POLYTECHNIC UNIVERSITY OF BUCHAREST
FACULTY OF AUTOMATIC CONTROL AND COMPUTERS
COMPUTER SCIENCE AND ENGINEERING DEPARTMENT

BACHELOR THESIS

A comprehensive study on Integrating Quantum Physics in Deep Learning Networks for Optimizations in Computer Vision and Natural Language Processing Fields

QRKT-GAN: Neural Ordinary Differential Equation-Inspired Generative Adversarial Network with Numerical Runge-Kutta Methods for Quantum Visual Transformer-Based Generator and Discriminator

Cătălin-Alexandru Rîpanu

Thesis advisor:

Ș.l. dr. ing. Dumitru-Clementin Cercel

BUCHAREST
2024

ACKNOWLEDGEMENTS

First and foremost, I would like to express my deepest gratitude to my advisor, Prof. Dumitru-Clementin Cercel, for his unwavering support and insightful guidance throughout my research journey. His expertise and encouragement have been essential in exploring the intricate intersection of Machine Learning and Quantum Mechanics, widely referred to in the literature as Quantum Machine Learning. This interdisciplinary field, though initially daunting, is at the cutting edge of technological advancement and innovation in both areas.

I am also profoundly thankful to Prof. Pantelimon-George Popescu for his advice, recommendations, and exceptional lectures on Quantum Computing. His mentorship has been instrumental in building a robust understanding of this groundbreaking paradigm, which promises new solutions to problems previously deemed insurmountable by classical methods.

Furthermore, I wish to acknowledge the significant contributions of the academic teaching assistants and collaborators encountered during my bachelor's studies. Their guidance during challenging times has been crucial to my academic development. Last but not least, I am also grateful for the computational resources provided by the Computer Science and Engineering Department, which were essential for the design and implementation of this project.

In addition, I extend my appreciation to my friends for their continuous support, ideas, and advice, especially during moments when inspiration was scarce. Lastly, my heartfelt thanks go to my parents for their enduring sacrifices and unwavering support throughout this journey into the unknown. Their belief in me has been a constant source of motivation.

ABSTRACT

Deep Learning models, such as Generative Adversarial Networks (GANs) [1] and Visual Transformers (ViTs) [2, 3], have demonstrated remarkable results across various domains in Machine Learning and Artificial Intelligence, including Object Classification, Image Segmentation, Sentiment Analysis, and Synthetic Data Generation. These neural networks are pivotal in advancing systems that require high precision and a deep understanding of complex data across a variety of tasks in both Natural Language Processing (NLP) and Computer Vision (CV).

However, the effectiveness of Deep Learning models comes with significant challenges: they require an extensive [4] number of parameters to learn and extract meaningful features from real-world data. Additionally, these neural networks need vast amounts of information to achieve desired performance levels. Obtaining such large sets can be difficult as real-world data is often not publicly available and can be challenging to collect and curate. This results in substantial computational resource requirements for both training and hyperparameter optimization, often achieved through exhaustive techniques such as grid search [5].

To address these challenges and to further investigate this direction, this thesis proposes a novel hybrid Generative Adversarial Network architecture that employs Quantum Visual Transformers (QViTs) [6] as both the Generator and Discriminator. Visual Transformers are selected for their superior ability to manage intricate data representations. A key innovation in this architecture is the integration of Ordinary Differential Equation (ODE) [7] solvers as Encoders [8], enhancing the model capability to capture temporal dynamics and complex data structures, and improving the residual connections within the transformer architecture to mitigate the vanishing gradients problem [9] even more.

Moreover, this architecture incorporates Variational Quantum Circuits [10] within both the Self-Attention Mechanisms [11] and the Multi-Layer Perceptrons (MLPs) [12] of the Visual Transformers. By leveraging the principles of Quantum Physics, especially Mechanics, these quantum circuits [13] can perform complex algebraic operations more efficiently than classical methods, offering a significant computational advantage.

The performance of this hybrid approach is benchmarked against a purely classical neural network baseline from the literature using datasets from both CV and NLP areas. Specifically, the Quantum ODE configurations are tested on CIFAR-10 [14], MNIST [15] and IMDb Reviews [16] datasets. These neural network architectures are effectively trained and tested using numerical simulations tailored to Variational Quantum Circuits [17]. The obtained results indicate that these hybrid approaches achieves comparable classification and generation performance to the classical baseline, while requiring fewer trainable parameters.

