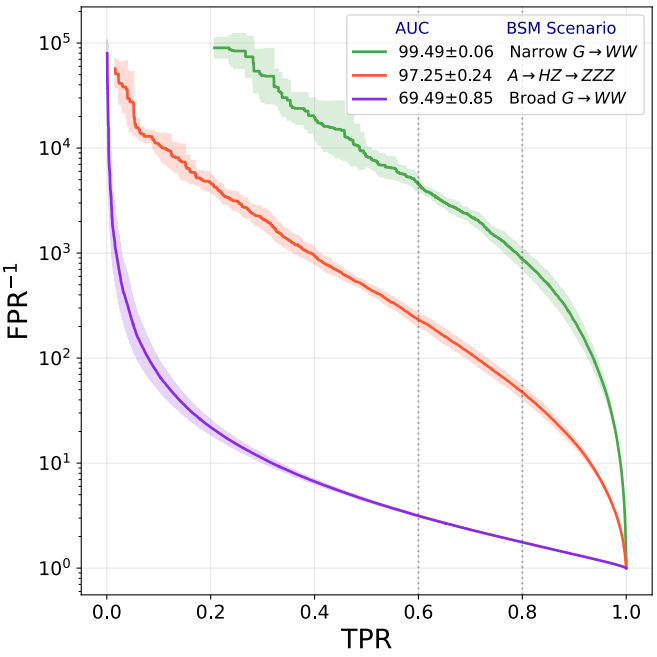


FIGURE 9.6: ROC curves showing performance of the MPS model - latent space $l = 4$ with unitary initializer, Laguerre polynomial feature map and bond dimension $\chi = 2$ - evaluated on different new-physics signals. Shaded regions around each ROC curve indicate repeated testing with different random seeds for initialization and data shuffling. In areas of low TPR values, large error bands are removed from the plot for smoother analysis.

$l = 4 \rightarrow \chi = 2$, $l = 8 \rightarrow \chi = 32$ and $l = 16 \rightarrow \chi = 32$. In Fig. 9.7, we observe that as model capacity increases, the performance saturates, obtaining the best results with latent dimension $l = 8$ and a bond dimension of $\chi = 32$.

To study important metrics for HEP analysis, Table 9.1 shows values of ε_b^{-1} at signal efficiencies $\varepsilon_s \in \{0.6, 0.8\}$ for latent space dimensions $l \in 4, 8, 16$. Firstly, the table supports results from Figs. 9.7 and 9.6 for the BSM scenario $A \rightarrow HZ \rightarrow ZZZ$, confirming a positive trend up to a peak of performance. Additionally, the table includes numerical results for the new-physics scenario Broad G → WW, which achieves a peak of performance already for latent space $l = 4$ and bond dimension $\chi = 2$. This additionally highlights the earlier observation that increased model complexity does not necessarily lead to improved performance, as even highly anomalous signatures can be effectively detected by simpler architectures. Compared to the AE's performance across both signal types and all latent space dimensions in Table 9.1, the addition of the MPS model in the anomaly detection pipeline leads to improved results across all evaluated metrics. Although a direct comparison with the p-VAE model from Ref. [134] is not entirely fair due to differences in the number of training and testing samples, the AE-MPS anomaly detection pipeline still

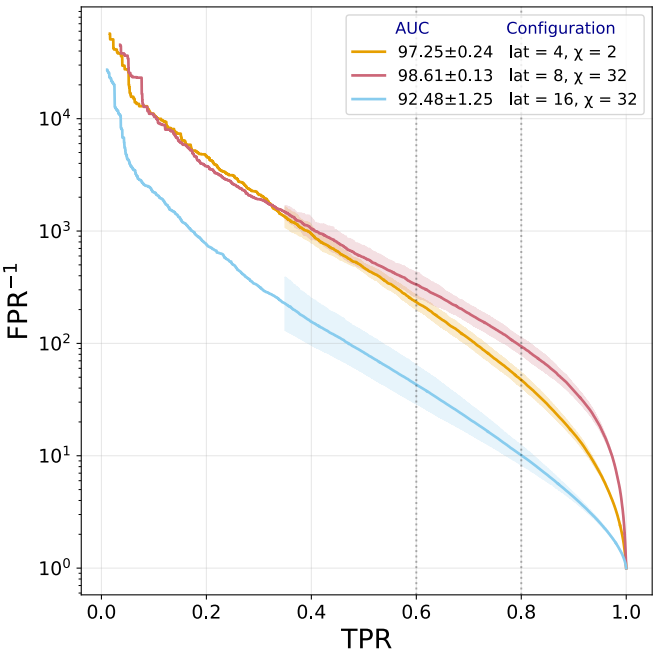


FIGURE 9.7: ROC curves showing performance for the MPS model - unitary initializer, Laguerre polynomial feature map - for different latent dimension $l \in \{4, 8, 16\}$. Shaded regions around each ROC curve indicate error bands from repeated testing with different random seeds for initialization and data shuffling. In areas of low TPR values, large error bands are removed from the plot for smoother analysis.

shows competitive performance. These results highlight the potential of using the more lightweight MPS model to replace the full VAE decoder structure while maintaining the anomaly detection capabilities.

$A \to HZ \to ZZZ$

l	χ	$\varepsilon_b^{-1}(\varepsilon_s = 0.8)$	$\varepsilon_b^{-1}(\varepsilon_s = 0.6)$
4	2	47±8	233±33
8	32	94±17	334±85
16	32	10±3	44±18

BR G $\to$ WW

l	χ	$\varepsilon_b^{-1}(\varepsilon_s = 0.8)$	$\varepsilon_b^{-1}(\varepsilon_s = 0.6)$
4	2	1.76±0.02	3.15±0.12
8	32	1.55±0.06	2.32±0.15
16	32	1.39±0.04	1.92±0.12

TABLE 9.1: Comparison of performance metric ε_b^{-1} at fixed signal efficiencies $\varepsilon_s \in \{0.6, 0.8\}$ for different model configuration (latent space l, bond dimension χ) and anomalous signatures. Standard deviations for each result are obtained from five cross-validation runs with different random initializations and data shuffling.

Runtime execution We execute the inference of the full pipeline per one event, consisting of the encoder and the MPS model. Inference times per event, shown in Fig. 9.8, are compared for three models from Fig. 9.7 on a single CPU core of an Intel Core i5-9600KF (3.7GHz base, 9MB L3 cache) with 16GB DDR4 RAM. Each run is characterized by the total number of parameters of the pipeline and the runtime per event in microseconds (ms). AE predominates the execution time, with 95% contribution, leaving a small additional cost taken by the TN model. However, even the pipeline with the largest latent space $l = 16$ is suitable for deployment in a typical LHC HLT system, with the runtime of 3.8ms,. The potential reduction in latency could come from the execution in the CMS HLT farm, which has CPUs such as Intel Xeon processors with higher memory bandwidth, larger cache sizes and increased parallel throughput. Executing such an algorithm on an FPGA with $\mathcal{O}(100)$nsec latency and an acceptable consumption of computational resources (e.g., look-up tables, flip-flops) is still to be demonstrated. This motivates further exploration of model compression techniques such as pruning or quantization, while considering the trade-off between model accuracy and computational efficiency.

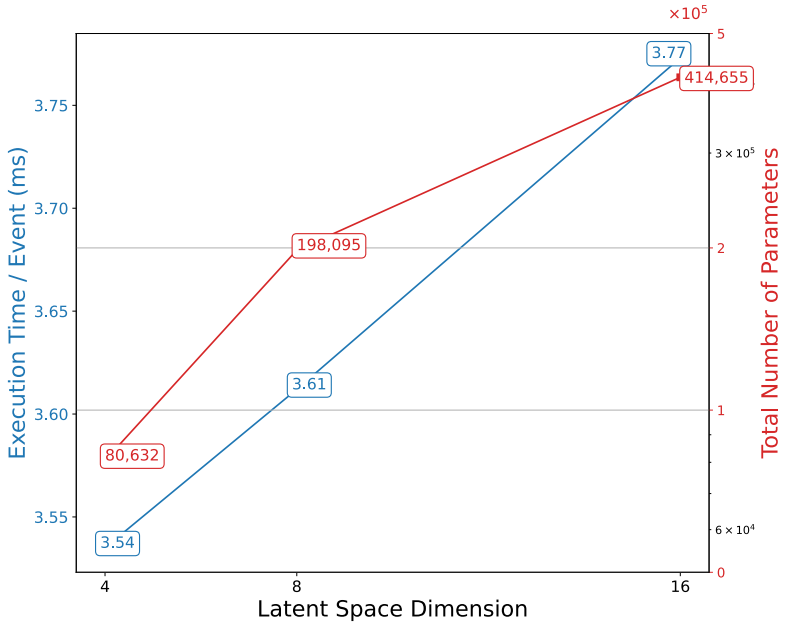


FIGURE 9.8: Execution time per batch (ms) and total number of model parameters as a function of latent space dimension. As latent space increases from 4 to 16, both execution time (from 3.54 ms to 3.77ms) and model complexity ($\sim$ 80k to $\sim$ 414k) increase.

9.3 Conclusion

This study presents a novel application of TNs for anomaly detection of novel physics processes in proton-proton collision events at the LHC. A key contribution of our work to TN applications in ML includes the development of a thorough optimization pipeline for the MPS model, consisting of: (1) input data analysis, (2) choice of the most suitable embedding technique, (3) selection of the tensor initializer function, and (4) evaluation of performance through data-driven factors and model complexity. All this resulted in designing an MPS model that learns the probability distribution of the QCD background samples and assigns low probability to unknown new-physics signatures. When compared to the previous study of quantum-classical clustering (discussed in Chapter 8) and the quantum kernel method (from Ref. [1]), our model with a unitary initializer and Laguerre feature map can achieve competitive results, even when trained on a smaller latent representation and using lower model complexity. Its efficient performance in computationally low environments offers a possibility of future deployment in the software-based event selection system at the HLT, and further motivates exploration of model compression techniques that would allow for a faster enough algorithm to be implemented on the FPGAs. This study opens a path towards exploring the practical utility of quantum-inspired machine learning algorithms in HEP.

Chapter 10

Tensor Network for Classification of Jet Substructure at the LHC

In this chapter, we present a preliminary study that serves as a baseline for integrating TN models in the Level-1 Trigger (L1T) system of the CMS detector at the LHC. Jet substructure classification is a commonly studied problem in particle physics, as it enhances the detector's sensitivity to potential new physics by filtering the background processes, thereby improving the precision of measurements [149]. Traditional jet classification techniques recently employed in the detectors at the LHC, heavily rely on theoretical assumptions and features engineered by expert knowledge [150, 151]. However, deep learning approaches have shown improved performance by leveraging low-level data features without the significant loss of information [149, 152, 153]. In the following sections, we outline the motivation for and challenges of designing a machine learning algorithm suitable for deployment in the LHC's trigger system, along with the theoretical context and relevance of the problem. We then motivate our choice of TN-based model architecture and detail the training process. Finally, we describe the evaluation strategy and lay the foundations for future practical deployment of this model on hardware accelerators.

10.1 Motivation and Challenges

Jets are collimated sprays of particles originating from partons - quarks and gluons - produced in high-energy collisions, such as those at the LHC. Through the process of hadronization, these partons transform into hadrons - composite particles like protons and neutrons - that are finally detected by various detector subsystems. By combining these inputs from subdetectors, these particles are reconstructed into jets using clustering algorithms. Identifying the origin (or flavor) of a jet - whether it was built from a light-flavor quark, gluon, heavy-flavor quark, or a decay of a heavy particle like W, Z, Higgs boson, or top quark - is very important to many physics analyses. This classification task, known as *jet tagging*, is crucial for improving trigger decisions by increasing the accuracy of physics measurements and enhancing the sensitivity of searches for new-physics signatures by helping to suppress backgrounds.
Designing ML algorithms for efficient jet tagging in real-time at the L1T introduces several challenges:

- restricted access to a small subset of reconstructed particles,
- limited and time-consuming sorting operations require algorithms invariant to permutations of features (due to jets being unordered sets of a variable number of particles),
- latency of $\mathcal{O}(100\text{ns})$ per decision for integration in the High-Luminosity upgrade of the LHC (HL-LHC).

The overview of the trigger system in the CMS detector is detailed in Appendix C. Traditional models were based on vertex reconstruction and physics-motivated features to reduce complexity, but modern ML algorithms need to be expressive and have fewer parameters to operate in a low-latency environment. A high-level view on the pipeline for developing and deploying these algorithms at the L1T consists of designing a well-defined ML model based on particle constituents, and porting it to the FPGA with specialized libraries, such as `hls4ml` [154]. The quality of these algorithms is assessed through the evaluation of their classification performance, latency, throughput, and resource consumption. Successful ML algorithms using only particle data as inputs have been proposed in the past [115, 155–159], some of them being designed specifically with permutation-invariance [149, 155, 160].