Furthermore, the reduced parameter count in these hybrid models opens up the possibility of running it on Noisy Intermediate-Scale (NISQ) quantum devices for both training and inference. This feasibility is a significant breakthrough, as it implies that quantum-enhanced models can be trained and deployed on actual quantum computers, which are currently limited in terms of the number of qubits and operational fidelity [18].

This thesis demonstrates the potential of integrating Quantum Computing [19], especially Quantum Mechanics, with advanced Deep Learning architectures to create more efficient and powerful networks which can significantly reduce computational costs while maintaining high performance, paving the way for more scalable and effective AI applications [20].

ABSTRACT

Modelele de Învățare Profundă, cum ar fi Rețelele Adversariale Generative (GAN-uri) [1] și Transformers Vizuali (ViTs) [2, 3], au demonstrat rezultate remarcabile în diverse domenii ale Învățării Automate și Inteligenței Artificiale, inclusiv Clasificarea Obiectelor, Segmentarea Imaginilor, Analiza Sentimentelor și Generarea de Date Sintetice. Aceste rețele neuronale sunt esențiale în avansarea sistemelor care necesită precizie înaltă și o înțelegere profundă a datelor complexe într-o varietate de sarcini, atât în Procesarea Limbajului Natural (NLP), cât și în Vederea Computațională (CV).

Totuși, eficacitatea modelelor de Învățare Profundă vine cu provocări semnificative: acestea necesită un număr mare [4] de parametri pentru a învăța și a extrage caracteristici semnificative din datele reale. În plus, aceste rețele neuronale au nevoie de cantități vaste de informații pentru a atinge nivelurile de performanță dorite. Obținerea unor astfel de seturi mari de date poate fi dificilă, deoarece datele reale nu sunt adesea disponibile public și pot fi greu de colectat și filtrat. Acest lucru duce la cerințe semnificative de resurse computaționale atât pentru antrenare, cât și pentru optimizarea hiperparametrilor, adesea realizate prin tehnici exhaustive cum ar fi grid search [5].

Pentru a aborda aceste provocări și pentru a investiga mai departe această direcție, această teză propune o arhitectură hibridă nouă de Rețea Generativă Adversarială care utilizează Transformers Vizuali Cuantici (QViTs) [6] atât ca Generatori, cât și ca Discriminatori. Visual Transfomers sunt selectate pentru abilitatea lor superioară de a gestiona reprezentările de date complexe. O inovație cheie în această arhitectură este integrarea metodelor numerice ce rezolvă Ecuații Diferențiale Ordinare (ODE) [7] ca Encoders [8], îmbunătățind capacitatea modelului de a captura dinamica temporală și structurile complexe, și îmbunătățind conexiunile reziduale din cadrul arhitecturii Transformer-ului pentru a atenua problema vanishing gradients [9].

Mai mult, această arhitectură încorporează Circuite Variaționale Cuantice [10] atât în Mecanismele de Auto-Atenție [11], cât și în Perceptronii Multistrat (MLPs) [12] ai Visual Transformers. Prin valorificarea principiilor Fizicii Cuantice, în special Mecanica, aceste circuite cuantice [13] pot efectua operații algebrice complexe mai eficient decât metodele clasice, oferind un avantaj computațional semnificativ.

Performanța acestei abordări hibride este evaluată în raport cu o bază de date neurală pur clasică din literatură, folosind seturi de date din domeniile CV și NLP. Specific, configurațiile ODE cuantice sunt testate pe seturile de date CIFAR-10 [14], MNIST [15] și IMDb Reviews [16]. Aceste arhitecturi de rețele neuronale sunt antrenate și testate eficient utilizând simulări numerice adaptate pentru Circuitele Variaționale Cuantice [17]. Rezultatele obținute indică faptul că aceste abordări hibride obțin performanțe comparabile de clasificare și generare cu baza clasică, în timp ce necesită mai puțini parametri antrenabili.

În plus, numărul redus de parametri în aceste modele hibride deschide posibilitatea rulării acestora pe dispozitive cuantice NISQ pentru antrenare și inferență. Această fezabilitate este un progres semnificativ, deoarece implică faptul că modelele îmbunătățite cuantice pot fi antrenate și implementate pe calculatoare cuantice reale, care sunt în prezent limitate în termeni de număr de qubiți și fidelitate operațională [18].