Recent work in Ref. [8] demonstrated the feasible practical deployment of Tree Tensor Networks (TTNs) in the LHC trigger system, which motivated our development of an MPS-based model - usually more lightweight and easier to optimize.

In this work, motivated by the expressive power of TN-based models and their ability to leverage the quantum nature of physics data [1, 161], we design a Matrix Product State for the classification of jet substructure using particle-specific features, emphasizing the importance of data embedding for successful design. We evaluate the model on data-related parameters and show the impact of model complexity on the final performance.

10.2 Classification pipeline

This section introduces the final setup for the classification pipeline used for *jet tagging*, consisting of (1) data embedding technique, (2) TN model description, and (3) training pipeline.

10.2.1 Data Embedding

From the low-level feature of the Jet Dataset described in Sec. 7.2, we choose n significant features and proceed with the embedding on the particle level. The input is described in a format $N \times n$, where N is the number of particles, and n is the number of features per particle. This means that each tensor in the TN model is modeled to contain information about one particle (similarly done in the QML application for anomaly detection in HEP [161]). For a feature function of each particle, we use a polynomial embedding function (described in Sec. 6.2.1) such that it expands a particle input $[x_1, ..., x_n]$ into a vector of polynomial features, including interactions and powers of input components up to a *degree* of two. A constant bias term of 1.0 is included. This allows a TN model to capture non-linear patterns in the input data. The feature function $\phi(x^{(i)})$ expands the i-th particle input x^i to a vector of dimension d. The mathematical expression of the *particle embedding* for one particle x is described as:

$$\phi(x) = \left[1,\ x_1,\ x_2,\ \ldots,\ x_n,\ x_1^2,\ x_1x_2,\ \ldots,\ x_n^2,\ \ldots,\ \prod_{j=1}^{k} x_{i_j}\right] \quad , \tag{10.1}$$

for all $k \in [1, \textit{degree}],\ i_j \in \{1, \ldots, n\}$.

The total embedding is expressed as a product state of each *particle embedding*:

$$\Phi(x) = \bigotimes_{i=1}^{N} \phi_i(x^{(i)}), \tag{10.2}$$

as visualized in Fig. 10.1.

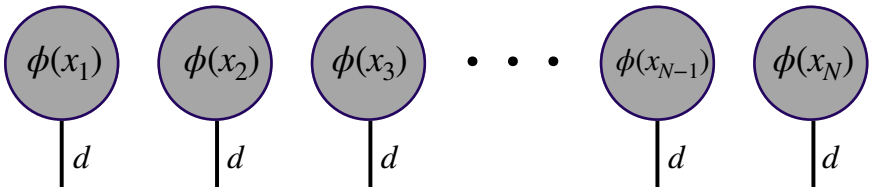


FIGURE 10.1: Graphical representation of a product state describing the embedded input where function ϕ embeds each particle input vector x_i into a polynomial expansion vector with dimension d for N particles.

10.2.2 Tensor Network Model

The model used for the classification task is an MPS with an additional output index in the middle of the chain, having dimensionality equal to the number of classes. The model is mathematically described as:

$$|\psi_c\rangle = \sum_{\substack{t_1,\ldots,t_N \\ \chi_1,\cdots,\chi_{N-1}}} A^{t_1}_{\chi_1} A^{t_2}_{\chi_1,\chi_2} \cdots A^{t_k,c}_{\chi_{k-1},\chi_k} \cdots A^{t_N}_{\chi_{N-1}} |t_1 t_2 \cdots t_N\rangle, \tag{10.3}$$

where $c \in \{1, ..., C\}$ and C is number of classes. Trainable parameters of the MPS model are described with the tensor $A^{t_k}_{\chi_k,\chi_{k+1}}$, where t_k is a physical index with dimension dictated by the embedding procedure, in this case generally noted as d, χ_k is the bond dimension connecting i^{th} and $(i+1)^{\text{th}}$ tensor, and $1 < k < N$ where N is the number of tensors in MPS chain. The Eq. (10.3) is graphically illustrated in Fig. 10.2.

10.2.3 Training Procedure

Input samples are paired as tuples of input vectors and labels $(x, y_t) \in \mathcal{D}$ where $\mathcal{D}$ is the dataset described in Sec. 7.2. The parametrized MPS model is trained with Mini-batch gradient descent by minimizing a *cross-entropy* loss, as is usually done in multi-classification tasks. The loss value is obtained by contracting the product state $\Phi(x)$ with the MPS model $|\psi_c\rangle$, resulting in an output vector y_c of dimension C, where C is the

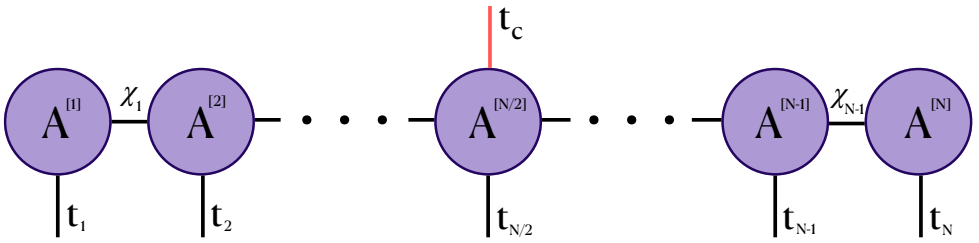


FIGURE 10.2: Matrix Product State model for classification of jet substructure at the LHC. t_i is the physical dimension of i^{th} site. χ_i is the bond dimension between i^{th} and $(i+1)^{\text{th}}$ site. t_c index has dimensionality equal to the number of classes C.

number of classes. The vector y_c is passed through a *softmax* function to obtain class probabilities. The contraction scheme is illustrated in Fig. 10.3. The final loss value is

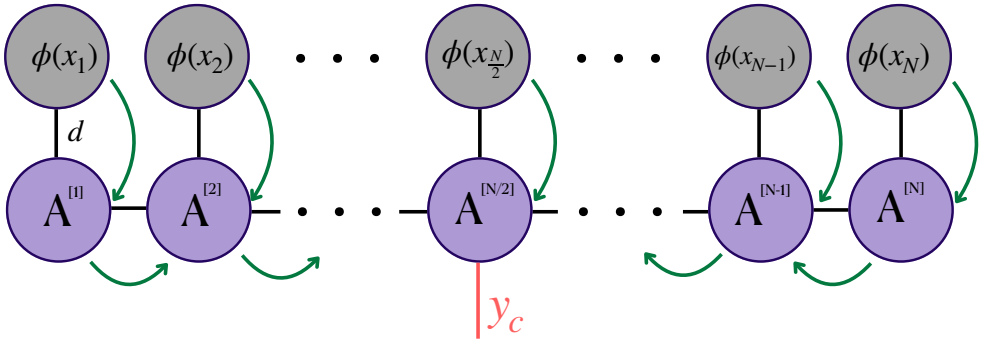


FIGURE 10.3: Graphical representation of contraction scheme between the embedded input state $\Phi(x)$ and the parametrized MPS model $|\psi_c\rangle$, resulting in the output vector y_c. The contraction scheme is not necessarily optimal, just schematic.

obtained by comparing the probability scores returned by the TN to the corresponding ground truth y_t for that specific input x:

$$\mathcal{L} = -\sum_{i=1}^{N} y_{\text{t}}^{(i)} \log(\text{softmax}(y_{\text{c}}^{(i)})). \tag{10.4}$$

10.3 Performance Evaluation

The performance of the MPS model is compared with results from the Ref. [149], where the authors compared several deep learning techniques in terms of classification accuracy, number of parameters, the AUC value for each class type, and FPR at the fixed TPR of 80%.

Training Setup For each jet, we choose the N highest-p_{T} particles, ensuring $p_{\mathrm{T}} > 2\,\mathrm{GeV}$ as motivated by the CMS tracking capabilities at the L1T for HL-LHC [162]. We shuffle the particles to mimic the HL-LHC L1T environment, where particles are unordered, as done in Ref. [149]. We used the standardization method applied on the same dataset in Ref. [149], which normalizes each feature distribution to its $[5, 95]\%$ interquantile range for increased robustness against outliers. Additionally, in the case of jets with less than N constituents, data are zero-padded up to N. The tensors of the MPS model are initialized as normally distributed random values with standard deviation $\sigma = 10^{-2}$. The MPS model is trained using the Mini-batch gradient descent procedure (detailed in Sec. 6.2.3) with the Adam optimizer and learning rate 10^{-4}. First, we perform an analysis of various embedding techniques to find those most suitable for our application. Then, we determine the optimal bond dimension, and finally, we evaluate how the number of particles in the input data affects the performance.

Data Embedding Experiments Based on the distributions of low-level features visualized in Fig. 7.2, we choose to first explore different embedding techniques using the features $f = [E, E^{rel}, p_T, p_T^{rel}, \Delta R]$. To find the best embedding setup, we fix the number of particles to $N = 8$, and perform a 5-fold cross-validation procedure with the total training size of $10,000$ and the batch size set to 128. Table 10.1 shows the list of all embedding strategies we explored with their corresponding test accuracy. When not stated, the bond dimension χ is set to 10.

#	**Embedding Description**	Accuracy (%)
1	flatten $N \times f$ + trigonometric	46.2 ± 0.4
2	flatten $N \times f$ + poly$(d = 2)$	54.6 ± 0.2
3	flatten $N \times f$ + poly$(d = 5)$	52.9 ± 0.6
4	Eq. (10.1)$([e^{rel}, p_T, p_T^{rel}, \Delta R]]$	59.5 ± 0.3
5	flatten $N \times [e^{rel}, p_T, p_T^{rel}, \Delta R]$ + poly$(d = 2)$	56.3 ± 0.2

TABLE 10.1: Comparison of different embedding strategies and their test accuracy. Standard deviations are the result of a 3-fold cross-validation strategy. The best result is highlighted in gray.

From these experiments, we decided to use the embedding of Eq. (10.1), where one tensor represents one particle, and a polynomial embedded vector corresponds to the polynomial expansion. We also decided to only keep three features $[E^{rel}, p_T, \Delta R]$, reducing further the number of parameters without causing a significant performance degradation.

Bond dimension optimization To find the most suitable bond dimension χ, for $N = 8$ particles, we set the embedding method as above, used three features $[E^{rel}, p_T, \Delta R]$, and run a 3-fold cross-validation with training size of $10,000$. Table 10.2 displays results for $\chi \in \{2, 5, 10, 50\}$ and shows that the best result is obtained with bond dimension $\chi = 10$.

χ	Test Acc (%)
2	55.63 ± 1.82
5	60.65 ± 0.60
10	60.79 ± 0.40
50	55.93 ± 0.43

TABLE 10.2: Test accuracy as a function of bond dimension χ. Standard deviations are the result of a 3-fold cross-validation strategy. The best result is highlighted in gray.

Final configuration We fix the bond dimension $\chi = 10$ across all choices of number of particles $N \in \{8, 16, 32, 50\}$ in a jet, using input features $[e^{rel}, p_T, \Delta R]$, and embedding scheme defined in Sec. 10.2.1. The model is trained on 496,000 samples, validated on 124,000, and tested on an independent dataset of 260,000 samples. The batch size is set to 2048 and the model is trained for 400 epochs with early stopping patience of 50 epochs. We observe that increasing the training set size results in an improved test accuracy for all model sizes, highlighting the benefit of training on a larger amount of samples. To evaluate the final classification performance, we present results in Table 10.3 for the considered values of N and AUC scores for each class, following Ref. [149]. The uncertainties for accuracy and AUC correspond to the square root of the mean square error across the three folds, but they are negligible for AUC values across all classes. Overall, our MPS-based model achieves competitive results compared to deep learning approaches in floating-point precision from Ref. [149], especially for a lower number of constituents $N = [8,\ 16]$ with statistically significant improvement in terms of accuracy. An increasing number of particles has a positive impact on the performance, with $N = 50$ achieving the best results.

To support this table, Fig. 10.4 shows ROC curves for each number of particles N across all classes with corresponding AUC value and FPR (misidentification rate) at 80% TPR.

Fig. 10.5 shows the inverse of average FPR for all five classes at 80% TPR as a function of the number of constituents. The error bands correspond to the RMS of the 3-fold

N	#params	Acc(%)	AUC				
			g	q	W	Z	t
8	10200	66.1 ± 0.10	0.89	0.86	0.87	0.85	0.91
16	18200	72.0 ± 0.03	0.90	0.88	0.92	0.89	0.93
32	34200	74.8 ± 0.10	0.91	0.88	0.94	0.92	0.94
50	52200	75.9 ± 0.01	0.91	0.89	0.94	0.93	0.93

TABLE 10.3: Test accuracy and AUC scores for different numbers of particles N in the input events. The error bands are the result of a 3-fold cross-validation strategy, and are not included in the table for AUC values, as values range from 0.0001 to 0.001. The best result is highlighted in gray.

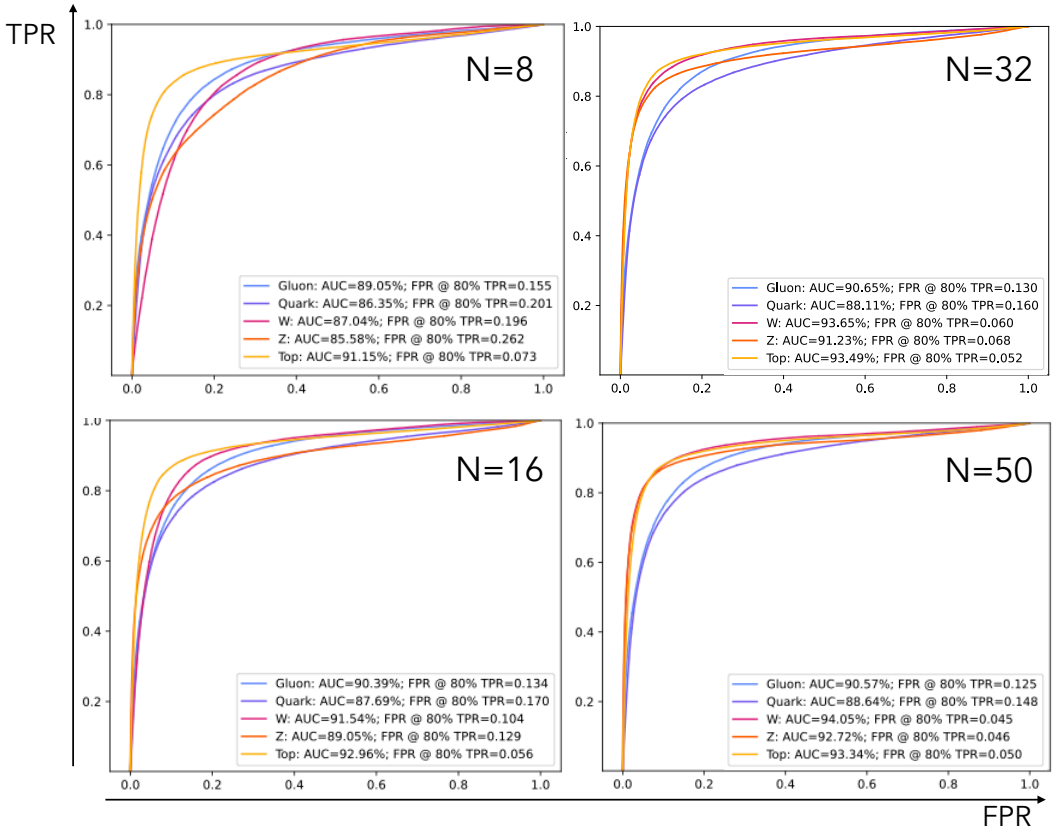


FIGURE 10.4: ROC curves representing performance of the classification model for the number of particles $N = \{8, 16, 32, 50\}$. Performance of the classifier for each class Gluon, Quark, W, Z, Top is measured with the corresponding AUC value and FPR at 80% TPR value.

cross-validation. The floating-point models for $N = \{8, 16\}$ are performing better than the 8-bit deep learning models from Ref. [149], as expected, except for the model with $N = 32$. To prove the practical utility, the performance should be kept consistent even with the quantized version of the model. This is something that is explored for the future preparation of the collaborative manuscript.

When comparing with state-of-the-art deep learning models in Ref. [152] and [149],

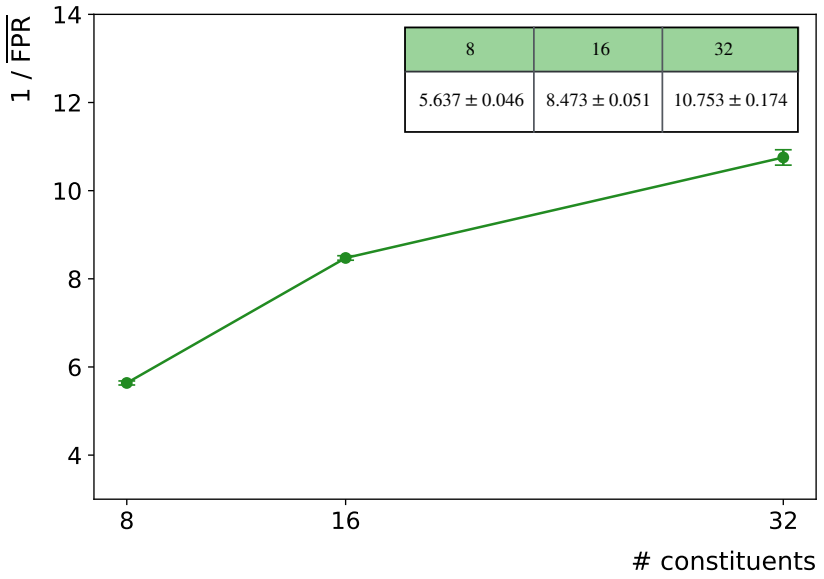


FIGURE 10.5: Inverse of Average False Positive Rate across all classes at 80% True Positive Rate as a function of number of constituents.

our results show competitive performance for three input features per particle. This establishes a promising path for future efforts in deploying TN-based models onto FPGAs, which will be addressed in future studies.

10.4 Implications for Real-Time Deployment

Inspired by the recent work from Ref. [8] on deploying TTNs to the FPGAs, our future work aims to apply similar techniques to enable the practical deployment of the MPS-based model. The targeted FPGA will be the Ultrascale FPGA KCU1500. Before the actual deployment, it is necessary to perform analyses, such as exploring quantum information tools, e.g., the entropy profile, to determine where the model concentrates its attention. This could contribute to parameter-reduction techniques and improve efficiency. Additionally, it will be necessary to perform model quantization, either through post-training quantization or quantization-aware training, which requires extending the `tn4ml` library to enable quantization of TN models. The quantized model would be ported to an FPGA using a specialized library, followed by evaluation of performance in terms of classification accuracy, FPR at fixed TPR, latency, throughput, and hardware resource usage. If the latency does not exceed $\mathcal{O}(100)$ns, the model could be potentially useful for online deployment in the LHC trigger.

10.5 Conclusion

In this chapter, we presented a study of exploring the Tensor Network model, specifically MPS, for the jet classification problem with the future goal of deployment at the LHC's L1 trigger system. Focusing on the jet tagging task, our goal was to develop an expressive and lightweight architecture with particle-inspired embeddings to enable physics-motivated structure compatible with FPGA requirements. This physics-aware design offers the possibility of integrating domain knowledge into trigger algorithms. Through various training and evaluation experiments on the jet substructure dataset, we demonstrated that our approach yields competitive performance when compared to the deep learning models. This study is a foundation for future work on applying quantum-inspired models for online event selection and real-time inference in HEP.

Chapter 11

Discussion

11.1 Reflections on the Research Journey

The intersection of quantum computing, machine learning and high-energy physics offers numerous research opportunities, as each field is conceptually and technologically challenging. Together, these three fields provide a rich set of opportunities for exploring new research directions. This thesis was shaped by a focus on the practical utility of this intersection in the near term. This included an iterative process of design, development, and implementation, with failures and successes, similar to any research domain.
The research journey began with a set of theoretical questions:

1. When do quantum components offer relevant benefits within classical algorithms?
2. Could quantum algorithms become practically deployable in the LHC in the near future?
3. Do Tensor Networks outperform quantum-enhanced methods in realistic settings?
4. Can TNs deliver competitive results for real-time ML tasks at the LHC, while maintaining efficiency and explainability?
5. What if we could train Tensor Networks like neural networks - while also understanding the contribution of each component of the pipeline?

All these questions remain central throughout the whole research process, and are addressed in the work presented in this thesis. One of the most important insights comes

from the modular nature of hybrid or TN-based approaches: each component - whether quantum, quantum-inspired or classical - from embedding to optimization schemes, can be independently replaced, tested and interpreted. This modularity enables robust insights about model performance, structural efficiency and, in certain cases, physical relevance. Looking forward, I hope this work contributes to a continued effort toward efficient and explainable ML models in physics, and to a constant examination of the practical utility of existing quantum algorithms in applied science.

11.2 Key Takeaways on Results

This thesis explored hybrid quantum-classical pipeline and Tensor Network methods, as representatives of quantum-inspired approaches, in various applied machine learning scenarios. The initial motivation for researching quantum-based approaches in physics analysis arose from the quantum nature of HEP datasets, and was extended to analyzing the performance of quantum-inspired yet classical methods as more practical models for near-term deployment in the LHC systems.

The development of the hybrid clustering approach for anomaly detection showed that, in particular settings, the tradeoff between classical and quantum components of an algorithm is motivated by computational efficiency and overall algorithmic performance. This revealed important insights when quantum components of an algorithm provide true benefits versus when they provide unnecessary complexity (see Fig. 8.5). The studies presented in Sec. 8.6 demonstrate that a classical pipeline, enriched with quantum components, shows comparable performance to its pure classical counterpart. This motivates further studies for the development of quantum counterparts comparable to the state-of-the-art classical algorithms at the LHC experiments.

The motivation for seeking practical methods with quantum-enhanced structure and challenging established quantum methods has led us to explore TN architectures for various ML tasks in HEP analyses. The comparison between the TN-based approach and the hybrid clustering algorithm for anomaly detection revealed superior performance of the former, which challenged the practical utility of the quantum components in the hybrid approach. This has been proven from the comparison of two performance analysis: (1) Figures 8.6 and 9.6, and (2) Figures 8.7 and 9.7, each numerically supported by evaluation

metrics in Tables 8.1 and 9.1. Furthermore, the TN architecture proved to be executable in the real-time selection pipeline of the HLT at the LHC, which demonstrated even greater utility (see Fig. 9.8).

The final study of this thesis demonstrated that the TN model can be compared favorably to the deep learning techniques (refer to Table 10.3), and left room for future development with the possibility of designing a portable model for the L1 trigger, where the latency and resource constraints are stricter.

This work established a foundation for future efforts to explore the practical utility of quantum or quantum-inspired algorithms at the LHC, providing both successful implementations and important insights into their limitations and challenges.

11.3 Limitations and Challenges

While this work shows the potential and feasibility of hybrid quantum-classical approaches and TN methods in HEP ML applications, several limitations and challenges are still present. Addressing these in future work would be necessary for scaling and deploying these models in realistic experimental setups.

Quantum Integration and Scalability

Although quantum subroutines can be integrated successfully into classical pipelines, they were tested in simulation with a limited number of qubits. The practical deployment of such methods in near-term quantum hardware is currently constrained by qubit fidelity, gate depth, noise, etc. All this would affect the precision of the final measurements. Moreover, quantum components in this setting still have not demonstrated clear superior performance. Scaling quantum-enhanced models to higher-dimensional data still remains an open research challenge.

Expressivity and Training of Tensor Networks

Matrix Product States, the TNs considered in this work, are inherently limited by their 1D structure, which does not allow them to efficiently model long-range correlations and

non-sequential data. Although in our work, MPS models showed strong performance, the tradeoff between expressivity and efficiency becomes more apparent in settings with higher-dimensional correlations. Moreover, TN performances are still largely dependent on the choice of many hyperparameters, such as initialization functions, data embedding schemes, and model size hyperparameters (e.g., bond dimension). The optimal choice for these parameters is typically specific to the considered problem. In the case of optimization of classical neural networks, the computational weight of hyperparameter tuning is well addressed by many specific libraries, which make the process efficient. The same doesn't apply to TN training. With `JAX` as a backend, `tn4ml` aims to address the hyperparameter tuning in TN training by leveraging the progress made for deep neural networks.

Data Embedding for Tensor Networks

One of the most critical aspects that influences the entire TN-ML pipeline is the data embedding procedure, which is the starting point of every training. The choice of appropriate data embedding directly affects the expressivity of the TN and steers the optimization process toward meaningful solutions. However, this is a problem- and data-specific question. A general theoretical framework for developing embeddings would help enhance the performance and robustness of TN-based models. In practice, a thorough analysis of the dataset is often required to uncover underlying correlations and choose how to best represent the data within a TN structure.