Această teză demonstrează potențialul integrării Calculului Cuantic [19], în special Mecanica Cuantică, cu arhitecturi avansate de Învățare Profundă pentru a crea rețele eficiente [20].

CONTENTS

1 Introduction

1.1 Context

Artificial Intelligence models, particularly Deep Learning ones, have made significant contributions to solving real-world tasks, greatly improving human lives in various fields, such as Medical Image Recognition [21]. Despite their impressive capabilities, Deep Learning models come with substantial drawbacks regarding computational resources [4] and effort. Achieving high performance with these models necessitates learning millions to billions of parameters, also called weights or artificial neurons, which demands considerable resources and preparation time. This limitation also has negative environmental impacts due to high power consumption.

Over the years, researchers have developed numerous solutions to mitigate the problem of minimizing the number of parameters using interesting classical algorithms and techniques. These include, for example, specialized activation functions for neural layers like Rectified Linear Unit [22], Leaky ReLU [23], Gaussian Error Linear Unit [24], Softmax [25], Sigmoid [26] and Hyperbolic Tangent [27]. Additionally, effective optimizers such as AdamW [28] and Stochastic Gradient Descent [29] have been utilized to combat this limitation, along with methods like Learning Rate Schedulers [30], Weight Decay [31], and Dropout [32] to reduce also the chances of model overfitting. Over the years, various architectural designs and network combinations have emerged to address these challenges. However, the tradeoff between high computational resource consumption and performance remains a difficult issue, especially when scaling up the dimensions of available noisy real-world datasets used for training. It is unlikely[1] to be efficiently resolved in the near future using solely classical approaches.

A segment of Computer Science researchers, in collaboration with physicists, have taken a bold and innovative approach to these challenges by exploring solutions from a physical perspective, leading to the emergence of Quantum Computation and Quantum Information [19]. Leveraging the principles of Quantum Mechanics, AI research has begun to explore Deep Learning optimizations using quantum facilities, such as Variational Quantum Circuits [13] or Quantum Algorithms [10], to develop hybrid Deep Learning models. By combining classical and quantum methodologies, new experiments can be conducted to address performance and resource demands more efficiently, thus utilizing the unique capabilities of quantum hardware. This involves training these models on it, also known as Parameterized Quantum Circuits [13], where the parameters are referred to as the qurons of the quantum model.

Quantum Computing leverages properties like superposition [33] and entanglement [34], enabling the execution of complex algebraic operations that are infeasible for classical computers. Quantum Algorithms [10], such as Grover's Algorithm [35], Quantum Phase Estimation [36], Quantum Fourier Transform [37] and Deutsch-Jozsa Quantum Parallelism [38] provide exponential speedups for certain tasks. Integrating these Quantum principles with Deep Learning can significantly reduce the number of parameters and computational resources required while maintaining or even enhancing model performance. This interdisciplinary distinctive approach, also known as Quantum Machine Learning, opens up exciting possibilities for the future of AI, promising advancements that could transform how complex problems are approached and solved in ways that classical computers, with their current architecture, could never achieve.

[1]https://www.theverge.com/24066646/ai-electricity-energy-watts-generative-consumption

1.2 Problem

Over the years, numerous algorithms, techniques, network architectures, and methodologies have been proposed in the field of Deep Learning to tackle a wide array of tasks, such as Object Classification, Synthetic Data Generation and Detection, across various domains including Natural Language Processing, Computer Vision, Domain Adaptation [39], and Knowledge Distillation [40]. These approaches have achieved high performance metrics like Accuracy, Area Under the Curve (AUC), Receiver Operating Characteristic (ROC), and F1 scores. The advent of High Performance Computing (HPC) facilities, particularly classical parallelism, has enabled Deep Learning models to process billions of examples from noisy real-world datasets effectively, thus yielding impressive results.