Software Generalization

The `tn4ml` library provides a modular and extensible framework for TNs in ML, but it is currently limited to 1D structures (e.g., MPS, MPO, SMPO). The extension would include the application to more complex data domains. Additionally, the library does not yet support integration of TN layers into neural network architectures, which could offer a broader range of modern ML problems. This has been implemented offline, and it is the first feature to be included in the future. `tn4ml` is still a work in progress, which would require person-power investment to reach the stage of a fully functional library.

Generalization to Real Physics Data

The datasets used in this work are simulated LHC events. They are used to provide meaningful results, but generalization to real LHC data could potentially include challenges such as detector effects, noise, class imbalance and larger systematic uncertainties. If these TN-based models are going to be realistically deployed, these challenges need to be addressed.

11.4 Future Directions

Here, we outline the future work that can be addressed for each application explored in this thesis. Some of these directions are already being actively pursued, while others highlight promising areas for future research.

Hybrid Quantum-Classical ML

The hybrid k-medians clustering algorithm developed in this work demonstrated how quantum subroutines can be integrated into classical pipelines, achieving comparable or better performance than pure classical methods. Future research in this area could explore:

- Joint training of the encoder for dimensionality reduction and quantum algorithm, enabling domain-specific optimization of the input space.
- Investigation of variational quantum circuits (VQCs) as quantum encoders for dimensionality reduction.
- Leveraging the full quantum nature of the dataset by developing quantum models that operate directly in high-dimensional settings.
- Practical deployment of such hybrid pipelines within real-time selection systems at the LHC.

Enhancing Interpretability for Optimization of Matrix Product States

An interesting aspect for future work could involve the exploration of the interpretability of MPS models by systematically analyzing feature-wise probabilities, mutual information, and entropy-related quantities either while training or post-training. One could develop tools for mapping the contribution of individual input features or groups of features to the final decision. This would lead to an information-based relevance analysis, where features with the highest mutual information with respect to the output are marked as key drivers of the model's prediction. On top of this, it would be useful to track entanglement entropy across tensors that may reveal how complexity and correlations propagate through the model, which would contribute to reducing possible redundancy in the representation. This could lead to TN pruning techniques, such as optimizing bond dimension while training. All these would contribute to improving the transparency and interpretability of tensor network models in real-life ML applications.

Hardware Accelerated Tensor Networks

This thesis demonstrated the feasibility of low-complexity deployment for TN models, which is important for integration into a real-time trigger system at the LHC. Open research directions include:

- Exploration of quantum-information tools, such as the entropy profile, to analyze where the model concentrates its attention to reduce parameters of the model and improve efficiency.
- Development of pruning and quantization functionality in the `tn4ml` library, with the goal of supporting both quantization-aware training and post-training quantization strategies.
- Optimization of TN inference pipelines for implementation on FPGAs to ensure the compatibility with the latency requirement of $\mathcal{O}(100)$ns.
- Benchmarking of TN inference in terms of time and resource consumption compared to compressed classical ML models.

Expanding `tn4ml` and Open Science

Although the `tn4ml` library currently offers a wide range of functionalities for 1D TN structures, its design is open for future development:

- Expanding support for multidimensional TNs, such as 2D or tree-like structures.
- Adding support for hybrid models combining neural network and TN layers.
- Creating educational modules, tutorials and benchmarks to promote cross-disciplinary applications.
- Supporting multiple backends, such as `PyTorch`.

Continued development of `tn4ml` will encourage versioning, community contributions and reproducibility, thus contributing to advancing quantum-inspired architectures in ML.

Chapter 12

Conclusions

This thesis investigates hybrid quantum-classical methods and quantum-inspired Tensor Networks for applied machine learning problems in HEP. This research program was motivated by the potential of quantum algorithms to enhance physics analysis, and expands this perspective by critically evaluating the utility of quantum-inspired classical methods as scalable and efficient alternatives. The primary ML tasks addressed in this work are unsupervised anomaly detection, essential for discovering previously unknown physics phenomena, and classification of jet substructure, which is crucial for improving the trigger resolution and filtering of the background events in real-time LHC data processing. Both tasks are explored in the context of jet-related processes, aligning with active, relevant LHC research scenarios and providing a realistic benchmark for evaluating new algorithms. A central technical contribution of this work is the development of `tn4ml`, a Python library that facilitates the training, customization, and benchmarking of TN models in practical ML workflows. This framework enables the implementation of a series of applications, including: TN-based anomaly detection in the latent space of LHC collision events, classification of jet substructure with TNs, and broader use cases such as anomaly detection on MNIST and breast cancer detection. Across all applications, this thesis emphasizes thorough performance analysis - evaluating metrics such as execution time, robustness of classification or anomaly detection, model complexity, or data-related parameters - to guide the design of the final algorithm as done in traditional ML pipelines. These results demonstrate the feasibility and competitive potential of quantum and TN-based models in both domain-specific and general-purpose ML settings.

12.1 Summary of Contributions

The following are the main contributions:

Package Development: `tn4ml` [2] is a user-friendly open-source Python library developed for designing and evaluating Tensor Network architectures for ML tasks. It supports both supervised and unsupervised settings, and provides modules for data embedding, model initialization, training strategies and customization, objective functions and evaluation procedures. The package is publicly available at github.com/bsc-quantic/tn4ml.

Hybrid Clustering for Anomaly Detection in HEP: A realistic hybrid quantum-classical anomaly detection pipeline was developed for LHC simulation data, combining an encoder for dimensionality reduction with k-medians clustering in the latent space [1]. The clustering algorithm integrates a quantum subroutine for distance estimation with a classical algorithm for geometric median computation. The results of this study demonstrate that quantum components can be successfully integrated into a classical ML pipeline without degrading overall performance.

Anomaly Detection with Matrix Product States in HEP: We developed a continuous probabilistic model based on a parametrized MPS, trained to learn the QCD background samples, and to potentially identify unseen anomalous signatures [3]. This approach achieved competitive - and in certain cases even improved - performance when benchmarked against quantum and hybrid baselines. The results highlight the importance of choosing an appropriate initialization technique and data embedding procedure for the parametrized TN model. Furthermore, this model is suitable for deployment in the real-time selection system of the High-Level Trigger at the LHC.

Jet Tagging with Tensor Network: A TN model was developed for jet tagging using particle-inspired embeddings. The design was motivated by future deployment on FPGAs, laying the groundwork for realistic, practical applications of TN structures in low-latency environments (fully described in Chapter 10).

Robust Design and Analysis of Parameterized TN models for ML applications: The development of TN models for various ML applications in HEP was conducted following ML practices. This includes k-fold cross-validation, systematic hyperparameter optimization, evaluation using multiple performance metrics, analysis of the impact of model complexity, and assessment of execution metrics. This is contribution is proven through Refs. [2, 3] and in this thesis's chapters 6, 9 and 10.

Appendix A

Author Contributions and Dissemination

The following publications, conference contributions, scientific dissemination activities, teaching, and supervision were completed during this PhD. Some of them reflect the development and application of the thesis research and some present smaller projects done throughout this thesis, which are not fully included in the overall text of the thesis.

1. Peer-Reviewed Journal Articles

- E. Puljak, V. Belis, K. A. Woźniak, *et al.* **Quantum anomaly detection in the latent space of proton collision events at the LHC**, *Nature Communications Physics*, 2024 [1].

- E. Puljak, *et al.*, **Tensor Network for Anomaly Detection in the Latent Space of Proton Collision Events at the LHC**, *Machine Learning: Science and Technology*, 2025 [3].

2. Pre-print Submissions to Peer-Reviewed Journals

- E. Puljak, *et al.*, `tn4ml`**: Tensor Network Training and Customization for Machine Learning**, submitted to *Quantum Journal*, 2025 [2].

3. Manuscripts in Preparation

- E. Puljak, *et al.*, **Tensor Networks for Jet Classification on FPGA at the L1-Trigger for HL-LHC**. This study is a collaborative effort between CERN, Universitat Autonoma de Barcelona and the University of Padova. Chapter 10 presents an important part of this study.

4. Contributions at International Conferences

- **Towards Ultrafast Quantum-Inspired Tensor Networks Models for Jet Tagging on FPGAs**. Talk presented at *Fast Machine Learning for Science Conference*, Zurich, Switzerland, 2025.

- **Quantum-Inspired Tensor Networks for Anomaly Detection in the latent space of proton collision events at the LHC**. Poster presented at *International Conference on Quantum Technologies for High-Energy Physics*, Geneva, Switzerland, 2025.

- **Quantum-Inspired Tensor Networks for Unsupervised and Supervised Cancer Detection in Medical Imaging**. Poster presented at *Quantum Techniques in Machine Learning*, Melbourne, Australia, 2024. qtml2024/105

- **How Tensor Networks connect Quantum and Classical Machine Learning**. Talk presented at conference *AI2FUTURE*, Zagreb, Croatia, 2024.

- **What the fuss is Quantum Machine Learning?**. Talk presented at two occasions: (1) gathering of ML enthusiasts (*ML Pub*) in the organization of *croAI*. Zagreb, Croatia, 2024; (2) *Data Science Conference*, Belgrade, Serbia, 2024.

- **Quantum-Inspired Tensor Networks for Anomaly Detection at the LHC**. Poster presented at *Quantum Techniques in Machine Learning*, Geneva, Switzerland, 2023.

- **The Role of Quantum Computing in shaping the future of Machine Learning**. Talk presented at conference *AI2FUTURE*, Zagreb, Croatia, 2023.

- **Quantum Computing: technology that will change the world**. Talk presented at computer science conference for young programmers *DUMP days*, Split, Croatia, 2023.

- **Quantum anomaly detection in the latent space of proton collision events at the LHC**. Poster presented at *International Quantum Tensor Network second plenary meeting*, Center for Computational Quantum Physics, New York City, USA, 2023. `Zenodo` [163].

5. Lectures

- **Introduction to Quantum Machine Learning and Tensor Networks**. Lecture at the University of Zurich, Switzerland, 2024.

- CERN Summer School lecture series (Geneva, Switzerland, 2024):

  1. **Basics of Quantum Computing**;
  2. **Introduction to Tensor Networks** (including practical tutorial with `tn4ml` library).

Supervision and Mentoring

During this PhD, I had the opportunity to write a proposal for a Master's Thesis project, supervise and mentor a Master's student, Gabriele Di'Angeli, at EPFL. This research project was related to exploring data embedding procedures for Tensor Networks applied to lung cancer diagnosis, and was designed to contribute to the development of `tn4ml` library. The project was conducted from September 2024 to April 2025. I was responsible for writing the project scope, providing initial literature, conducting regular meetings, motivating the student with new ideas, and providing constant feedback on code, analysis and thesis writing. I thank Maurizio Pierini and Michele Grossi for making this possible. This experience has enriched my skills with research leadership, communication and has contributed to broader research done during my thesis.

Appendix B

Code and Data Availability

Code Availability

The source code for developing studies in this thesis is publicly available in the following repositories:

- **Tensor Networks for Machine Learning** - `tn4ml` (Chapter 6)
  - `GitHub`: github.com/bsc-quantic/tn4ml/
- **Quantum-Classical Clustering for Anomaly Detection** (Chapter 8)
  - `GitHub`: github.com/vbelis/latent-ad-qml/qad/algorithms/kmedians;
  - `Zenodo` - Ref. [164];
  - Tutorial in `Qibo`: .github.com/qiboteam/qibo/examples/qclustering
- **Matrix Product State for Anomaly Detection** (Chapter 9)
  - `GitHub`: github.com/bsc-quantic/tn4ml/docs/source/examples/tnad_latent

All code is released under the MIT License. Instructions for reproducing the experiments and figures from Chapters 6, 8 and 9 are included in the corresponding README files.

Data Availability

The datasets used in this thesis that are publicly available are:

- **Jet Dataset for Anomaly Detection** presented in Section 7.3 and processed with the encoder, as described is Chapter 8, is publicly accessible from `Zenodo` [165]. For more details on the encoder pre-processing, refer to Chapter 8. This dataset is also explored in Chapter 9.
- **Jet Dataset for Tagging**, described in Section 7.2, is publicly available on `Zenodo` [116] and used for study in Chapter 10.
- **Breast Cancer** dataset is available on the Kaggle website [111, 112], and it is used for an example of supervised learning to showcase the `tn4ml` library.