Despite these advancements, a significant drawback persists: the enormous number of neurons and the corresponding computational resources required by these models. Traditional solutions in classical Deep Learning, such as Cross Validation [41], Hyperparameter Tuning techniques like Grid Search [5], Random Search [5] and Keras Tuner [42] are commonly employed, together with Layer Augmentation until model performance plateaus or overfits. To also mitigate overfitting, regularization techniques such as Ridge, Lasso, Elastic-Net [43] or Dropout [32] can be applied. However, these methods often involve exhaustive hyperparameter searches, which are extremely time-consuming, especially when dealing with validation sets comprising billions of data points for complex tasks. Group sparsity regularizers can also be applied to network parameters, where each group acts on a single neuron, thus reducing the number of parameters by up to 80% while maintaining or even enhancing the network accuracy score [44].

Another method employed to address these challenges is Transfer Learning [45], which involves finding and loading a pretrained model as a starting point and then fine-tuning it on a specific dataset. This approach can help achieve good performance with less effort on architecture design and expedite the training process. However, the availability of suitable pretrained models is limited, and fine-tuning may yield only decent, if not disappointing, results, due to differences in the data distributions between the pretraining and fine-tuning datasets.

The primary challenge lies in balancing time, memory and power consumption on classical systems to make robust and valid neural architectural choices. Furthermore, reducing the number of neurons too drastically is not a solution as it can lead to underfitting and degraded performance, making it difficult to solve complex and challenging tasks with decent results. The iterative process of tuning hyperparameters and optimizing model architectures is computationally intensive and often results in a significant trade-off between resource consumption and model performance [5].

Moreover, the reliance on vast amounts of training data for achieving desired performance further exacerbates these issues. Real-world datasets are often difficult to obtain and may not be publicly available, which complicates the development and fine-tuning of effective Deep Learning models. The extensive time and computational costs associated with hyperparameter optimization and model training present significant obstacles to advancing AI capabilities while maintaining efficiency and sustainability. This trade-off between computational resource consumption and performance remains a persistent and challenging problem in the field of Deep Learning, highlighting the need for innovative solutions that can overcome these limitations.

1.3 Objective

In the current landscape of Deep Learning research, there is an ongoing investigation into whether integrating models into a purely quantum environment can significantly reduce power consumption and the number of trainable parameters without sacrificing performance. This exploration leverages techniques such as the Ansatz [46] and Variational Quantum Circuits [10]. As classical supercomputers advance to handle increasingly complex deep models, their power consumption grows almost exponentially. In contrast, quantum machines exhibit an exponential increase in computational power while maintaining a linear increase in power consumption. This intriguing characteristic makes a strong case for Quantum Computing: with the advent of fault-tolerant quantum computers, qubits could serve as highly efficient artificial neurons in neural networks. Quantum-inspired techniques could enable networks to operate with a vast number of neurons per layer at minimal energy cost, dramatically reducing overall energy consumption[2].

Building on the challenges outlined in the above section, the primary goal of this thesis is to explore Deep Learning models, both pure Quantum and hybrid, that leverage neural ODE-based architectures [47] inspired by non-trivial Variational Quantum Circuits [10] within the context of Visual Transformers [2, 3] and their complex Encoders [8] and Decoders [48]. The most important objective is to minimize the number of parameters used by deep network layers utilizing the idea [49] of replacing classical linear projection layers in Multi-Head Attention [11] subroutines and Multi-Layer Perceptrons [12] with Quantum Circuits [13]. Additionally, low truncation error Neural Ordinary Differential Equation [47][50] techniques will be employed to further optimize Transformer Layers, namely Runge-Kutta Methods [51] of 1st, 2nd, 3rd, and 4th order. An analysis of Runge-Kutta 4th order method optimization which is not theoretically grounded in standard RK methods, but in terms of training, will also be included [50].

To better understand the rationale and motivation behind the proposed Deep Learning model in this paper, it is necessary to compare these Quantum ODE-based Transformers configurations with their classical counterparts using the datasets mentioned in Abstract. The network utilizing the optimized Runge-Kutta 4th order method will be leveraged within the context of Vision Transformers (ViTs) [2, 3] and integrated into a Generative Adversarial Network (GAN) [1] architecture. In this configuration, both the Generator and Discriminator will be Quantum Visual Transformers (QViTs)[3], incorporating the optimized Neural ODE solver to generate realistic and informative synthetic data without being limited to specific types of distributions. This will be achieved using techniques such as Data Augmentation [52] and Image Recognition at Scale [3] in the context of ViTs.

The final proposed architecture in this thesis, named QRKT-GAN, will be tested on a real-world consistent dataset, namely CIFAR-10 [14], and compared using the same data with a renowned classical counterpart, TransGAN [53], which is a strong architecture built entirely free of Convolutions [54]. These comparative analysis and results will provide another argument for the possible advantages of integrating Quantum Computing [19] with advanced Deep Learning neural networks under the right context.

[2]`https://www.eetimes.eu/how-Quantum-Computing-can-help-make-ai-greener`
[3]`https://openreview.net/pdf?id=p7xPXoKB0H`

1.4 Paper Structure

The next chapter delves into the most relevant and important classical and Variational Quantum Algorithms [10] and techniques developed over the years to leverage both Quantum and hybrid logic in Deep Learning for model optimization. This chapter will illustrate how these both hybrid and pure Quantum Neural Networks (QNNs) [48] are designed to be compatible with Noisy Intermediate-Scale (NISQ) [18] devices provided by specialized vendors such as IBM [55], Google [56], Microsoft [57], and Amazon [58]. It will explore the advancements that make it possible to run complex Deep Learning models on quantum computers, highlighting the interplay between Quantum Computing [19] and Deep Learning for neural depth decreasing while not losing too much performance.

Section three will discuss the foundational concepts of Machine Learning, Deep Learning and Ordinary Differential Equations. It will also provide a comprehensive overview of Quantum Computation and Quantum Information [19], and will also introduce the emerging field of Quantum Machine Learning [59]. This section aims to explain how quantum principles can enhance the capabilities of Deep Learning solutions in addressing high-demanding tasks, some of which are classified as NP-Hard [60] or NP-Complete [61]. By understanding these fundamentals, the reader will gain insight into the potential synergies between Classical and Quantum Computing [19], which together can form a duality in the world of Artificial Intelligence.

Section four will present an in-depth analysis of the proposed Ordinary Differential Equation-based Quantum Neural Networks (QNNs) [62, 50] configurations. This includes their motivation, underlying concepts, mathematical and quantum frameworks, but also their physical limitations, and future potential within their problem domain environment. The chapter will also discuss in-detail the new ODE-based GAN architecture in detail, QRKT-GAN, explaining how it incorporates the traversed knowledge to minimize the solution of the optimization problem outlined in 1.1 section, while maintaining acceptable performance in Synthetic Image Generation.

The fifth chapter will detail the methodologies adopted for conducting the experiments. This includes a comprehensive description of the datasets used, their internal characteristics, and the reasons for their selection. It will compare various Quantum ODE-based [7] configurations with their classical ODE-based networks [47] counterparts to fully understand the benefits and limitations of quantum-enhanced models. Additionally, the chapter will provide a detailed comparison between the proposed QRKT-GAN model and its classical baseline, TransGAN [53], using the CIFAR-10 [14] dataset. This analysis will highlight the benefits and challenges of integrating Quantum Computing into Deep Learning, presenting it as a viable alternative solution for very specific optimization problems.

The final chapter will summarize the main ideas presented in the thesis, highlighting the relevance of the results and drawing key conclusions. It will also propose future work and outline potential directions for further research in the field of Quantum Deep Learning models, focusing on power consumption, computational resources and performance optimization. This chapter will emphasize the importance of continued exploration in combining Quantum Computing with Machine Learning [59] to achieve more efficient and powerful neural solutions.

2 Related Work

The relentless demand for improving human lives continues unabated, with increasingly complex and data-intensive needs emerging. This necessitates the development of sophisticated models capable of delivering accurate and reliable results. However, the complexity and depth of these models often result in significantly high computational resource consumption, which has environmental implications due to the energy required for such intensive processing.

Over the years, numerous classical, quantum, and hybrid methods have been proposed to address the optimization problem of minimizing model depth without significantly sacrificing performance. One notable approach is real-time learning of the number of neurons in deep networks using structured sparsity during training, which dynamically adjusts the network structure to optimize performance without excessive resource use [44]. Another method involves crafting specialized architectures that maintain a constant computational budget while slightly increasing depth to improve performance. This is achieved by dimensionality reduction before applying expensive Convolutions [30] with larger patch sizes [63]. Unfortunately, Convolutions [30] have been shown [54] to be less effective in capturing global feature interactions, as they primarily focus on local patterns in data processing. In contrast, the Self-Attention mechanism [11] utilized in Transformers [2] excels at capturing these global interactions.