Appendix C

CMS Detector

C.1 General description

The particle accelerator Large Hadron Collider (LHC) hosts seven bigger experiments along the length of its ring, each equipped with detectors or systems recording and analyzing particles generated in high-energy proton-proton collisions. One of them is the Compact Muon Solenoid (CMS) experiment, a general-purpose detector designed to explore a wide range of physics phenomena, such as the search for new particles and precise measurements of the Standard Model processes [166]. The characteristic feature of the CMS detector is its powerful superconducting solenoid magnet, which generates a magnetic field of 3.8T. This strong magnetic field is important for bending trajectories of charged particles, allowing for accurate reconstruction of their transverse momentum components. The architecture of the detector is built around the collision point in the beam, and it contains several subdetectors, as illustrated in the cross-sectional view (see Fig. C.1). The first innermost layer is designed as an all-silicon tracking system, followed by electromagnetic and hadronic calorimeters, with the last layer consisting of muon detection chambers. Each of the subparts is described in the following sections.

The data acquisition strategy at the CMS is designed as a two-level trigger system operating in real-time and selecting potentially interesting events. The Level-1 Trigger (L1T) provides fast initial filtering of events in the Field Programmable Gate Arrays (FPGAs), while the High-Level Trigger (HLT) further reduces data rates in CPU farms

using software algorithms. This system is crucial for handling enormous data rates and storing a manageable amount of data for offline analysis.

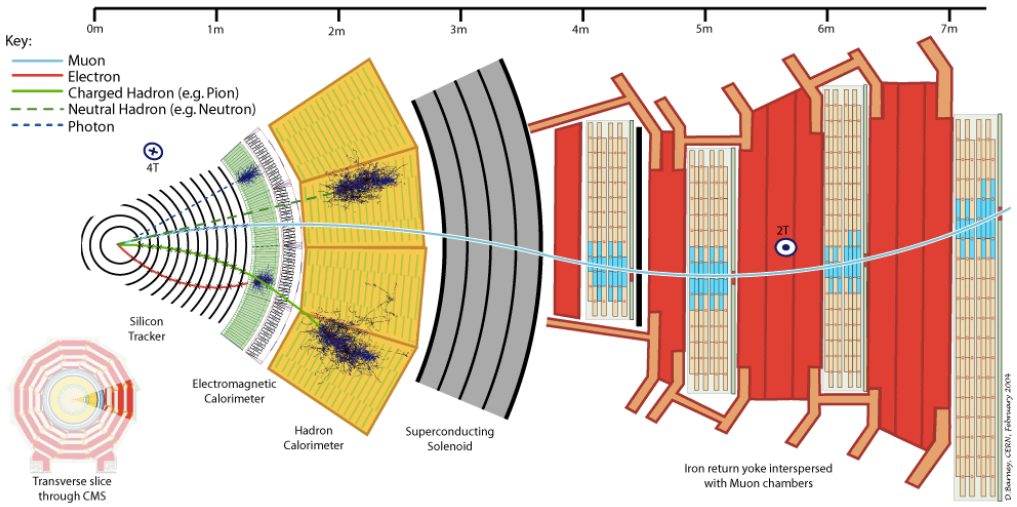


FIGURE C.1: A cross-section slice of the CMS detector barrel, with all sub-detectors and trajectories of known particles.

C.2 LHC coordinate system

Since this work uses events simulated with the CMS detector characteristics, here we will describe the parameters used for the analysis. Fig. C.2 visually describes the setup, following the standard Cartesian coordinate system. The z-axis follows the beam direction, and x-y plane defines the transverse plane. Polar and azimuthal angles are denoted as θ and ϕ, respectively. Pseudorapidity is expressed as $\eta = -\log(\tan(\theta/2))$, and p_{T} is the momentum p component perpendicular to the beam (z) axis.

C.3 Silicon Tracker

The Silicon Tracker is the innermost, all-silicon layer of the CMS detector [167]. It has a sensitive area of approximately $200\mathrm{m}^2$, and it is designed to operate in extreme radiation conditions near the collision point in the LHC beam. It has a lightweight design, which does not disturb particles passing through. The primary purpose of the tracker is to precisely reconstruct the trajectories of charged particles from the proton-proton collisions. It is essential for high-resolution and three-dimensional spatial measurements, contributing to vertex reconstruction and momentum estimation [168]. By measuring the curvature of

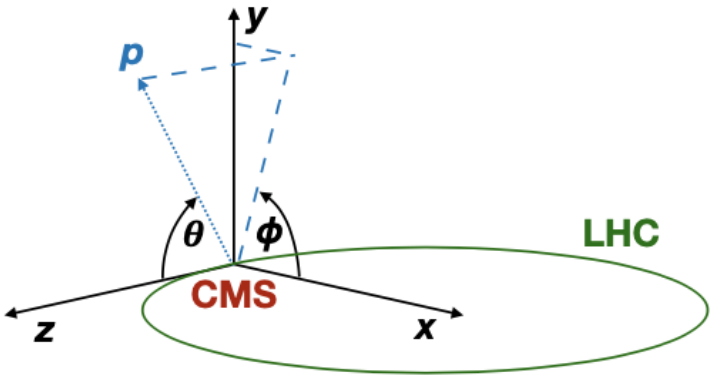


FIGURE C.2: The coordinate system of CMS, showing the parameters: azimuthal angle ϕ, polar angle θ, and particle's momentum, p. The collision point is around $x = y = 0$, and the z-axis is parallel to the beam axis.

the charged particles through the CMS magnetic field, the transverse momentum p_{T} can be determined with high accuracy. The track curvature is inversely proportional to the momentum, making the tracker important for separating high- and low-momentum particles. Main components of the tracker are: the pixel detector, positioned at the interaction point provides fine granularity, and the silicon strip detectors around it, which extend the tracking area. A detailed description of the tracker architecture and performance can be found in Refs. [169, 170]

C.4 Calorimeter System

The CMS calorimeter system measures the energy of particles emerging from collisions. When a particle (charged or neutral) interacts with the calorimeter, it initiates an electromagnetic or hadronic shower, which leaves behind the energy in a form (e.g., light or charge) that can be converted into a measurable signal. The calorimeter system consists of two main parts: the Electromagnetic Calorimeter (ECAL) [171] and the Hadron Calorimeter (HCAL) [172].

- ECAL: The first layer after the tracker designed to measure the energy of electrons and photons with excellent resolution. It is built of lead tungstate crystals ($PbWO_4$) arranged in a homogeneous structure consisting of a barrel and two endcap regions.

- HCAL: The layer after ECAL captures strongly interacting particles (quarks, gluons) through hadronic showers created when particles enter its structure. It is designed as a composition of alternating layers of dense absorbers (e.g., brass or steel) and plastic scintillator tiles.

Together, these two parts form a system designed to detect almost all of the energy coming from a collision. Any imbalance in measured transverse energy indicates the possibility of the presence of weakly interacting particles, such as neutrinos or new neutral states. This *missing transverse energy* (MET or E^T_{miss}) is opening possibilities for searches of invisible particles and new physics.

C.5 Muon system

The CMS Muon system, placed outside the calorimeter system in the magnet return yoke, is designed to specifically detect muons. It consists of three gas-based subdetectors:

- Drift Tubes (DT) in the barrel region,
- Cathode Strip Chambers (CSC) in the endcap,
- Resistive Plate Chambers (RPC) in both barrel and endcaps.

Data from each part is combined and matched with silicon tracker tracks to reconstruct muon trajectories with high precision. A more detailed description can be found in Ref. [173].

C.6 Trigger System at the LHC

Collisions at the LHC occur at a frequency of 40 MHz, producing data at the rate of several thousand terabits per second, which is impossible to store or analyze in real-time. To address this challenge, the CMS experiment has integrated an online trigger system that identifies and selects the most relevant events. The trigger system consists of two stages:

1. L1T - reduces the data rate from 40 MHz to $\mathcal{O}(100)$kHz, algorithms running on Field-Programmable Gate Arrays (FPGAs) with latency constraint $\mathcal{O}(1)\mu s$;

2. HLT - reduces the data rate from $\mathcal{O}(100)$kHz to $\mathcal{O}(1)$kHz, running on CPU farm

The trigger system selects the data for offline analysis, making the development of high-quality selection algorithms and improvements in the trigger precision highly important. Ongoing development is also driven by the scheduled upgrade of the LHC, the *High-Luminosity* (HL-LHC) program [162], which will increase the number of simultaneous proton-proton interactions per bunch crossing - known as *pileup* - from ~ 60 to ~ 140 [149], resulting in increased data complexity. This presents significant challenges for real-time event selection and increases the need to develop methods to preserve physics sensitivity and efficiency. In the CMS experiment, the biggest improvement will be driven by incorporating the particle-level reconstruction algorithm, the so-called particle-flow (PF) [120], which performs single-particle reconstruction and identification by combining information from various detector components. This will enable the integration of particle tracks from the silicon tracker, through calorimeters, and into the muon system in order to improve the precision of event selection.

—-

List of Figures

List of Tables

Bibliography

[1] Vasilis Belis, Kinga Anna Woźniak, Ema Puljak, et al. Quantum anomaly detection in the latent space of proton collision events at the LHC. *Communications Physics*, 7(1):334, 2024. ISSN 2399-3650. doi: 10.1038/s42005-024-01811-6. URL https://doi.org/10.1038/s42005-024-01811-6.

[2] Ema Puljak et al. tn4ml: Tensor network training and customization for machine learning, 2025. URL https://arxiv.org/abs/2502.13090.

[3] Ema Puljak, Maurizio Pierini, and Artur Garcia-Saez. Tensor network for anomaly detection in the latent space of proton collision events at the LHC. *Machine Learning: Science and Technology*, 6(4):045001, oct 2025. doi: 10.1088/2632-2153/ae0243. URL https://doi.org/10.1088/2632-2153/ae0243.

[4] Sukhpal Singh Gill et al. Quantum computing: Vision and challenges, 2025. URL https://arxiv.org/abs/2403.02240.

[5] Frank Verstraete, Valentin Murg, and Juan Ignacio Cirac. Matrix product states, projected entangled pair states, and variational renormalization group methods for quantum spin systems. *Advances in Physics*, 57(2):143–224, 2008. URL https://doi.org/10.1080/14789940801912366.

[6] Román Orús. A practical introduction to tensor networks: Matrix product states and projected entangled pair states. *Annals of Physics*, 349:117–158, 2014. ISSN 0003-4916. URL https://www.sciencedirect.com/science/article/pii/S0003491614001596.

[7] Shi-Ju Ran et al. *Tensor Network Contractions: Methods and Applications to Quantum Many-Body Systems*, volume 964 of *Lecture Notes in Physics*. Springer,

2020. ISBN 978-3-030-34488-4. doi: 10.1007/978-3-030-34489-4. URL https://link.springer.com/book/10.1007/978-3-030-34489-4.

[8] Lorenzo Borella et al. Ultra-low latency quantum-inspired machine learning predictors implemented on FPGA, 2024. URL https://arxiv.org/abs/2409.16075.

[9] Edwin Miles Stoudenmire and David J. Schwab. Supervised learning with tensor networks. In *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/5314b9674c86e3f9d1ba25ef9bb32895-Paper.pdf.

[10] Xingwei Cao and Guillaume Rabusseau. Tensor regression networks with various low-rank tensor approximations, 2018. URL https://arxiv.org/abs/1712.09520.

[11] Justin Reyes and Edwin Miles Stoudenmire. Multi-scale tensor network architecture for machine learning. *Machine Learning: Science and Technology*, 2(3):035036, jul 2021. URL https://dx.doi.org/10.1088/2632-2153/abffe8.

[12] Alejandro Pozas-Kerstjens et al. Privacy-preserving machine learning with tensor networks. *Quantum*, 8:1425, 2024. doi: 10.22331/q-2024-07-25-1425. URL https://doi.org/10.22331/q-2024-07-25-1425.

[13] David Perez-Garcia et al. Matrix product state representations. *Quantum Info. Comput.*, 7(5):401–430, July 2007. ISSN 1533-7146.

[14] Juan Ignacio Cirac et al. Matrix product states and projected entangled pair states: Concepts, symmetries, theorems. *Reviews of Modern Physics*, 93(4), December 2021. ISSN 1539-0756. URL http://dx.doi.org/10.1103/RevModPhys.93.045003.

[15] Frank Verstraete et al. Matrix product density operators: Simulation of finite-temperature and dissipative systems. *Phys. Rev. Lett.*, 93:207204, Nov 2004. doi: 10.1103/PhysRevLett.93.207204. URL https://link.aps.org/doi/10.1103/PhysRevLett.93.207204.

[16] Jinhui Wang et al. Anomaly detection with tensor networks, 2020. URL https://arxiv.org/abs/2006.02516.

[17] Frank Verstraete and Juan Ignacio Cirac. Renormalization algorithms for quantum-many body systems in two and higher dimensions, 2004. URL https://arxiv.org/abs/cond-mat/0407066.

[18] Guifre Vidal. Class of quantum many-body states that can be efficiently simulated. *Physical Review Letters*, 101(11), sep 2008. ISSN 1079-7114. doi: 10.1103/physrevlett.101.110501. URL http://dx.doi.org/10.1103/PhysRevLett.101.110501.

[19] Ville Bergholm et al. PennyLane: Automatic differentiation of hybrid quantum-classical computations, 2022. URL https://arxiv.org/abs/1811.04968.

[20] Ali Javadi-Abhari et al. Quantum computing with Qiskit, 2024. URL https://arxiv.org/abs/2405.08810.

[21] Cirq Developers. Cirq, April 2025. URL https://doi.org/10.5281/zenodo.15191735.

[22] Stavros Efthymiou et al. Qibo: a framework for quantum simulation with hardware acceleration. *Quantum Science and Technology*, 7(1):015018, dec 2021. doi: 10.1088/2058-9565/ac39f5. URL https://doi.org/10.1088/2058-9565/ac39f5.

[23] Renato M. S. Farias et al. qiboteam/qibo: qibo 0.2.19, June 2025. URL https://doi.org/10.5281/zenodo.15705051.

[24] Jacob Miller. Torchmps. https://github.com/jemisjoky/torchmps, 2019.

[25] José Ramón Pareja Monturiol et al. TensorKrowch: Smooth integration of tensor networks in machine learning. *Quantum*, 8:1364, June 2024. ISSN 2521-327X. URL https://doi.org/10.22331/q-2024-06-11-1364.

[26] Johannes Hauschild et al. Tensor network Python (TeNPy) version 1. *SciPost Phys. Codebases*, page 41, 2024. URL https://scipost.org/10.21468/SciPostPhysCodeb.41.

[27] Johnnie Gray. quimb: a python library for quantum information and many-body calculations. *Journal of Open Source Software*, 3(29):819, 2018.

[28] Chase Roberts et al. TensorNetwork: A library for physics and machine learning, 2019. URL https://arxiv.org/abs/1905.01330.

[29] Matthew Fishman et al. The ITensor Software Library for Tensor Network Calculations. *SciPost Phys. Codebases*, page 4, 2022. URL https://scipost.org/10.21468/SciPostPhysCodeb.4.

[30] Matthew Fishman et al. ITensors.jl, 2024. URL https://github.com/ITensor/ITensors.jl/releases/tag/v0.8.0.

[31] Jutho Haegeman et al. TensorKit.jl, 2024. URL https://github.com/Jutho/TensorKit.jl.

[32] Sergio Sanchez-Ramirez et al. Tenet.jl: The Hackable Tensor Network library, 2025. URL https://github.com/bsc-quantic/Tenet.jl.

[33] Maria Schuld and Francesco Petruccione. *Machine Learning with Quantum Computers*. Quantum Science and Technology. Springer Cham, 2 edition, 2021. ISBN 978-3-030-83097-7.

[34] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In Yves Lechevallier and Gilbert Saporta, editors, *Proceedings of COMPSTAT'2010*, pages 177–186, Heidelberg, 2010. Physica-Verlag HD. ISBN 978-3-7908-2604-3.

[35] Diederik P. Kingma and Jimmy Ba. Adam: A method for stochastic optimization, 2017. URL https://arxiv.org/abs/1412.6980.

[36] Sebastian Ruder. An overview of gradient descent optimization algorithms, 2017. URL https://arxiv.org/abs/1609.04747.

[37] CMS Collaboration. Observation of a new boson at a mass of 125 GeV with the CMS experiment at the LHC. *Physics Letters B*, 716(1):30–61, sep 2012. ISSN 0370-2693. doi: 10.1016/j.physletb.2012.08.021. URL http://dx.doi.org/10.1016/j.physletb.2012.08.021.

[38] ATLAS Collaboration. Observation of a new particle in the search for the Standard Model Higgs boson with the ATLAS detector at the LHC. *Physics Letters B*, 716 (1):1–29, sep 2012. ISSN 0370-2693. doi: 10.1016/j.physletb.2012.08.020. URL http://dx.doi.org/10.1016/j.physletb.2012.08.020.

[39] Vinicius Mikuni and Florencia Canelli. Unsupervised clustering for collider physics. *Phys. Rev. D*, 103:092007, May 2021. doi: 10.1103/PhysRevD.103.092007. URL https://link.aps.org/doi/10.1103/PhysRevD.103.092007.

[40] ATLAS Collaboration. Dijet Resonance Search with Weak Supervision Using $\sqrt{s}$ = 13 TeV pp Collisions in the ATLAS Detector. *Physical Review Letters*, 125

(13), sep 2020. ISSN 1079-7114. doi: 10.1103/physrevlett.125.131801. URL http://dx.doi.org/10.1103/PhysRevLett.125.131801.

[41] Gregor Kasieczka et al. The LHC Olympics 2020 a community challenge for anomaly detection in high energy physics. *Reports on Progress in Physics*, 84(12):124201, dec 2021. doi: 10.1088/1361-6633/ac36b9. URL https://dx.doi.org/10.1088/1361-6633/ac36b9.

[42] Tobias Golling et al. The massive issue: Anomaly detection in jet physics, 2023. URL https://arxiv.org/abs/2303.14134.

[43] Louis Vaslin, Vincent Barra, and Julien Donini. GAN-AE: an anomaly detection algorithm for New Physics search in LHC data. *The European Physical Journal C*, 83(11), nov 2023. ISSN 1434-6052. doi: 10.1140/epjc/s10052-023-12169-4. URL http://dx.doi.org/10.1140/epjc/s10052-023-12169-4.

[44] Vasilis Belis, Patrick Odagiu, and Thea Klaeboe Aarrestad. Machine learning for anomaly detection in particle physics. *Reviews in Physics*, 12:100091, dec 2024. ISSN 2405-4283. doi: 10.1016/j.revip.2024.100091. URL http://dx.doi.org/10.1016/j.revip.2024.100091.

[45] CMS Collaboration. Model-agnostic search for dijet resonances with anomalous jet substructure in proton-proton collisions at $\sqrt{s} = 13$ TeV. *Reports on Progress in Physics*, 88(6):067802, jun 2025. doi: 10.1088/1361-6633/add762. URL https://dx.doi.org/10.1088/1361-6633/add762.

[46] Rolf Landauer. The physical nature of information. *Physics Letters A*, 217(4):188–193, 1996. ISSN 0375-9601. doi: https://doi.org/10.1016/0375-9601(96)00453-7. URL https://www.sciencedirect.com/science/article/pii/0375960196004537.

[47] Tony Hey. Quantum computing: An introduction. *Computing & Control Engineering Journal*, 10:105 – 112, 09 1999. doi: 10.1049/ccej:19990303.

[48] Charles Henry Bennett. Logical reversibility of computation. *IBM Journal of Research and Development*, 17(6):525–532, 1973. doi: 10.1147/rd.176.0525.

[49] David Deutsch and Roger Penrose. Quantum theory, the church–turing principle and the universal quantum computer. *Proceedings of the Royal Society of London.*

A. Mathematical and Physical Sciences, 400(1818):97–117, 1985. doi: 10.1098/rspa.1985.0070. URL https://royalsocietypublishing.org/doi/abs/10.1098/rspa.1985.0070.

[50] Peter Williston Shor. Polynomial-time algorithms for prime factorization and discrete logarithms on a quantum computer. *SIAM Journal on Computing*, 26(5): 1484–1509, 1997. doi: 10.1137/S0097539795293172. URL https://doi.org/10.1137/S0097539795293172.

[51] Lov Kumar Grover. A fast quantum mechanical algorithm for database search. In *Proceedings of the Twenty-Eighth Annual ACM Symposium on Theory of Computing*, STOC '96, page 212–219, New York, NY, USA, 1996. Association for Computing Machinery. ISBN 0897917855. doi: 10.1145/237814.237866. URL https://doi.org/10.1145/237814.237866.

[52] Leonard Susskind and Art Friedman. *Quantum Mechanics: The Theoretical Minimum*. Basic Books, New York, NY, 2014. ISBN 978-0-4650-3629-8, 978-0-4650-6290-4.

[53] Michael A. Nielsen and Isaac L. Chuang. *Quantum Computation and Quantum Information: 10th Anniversary Edition*. Cambridge University Press, 2010.

[54] Hsin-Yuan Huang et al. Power of data in quantum machine learning. *Nature Communications*, 12(1):2631, 2021. doi: 10.1038/s41467-021-22539-9. URL https://doi.org/10.1038/s41467-021-22539-9.

[55] Minati Rath and Hema Date. Quantum data encoding: A comparative analysis of classical-to-quantum mapping techniques and their impact on machine learning accuracy, 2023. URL https://arxiv.org/abs/2311.10375.

[56] Iulia Georgescu, Sahel Ashhab, and Franco Nori. Quantum simulation. *Reviews of Modern Physics*, 86(1):153–185, March 2014. ISSN 1539-0756. doi: 10.1103/revmodphys.86.153. URL http://dx.doi.org/10.1103/RevModPhys.86.153.

[57] Clémence Chevignard, Pierre-Alain Fouque, and André Schrottenloher. Reducing the number of qubits in quantum factoring. Cryptology ePrint Archive, Paper 2024/222, 2024. URL https://eprint.iacr.org/2024/222.

[58] Maria Schuld and Nathan Killoran. Is quantum advantage the right goal for quantum machine learning? *PRX Quantum*, 3:030101, Jul 2022. doi: 10.1103/PRXQuantum.3.030101. URL https://link.aps.org/doi/10.1103/PRXQuantum.3.030101.

[59] Martín Larocca et al. Barren plateaus in variational quantum computing. *Nature Reviews Physics*, 7(4):174–189, 2025. doi: 10.1038/s42254-025-00813-9. URL https://doi.org/10.1038/s42254-025-00813-9.

[60] Joseph Bowles, Shahnawaz Ahmed, and Maria Schuld. Better than classical? the subtle art of benchmarking quantum machine learning models, 2024. URL https://arxiv.org/abs/2403.07059.

[61] Ian Jolliffe. *Principal Component Analysis*. Springer Series in Statistics. Springer, New York, NY, 1986. ISBN 978-1-4757-1904-8. doi: 10.1007/978-1-4757-1904-8.

[62] Simone Montangero. *Introduction to Tensor Network Methods*. Springer, 2018. ISBN 978-3-03-001408-7. doi: 10.1007/978-3-030-01409-4. URL https://link.springer.com/book/10.1007/978-3-030-01409-4.

[63] Román Orús. Tensor networks for complex quantum systems. *Nature Reviews Physics*, 1(9):538–550, 2019. doi: 10.1038/s42254-019-0086-7. URL https://doi.org/10.1038/s42254-019-0086-7.

[64] Matthew B. Hastings. An area law for one-dimensional quantum systems. *Journal of Statistical Mechanics: Theory and Experiment*, 2007(08):P08024–P08024, August 2007. ISSN 1742-5468. doi: 10.1088/1742-5468/2007/08/p08024. URL http://dx.doi.org/10.1088/1742-5468/2007/08/P08024.

[65] Itai Arad et al. An area law and sub-exponential algorithm for 1d systems, 2013. URL https://arxiv.org/abs/1301.1162.

[66] Hai-Jun Liao et al. Differentiable programming tensor networks. *Physical Review X*, 9(3), sep 2019. ISSN 2160-3308. doi: 10.1103/physrevx.9.031041. URL http://dx.doi.org/10.1103/PhysRevX.9.031041.

[67] Gemma De las Cuevas et al. Irreducible forms of matrix product states: Theory and applications. *Journal of Mathematical Physics*, 58(12), dec 2017. ISSN 1089-7658. doi: 10.1063/1.5000784. URL http://dx.doi.org/10.1063/1.5000784.

[68] Frank Verstraete et al. Density matrix renormalization group, 30 years on. *Nature Reviews Physics*, 5(5):273–276, 2023. doi: 10.1038/s42254-023-00572-5. URL https://doi.org/10.1038/s42254-023-00572-5.

[69] Ulrich Schollwöck. The density-matrix renormalization group in the age of matrix product states. *Annals of Physics*, 326(1):96–192, 2011. ISSN 0003-4916. doi: https://doi.org/10.1016/j.aop.2010.09.012. URL https://www.sciencedirect.com/science/article/pii/S0003491610001752. January 2011 Special Issue.

[70] James Dborin et al. Matrix product state pre-training for quantum machine learning, 2021. URL https://arxiv.org/abs/2106.05742.

[71] Manuel S. Rudolph et al. Decomposition of matrix product states into shallow quantum circuits. *Quantum Science and Technology*, 9(1):015012, nov 2023. doi: 10.1088/2058-9565/ad04e6. URL https://dx.doi.org/10.1088/2058-9565/ad04e6.

[72] Jason Iaconis and Sonika Johri. Tensor network based efficient quantum data loading of images, 2023. URL https://arxiv.org/abs/2310.05897.

[73] Manuel S. Rudolph et al. Synergistic pretraining of parametrized quantum circuits via tensor networks. *Nature Communications*, 14(1):8367, 2023. doi: 10.1038/s41467-023-43908-6. URL https://doi.org/10.1038/s41467-023-43908-6.

[74] Thomas Ayral et al. Density-matrix renormalization group algorithm for simulating quantum circuits with a finite fidelity. *PRX Quantum*, 4(2), apr 2023. ISSN 2691-3399. doi: 10.1103/prxquantum.4.020304. URL http://dx.doi.org/10.1103/PRXQuantum.4.020304.

[75] Manuel S. Rudolph et al. Classical surrogate simulation of quantum systems with LOWESA, 2023. URL https://arxiv.org/abs/2308.09109.

[76] Joseph Tindall et al. Efficient tensor network simulation of ibm's eagle kicked ising experiment. *PRX Quantum*, 5:010308, Jan 2024. doi: 10.1103/PRXQuantum.5.010308. URL https://link.aps.org/doi/10.1103/PRXQuantum.5.010308.

[77] Siddhartha Patra et al. Efficient tensor network simulation of ibm's largest quantum processors. *Phys. Rev. Res.*, 6:013326, Mar 2024. doi: 10.1103/PhysRevResearch.6.013326. URL https://link.aps.org/doi/10.1103/PhysRevResearch.6.013326.

[78] Alexander Novikov et al. Tensorizing neural networks, 2015. URL https://arxiv.org/abs/1509.06569.

[79] Alexander Novikov, Mikhail Trofimov, and Ivan Oseledets. Exponential machines, 2017. URL https://arxiv.org/abs/1605.03795.

[80] Andrei Tomut et al. CompactifAI: Extreme Compression of Large Language Models using Quantum-Inspired Tensor Networks, 2024. URL https://arxiv.org/abs/2401.14109.

[81] Maolin Wang et al. Tensor networks meet neural networks: A survey and future perspectives, 2025. URL https://arxiv.org/abs/2302.09019.

[82] Saeed S. Jahromi and Román Orús. Variational tensor neural networks for deep learning. *Scientific Reports*, 14(1), August 2024. ISSN 2045-2322. doi: 10.1038/s41598-024-69366-8. URL http://dx.doi.org/10.1038/s41598-024-69366-8.

[83] Shi-Ju Ran and Gang Su. Tensor networks for interpretable and efficient quantum-inspired machine learning. *Intelligent Computing*, 2, jan 2023. ISSN 2771-5892. doi: 10.34133/icomputing.0061. URL http://dx.doi.org/10.34133/icomputing.0061.

[84] Zhao-Yu Han et al. Unsupervised generative modeling using matrix product states. *Physical Review X*, 8(3):031012, 2018.

[85] Borja Aizpurua et al. Tensor networks for explainable machine learning in cybersecurity, 2024.

[86] Alex Meiburg et al. Generative learning of continuous data by tensor networks. *SciPost Phys.*, 18:096, 2025. doi: 10.21468/SciPostPhys.18.3.096. URL https://scipost.org/10.21468/SciPostPhys.18.3.096.

[87] Jing Chen et al. Equivalence of restricted boltzmann machines and tensor network states. *Physical Review B*, 97(8), feb 2018. ISSN 2469-9969. doi: 10.1103/physrevb.97.085104. URL http://dx.doi.org/10.1103/PhysRevB.97.085104.

[88] Jing Liu et al. Tensor networks for unsupervised machine learning. *Physical Review E*, 107(1), jan 2023. ISSN 2470-0053. doi: 10.1103/physreve.107.l012103. URL http://dx.doi.org/10.1103/PhysRevE.107.L012103.

[89] Ivan Glasser et al. From probabilistic graphical models to generalized tensor networks for supervised learning, 2019.

[90] Glen Evenbly. Number-state preserving tensor networks as classifiers for supervised learning, 2022. ISSN 2296-424X. URL https://www.frontiersin.org/articles/10.3389/fphy.2022.858388.

[91] Diego Guala et al. Practical overview of image classification with tensor-network quantum circuits. *Nature: Scientific Reports*, 13, 2023. URL https://www.nature.com/articles/s41598-023-30258-y.

[92] Hao Chen and Thomas Barthel. Machine learning with tree tensor networks, cp rank constraints, and tensor dropout, 2023.

[93] Samuel T. Wauthier et al. Learning generative models for active inference using tensor networks, 2022. URL https://arxiv.org/abs/2208.08713.

[94] Friederike Metz and Marin Bukov. Self-correcting quantum many-body control using reinforcement learning with tensor networks. *Nature Machine Intelligence*, 5(7):780–791, jul 2023. ISSN 2522-5839. doi: 10.1038/s42256-023-00687-5. URL http://dx.doi.org/10.1038/s42256-023-00687-5.

[95] Bojan Žunkovič. Positive unlabeled learning with tensor networks. *Neurocomputing*, 552:126556, oct 2023. ISSN 0925-2312. doi: 10.1016/j.neucom.2023.126556. URL http://dx.doi.org/10.1016/j.neucom.2023.126556.

[96] Rohit Dilip et al. Data compression for quantum machine learning. *Phys. Rev. Res.*, 4:043007, Oct 2022. doi: 10.1103/PhysRevResearch.4.043007. URL https://link.aps.org/doi/10.1103/PhysRevResearch.4.043007.

[97] William Huggins et al. Towards quantum machine learning with tensor networks. *Quantum Science and Technology*, 4(2):024001, jan 2019. ISSN 2058-9565. doi: 10.1088/2058-9565/aaea94. URL http://dx.doi.org/10.1088/2058-9565/aaea94.

[98] Enrique Cervero Martín, Kirill Plekhanov, and Michael Lubasch. Barren plateaus in quantum tensor network optimization. *Quantum*, 7:974, apr 2023. ISSN 2521-327X. doi: 10.22331/q-2023-04-13-974. URL http://dx.doi.org/10.22331/q-2023-04-13-974.

[99] Aleksandr Berezutskii et al. Tensor networks for quantum computing, 2025. URL https://arxiv.org/abs/2503.08626.

[100] Ivan Glasser et al. Expressive power of tensor-network factorizations for probabilistic modeling, with applications from hidden markov models to quantum machine learning, 2019. URL https://arxiv.org/abs/1907.03741.

[101] Jacob C Bridgeman and Christopher T Chubb. Hand-waving and interpretive dance: an introductory course on tensor networks. *Journal of Physics A: Mathematical and Theoretical*, 50(22):223001, may 2017. ISSN 1751-8121. doi: 10.1088/1751-8121/aa6dc3. URL http://dx.doi.org/10.1088/1751-8121/aa6dc3.

[102] Jens Eisert et al. Colloquium: Area laws for the entanglement entropy. *Rev. Mod. Phys.*, 82:277–306, Feb 2010. doi: 10.1103/RevModPhys.82.277. URL https://link.aps.org/doi/10.1103/RevModPhys.82.277.

[103] James Bradbury et al. JAX: composable transformations of Python+NumPy programs, 2018. URL http://github.com/google/jax.

[104] Shi-Ju Ran. Encoding of matrix product states into quantum circuits of one- and two-qubit gates. *Phys. Rev. A*, 101:032310, Mar 2020. URL https://link.aps.org/doi/10.1103/PhysRevA.101.032310.

[105] Steven J. Leon et al. Gram-Schmidt orthogonalization: 100 years and more. *Numerical Linear Algebra with Applications*, 20(3):492–532, 2012.

[106] Francesco Mezzadri. How to generate random matrices from the classical compact groups, 2007. URL https://arxiv.org/abs/math-ph/0609050.

[107] Zheng-Zhi Sun et al. Tangent-space gradient optimization of tensor network for machine learning. *Phys. Rev. E*, 102:012152, Jul 2020. URL https://link.aps.org/doi/10.1103/PhysRevE.102.012152.

[108] Mikhail Usvyatsov et al. tntorch: Tensor network learning with pytorch, 2022. URL https://arxiv.org/abs/2206.11128.

[109] Lorenzo Loconte et al. What is the relationship between tensor factorizations and circuits (and how can we exploit it)?, 2025. URL https://arxiv.org/abs/2409.07953.

[110] Marco Ballarin et al. Quantum tea: qtealeaves, January 2024. URL https://doi.org/10.5281/zenodo.10498929.

[111] Rahma Sleam. Breast cancer dataset, 2024. URL https://www.kaggle.com/datasets/rahmasleam/breast-cancer/data. Accessed: 2025-02-04.

[112] UCI Machine Learning. Breast cancer wisconsin data, 2024. URL https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data/data. Accessed: 2025-02-10.

[113] Li Deng. The MNIST Database of Handwritten Digit Images for Machine Learning Research [Best of the Web]. *IEEE Signal Processing Magazine*, 29(6):141–142, 2012.

[114] M. Cacciari et al. The anti-k_t jet clustering algorithm. *JHEP*, 04:063, 2008. doi: 10.1088/1126-6708/2008/04/063.

[115] Eric A. Moreno et al. JEDI-net: a jet identification algorithm based on interaction networks. *The European Physical Journal C*, 80(1), jan 2020. ISSN 1434-6052. doi: 10.1140/epjc/s10052-020-7608-4. URL http://dx.doi.org/10.1140/epjc/s10052-020-7608-4.

[116] Maurizio Pierini et al. HLS4ML LHC jet dataset (150 particles), January 2020. URL https://doi.org/10.5281/zenodo.3602260.

[117] Evan Coleman et al. The importance of calorimetry for highly-boosted jet substructure. *Journal of Instrumentation*, 13(01):T01003–T01003, January 2018. ISSN 1748-0221. doi: 10.1088/1748-0221/13/01/t01003. URL http://dx.doi.org/10.1088/1748-0221/13/01/T01003.

[118] Torbjörn Sjöstrand et al. An Introduction to PYTHIA 8.2. *Comput. Phys. Commun.*, 191:159–177, 2015. doi: 10.1016/j.cpc.2015.01.024.

[119] Jérôme de Favereau et al. DELPHES 3, A modular framework for fast simulation of a generic collider experiment. *JHEP*, 02:057, 2014. doi: 10.1007/JHEP02(2014)057.

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

[121] Lisa Randall and Raman Sundrum. A Large mass hierarchy from a small extra dimension. *Phys. Rev. Lett.*, 83:3370, 1999. doi: 10.1103/PhysRevLett.83.3370.

[122] Matteo Cacciari et al. FastJet User Manual. *Eur. Phys. J. C*, 72:1896, 2012. doi: 10.1140/epjc/s10052-012-1896-2.

[123] Jai Bardhan et al. Unsupervised and lightly supervised learning in particle physics. *The European Physical Journal Special Topics*, 233(15–16):2559–2596, jul 2024. ISSN 1951-6401. doi: 10.1140/epjs/s11734-024-01235-x. URL http://dx.doi.org/10.1140/epjs/s11734-024-01235-x.

[124] Melissa van Beekveld et al. Combining outlier analysis algorithms to identify new physics at the LHC. *Journal of High Energy Physics*, 2021(9):24, 2021. doi: 10.1007/JHEP09(2021)024. URL https://doi.org/10.1007/JHEP09(2021)024.

[125] ATLAS Collaboration. Anomaly detection search for new resonances decaying into a Higgs boson and a generic new particle X in hadronic final states using $\sqrt{s} =$ 13 TeV collisions with the ATLAS detector. *Physical Review D*, 108(5), sep 2023. ISSN 2470-0029. doi: 10.1103/physrevd.108.052009. URL http://dx.doi.org/10.1103/PhysRevD.108.052009.

[126] ATLAS Collaboration. Search for new phenomena in two-body invariant mass distributions using unsupervised machine learning for anomaly detection at $\sqrt{s} = 13$ tev with the ATLAS detector. *Physical Review Letters*, 132(8), feb 2024. ISSN 1079-7114. doi: 10.1103/physrevlett.132.081801. URL http://dx.doi.org/10.1103/PhysRevLett.132.081801.

[127] Olmo Cerri et al. Variational autoencoders for new physics mining at the large hadron collider. *Journal of High Energy Physics*, 2019(5):36, 2019. doi: 10.1007/JHEP05(2019)036. URL https://doi.org/10.1007/JHEP05(2019)036.

[128] Ekaterina Govorkova et al. Autoencoders on field-programmable gate arrays for real-time, unsupervised new physics detection at 40 MHz at the Large Hadron Collider. *Nature Machine Intelligence*, 4(2):154–161, February 2022. ISSN 2522-5839. doi: 10.1038/s42256-022-00441-3. URL http://dx.doi.org/10.1038/s42256-022-00441-3.

[129] CMS Collaboration. Real-time Anomaly Detection at the L1 Trigger of CMS Experiment, 2024. URL https://arxiv.org/abs/2411.19506.

[130] Vishal S. Ngairangbam, Michael Spannowsky, and Michihisa Takeuchi. Anomaly detection in high-energy physics using a quantum autoencoder. *Physical Review D*, 105(9), may 2022. ISSN 2470-0029. doi: 10.1103/physrevd.105.095004. URL http://dx.doi.org/10.1103/PhysRevD.105.095004.

[131] Julian Schuhmacher et al. Unravelling physics beyond the standard model with classical and quantum anomaly detection. *Machine Learning: Science and Technology*, 4(4):045031, nov 2023. doi: 10.1088/2632-2153/ad07f7. URL https://dx.doi.org/10.1088/2632-2153/ad07f7.

[132] Elie Bermot et al. Quantum generative adversarial networks for anomaly detection in high energy physics. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*, page 331–341. IEEE, sep 2023. doi: 10.1109/qce57702.2023.00045. URL http://dx.doi.org/10.1109/QCE57702.2023.00045.

[133] Marco Farina, Yuichiro Nakai, and David Shih. Searching for new physics with deep autoencoders. *Physical Review D*, 101(7), apr 2020. ISSN 2470-0029. doi: 10.1103/physrevd.101.075021. URL http://dx.doi.org/10.1103/PhysRevD.101.075021.

[134] Kinga Anna Wozniak. *Machine learning applications for new physics searches at CERN's Large Hadron Collider*. PhD thesis, Wien, 2023. URL https://doi.org/10.25365/thesis.74519.

[135] Dawid Kopczyk. Quantum machine learning for data scientists, 2018. URL https://arxiv.org/abs/1804.10068.

[136] Sumsam Ullah Khan, Ahsan Javed Awan, and Gemma Vall-Llosera. K-means clustering on noisy intermediate scale quantum computers, 2019. URL https://arxiv.org/abs/1909.12183.

[137] Yehuda Vardi and Cun-Hui Zhang. The multivariate l_1-median and associated data depth. *Proceedings of the National Academy of Sciences*, 97(4):1423–1426, 2000. doi: 10.1073/pnas.97.4.1423. URL https://www.pnas.org/doi/abs/10.1073/pnas.97.4.1423.

[138] Carlile Lavor et al. Grover's algorithm: Quantum database search, 2003. URL https://arxiv.org/abs/quant-ph/0301079.

[139] Marcello Dalmonte and Simone Montangero. Lattice gauge theory simulations in the quantum information era. *Contemporary Physics*, 57(3):388–412, mar 2016. ISSN 1366-5812. doi: 10.1080/00107514.2016.1151199. URL http://dx.doi.org/10.1080/00107514.2016.1151199.

[140] Matthias Gerster et al. Fractional quantum hall effect in the interacting hofstadter model via tensor networks. *Phys. Rev. B*, 96:195123, Nov 2017. doi: 10.1103/PhysRevB.96.195123. URL https://link.aps.org/doi/10.1103/PhysRevB.96.195123.

[141] Mari Carmen Bañuls et al. Simulating lattice gauge theories within quantum technologies. *The European Physical Journal D*, 74(8):165, 2020. doi: 10.1140/epjd/e2020-100571-8. URL https://doi.org/10.1140/epjd/e2020-100571-8.

[142] Mari Carmen Bañuls and Krzysztof Cichy. Review on novel methods for lattice gauge theories. *Reports on Progress in Physics*, 83(2):024401, jan 2020. doi: 10.1088/1361-6633/ab6311. URL https://dx.doi.org/10.1088/1361-6633/ab6311.

[143] Simone Montangero, Enrique Rico, and Pietro Silvi. Loop-free tensor networks for high-energy physics. *Philosophical Transactions of the Royal Society A: Mathematical, Physical and Engineering Sciences*, 380(2216), dec 2021. ISSN 1471-2962. doi: 10.1098/rsta.2021.0065. URL http://dx.doi.org/10.1098/rsta.2021.0065.

[144] Timo Felser et al. Quantum-inspired machine learning on high-energy physics data. *npj Quantum Information*, 7(1):111, 2021. doi: 10.1038/s41534-021-00443-w. URL https://doi.org/10.1038/s41534-021-00443-w.

[145] Yannick Meurice et al. Tensor networks for high energy physics: contribution to Snowmass 2021, 2022. URL https://arxiv.org/abs/2203.04902.

[146] Giuseppe Magnifico et al. Tensor networks for lattice gauge theories beyond one dimension: a roadmap, 2024. URL https://arxiv.org/abs/2407.03058.

[147] Jack Y. Araz and Michael Spannowsky. Quantum-inspired event reconstruction with tensor networks: Matrix product states. *Journal of High Energy Physics*, 2021(8), aug 2021. ISSN 1029-8479. doi: 10.1007/jhep08(2021)112. URL http://dx.doi.org/10.1007/JHEP08(2021)112.

[148] Hans Hohenfeld et al. Explaining anomalies with tensor networks, 2025. URL https://arxiv.org/abs/2505.03911.

[149] Patrick Odagiu et al. Ultrafast jet classification at the HL-LHC. *Machine Learning: Science and Technology*, 5(3):035017, jul 2024. ISSN 2632-2153. doi: 10.1088/2632-2153/ad5f10. URL http://dx.doi.org/10.1088/2632-2153/ad5f10.

[150] CMS collaboration. Identification of b-quark jets with the CMS experiment. *Journal of Instrumentation*, 8(04):P04013, apr 2013. doi: 10.1088/1748-0221/8/04/P04013. URL https://dx.doi.org/10.1088/1748-0221/8/04/P04013.

[151] ATLAS Collaboration. Performance of b-jet identification in the ATLAS experiment. *Journal of Instrumentation*, 11(04):P04008, apr 2016. doi: 10.1088/1748-0221/11/04/P04008. URL https://dx.doi.org/10.1088/1748-0221/11/04/P04008.

[152] Chang Sun, Jennifer Ngadiuba, Maurizio Pierini, and Maria Spiropulu. Fast jet tagging with mlp-mixers on fpgas. *Machine Learning: Science and Technology*, 6 (3):035025, aug 2025. doi: 10.1088/2632-2153/adf596. URL https://doi.org/10.1088/2632-2153/adf596.

[153] Zhiqiang Que, Chang Sun, Sudarshan Paramesvaran, Emyr Clement, Katerina Karakoulaki, Christopher Brown, Lauri Laatu, Arianna Cox, Alexander Tapper, Wayne Luk, and Maria Spiropulu. Jedi-linear: Fast and efficient graph neural networks for jet tagging on fpgas, 2025. URL https://arxiv.org/abs/2508.15468.

[154] FastML Team. fastmachinelearning/hls4ml, 2023. URL https://github.com/fastmachinelearning/hls4ml.

[155] Patrick T. Komiske, Eric M. Metodiev, and Jesse Thaler. Energy flow networks: deep sets for particle jets. *Journal of High Energy Physics*, 2019(1):121, 2019. doi: 10.1007/JHEP01(2019)121. URL https://doi.org/10.1007/JHEP01(2019)121.

[156] Huilin Qu and Loukas Gouskos. Jet tagging via particle clouds. *Phys. Rev. D*, 101: 056019, Mar 2020. doi: 10.1103/PhysRevD.101.056019. URL https://link.aps.org/doi/10.1103/PhysRevD.101.056019.

[157] Yutaro Iiyama et al. Distance-weighted graph neural networks on FPGAs for real-time particle reconstruction in high energy physics. *Frontiers in Big Data*, Volume 3

- 2020, 2021. ISSN 2624-909X. doi: 10.3389/fdata.2020.598927. URL https://www.frontiersin.org/journals/big-data/articles/10.3389/fdata.2020.598927.

[158] Huilin Qu, Congqiao Li, and Sitian Qian. Particle transformer for jet tagging, 2024. URL https://arxiv.org/abs/2202.03772.

[159] Alexander Bogatskiy et al. Explainable equivariant neural networks for particle physics: PELICAN. *Journal of High Energy Physics*, 2024(3), March 2024. ISSN 1029-8479. doi: 10.1007/jhep03(2024)113. URL http://dx.doi.org/10.1007/JHEP03(2024)113.

[160] Jonathan Shlomi, Peter Battaglia, and Jean-Roch Vlimant. Graph neural networks in particle physics. *Machine Learning: Science and Technology*, 2(2):021001, dec 2020. doi: 10.1088/2632-2153/abbf9a. URL https://dx.doi.org/10.1088/2632-2153/abbf9a.

[161] Aritra Bal et al. 1 particle - 1 qubit: Particle physics data encoding for quantum machine learning, 2025. URL https://arxiv.org/abs/2502.17301.

[162] CMS Collaboration. The Phase-2 Upgrade of the CMS Level-1 Trigger. Technical report, CERN, Geneva, 2020. URL https://cds.cern.ch/record/2714892. Final version.

[163] Ema Puljak et al. Quantum anomaly detection in the latent space of proton collision events at the LHC, March 2023. URL https://doi.org/10.5281/zenodo.7711904.

[164] Vasilis Belis, Ema Puljak, and Kinga Anna Wozniak. vbelis/latent-ad-qml: v1.0.4. mar 2023. doi: 10.5281/zenodo.7696203. URL https://doi.org/10.5281/zenodo.7696203.

[165] Maurizio Pierini and Kinga Anna Wozniak. Dataset for Quantum anomaly detection in the latent space of proton collision events at the LHC, feb 2023. URL https://doi.org/10.5281/zenodo.7673769.

[166] CMS Collaboration. The CMS Experiment at the CERN LHC. *JINST*, 3:S08004, 2008. doi: 10.1088/1748-0221/3/08/S08004.

[167] CMS Collaboration. The CMS Silicon Strip Tracker. *AIP Conference Proceedings*, 842(1):1067–1069, 07 2006. ISSN 0094-243X. doi: 10.1063/1.2220455. URL https://doi.org/10.1063/1.2220455.

[168] CMS Collaboration. Operation and performance of the CMS tracker. *Journal of Instrumentation*, 9(03):C03005–C03005, mar 2014. ISSN 1748-0221. doi: 10.1088/1748-0221/9/03/c03005. URL http://dx.doi.org/10.1088/1748-0221/9/03/C03005.

[169] Paolo Azzurri. The CMS silicon strip tracker. *Journal of Physics: Conference Series*, 41(1):127, 2006. doi: 10.1088/1742-6596/41/1/011. URL https://dx.doi.org/10.1088/1742-6596/41/1/011.

[170] Erik Butz. Operation and performance of the CMS silicon detectors. *Nuclear Instruments and Methods in Physics Research Section A: Accelerators, Spectrometers, Detectors and Associated Equipment*, 1064:169377, 2024. ISSN 0168-9002. doi: https://doi.org/10.1016/j.nima.2024.169377. URL https://www.sciencedirect.com/science/article/pii/S0168900224003036.

[171] CMS Collaboration. Performance of the CMS electromagnetic calorimeter in pp collisions at $\sqrt{s} = 13$ TeV. *Journal of Instrumentation*, 19(09):P09004, September 2024. ISSN 1748-0221. doi: 10.1088/1748-0221/19/09/p09004. URL http://dx.doi.org/10.1088/1748-0221/19/09/P09004.

[172] CMS Collaboration. Calibration of the CMS hadron calorimeters using proton-proton collision data at $\sqrt{s} = 13$ TeV. *Journal of Instrumentation*, 15(05):P05002–P05002, May 2020. ISSN 1748-0221. doi: 10.1088/1748-0221/15/05/p05002. URL http://dx.doi.org/10.1088/1748-0221/15/05/P05002.

[173] CMS Collaboration. Performance of the CMS muon detector and muon reconstruction with proton-proton collisions at $\sqrt{s} = 13$ TeV. *Journal of Instrumentation*, 13(06):P06015–P06015, jun 2018. ISSN 1748-0221. doi: 10.1088/1748-0221/13/06/p06015. URL http://dx.doi.org/10.1088/1748-0221/13/06/P06015.