In addition, some techniques focus on minimizing weights during training by splitting network weights into sets or hierarchies of multiple groups, each using disjoint sets of features. This allows parallelization, even in an embarrassingly manner, as each subnetwork operates independently, enhancing computational efficiency [64]. Alternatively, population-based heuristic algorithms, such as Particle Swarm Optimization (PSO), have shown promise in optimally determining the number of parameters without exhaustive searches like grid search [5], and thus saves valuable computational resources during the tuning process of these Deep Learning models [65]. Another approach involves reducing model parameters in deep neural networks via product-of-sums matrix decompositions, which decompose linear operators as a product of sums of simpler linear operators [66]. For Deep Convolutional Neural Network [30] models, kernel-sharing between multiple convolutional layers can also be applied [67]. Kernel-sharing is possible only between layers having the same kernel size, input, and output channels.

Despite their advantages, these classical methods have inherent drawbacks. Designing neural architectures from scratch to meet specific requirements is challenging and time-consuming. Moreover, methods like Particle Swarm Optimization [65] require optimization of additional parameters, such as activation functions [26, 22, 23] and the number of epochs, to be truly effective. Quantum Neural Networks (QNNs) [62] and Quantum Algorithms [10] offer new avenues for addressing these limitations, providing potential solutions for power-efficient Deep Learning. Various Quantum frameworks, technologies, GPU-based numerical simulators, and access to real hardware have been developed to facilitate the implementation of quantum phenomena in Machine Learning.

However, the quantum approach shares a common challenge with classical methods: the need for specific adaptations for each type of Deep Learning model. In the classical domain, the entire architectural framework must be meticulously analyzed to prevent performance degradation. In quantum domain, constructing valid Variational Quantum Circuits (VQCs) [13] and Algorithms [10] that efficiently mimic classical functions is crucial to avoid the same pitfalls.

3 Background

This chapter will delve into the main and most important advancements in Machine Learning, presenting key concepts as they appeared in the literature. It will discuss the motivation and necessity of Deep Learning, a fundamental subdomain of Machine Learning, to fully understand the creation and utilization of neural networks. This will include an exploration of why Generative Adversarial Networks (GANs) [1] and Visual Transformers [2, 3] were developed, along with a detailed examination of their architectural specificities and functions.

Furthermore, the chapter will provide an overview of the techniques used to identify optimal hyperparameters, which are crucial for further neural model optimization. This will cover various approaches to fine-tuning Deep Learning models to enhance their performance.

A significant portion will be dedicated to Ordinary Differential Equations (ODEs) and their most popular numerical solvers, highlighting the foundational aspects of the Runge-Kutta methods [51]. It will explain how to interpret Transformer layers from an ODE perspective [50], providing a mathematical framework for understanding these non-trivial neural network structures.

Furthermore, a concise summary of Quantum Computing [19] will also be presented, establishing the prerequisites for understanding Quantum Machine Learning [59]. This section will articulate the motivation behind integrating Runge-Kutta ODE methods [51, 7] into Quantum Computing [19]. By exploring carefully Quantum Gates and Circuits [68, 13], Variational Quantum Algorithms and Circuits [10, 13], their motivation and application in the context of Quantum Transformers [49] will be illustrated.

By comprehensively covering these topics, the chapter aims to equip readers with a deep understanding of the theoretical and practical foundations necessary for advancing in the field of Quantum Machine Learning [59], particularly through the integration of ODE methods [47] in quantum frameworks [17].

3.1 Machine Learning

Machine Learning is the process of enabling computers to learn from data and make decisions without being explicitly programmed to do so. A crucial aspect of Machine Learning is training models on data before they can autonomously make decisions. Thus, data is fundamental to Machine Learning algorithms. Without data, Machine Learning cannot function, as models rely entirely on the information provided to make future predictions. The data used must accurately reflect real-world scenarios because Machine Learning models depend on this data to learn and generate accurate predictions for new, unseen data [69].

In practice, Machine Learning involves extensive mathematical operations such as matrix multiplication, feature scaling, and normalization. Consequently, the data fed into Machine Learning algorithms must be represented as numerical vectors or numbers, as opposed to text or other non-numerical forms that machines cannot directly interpret.

For instance, textual data and alphabets are not directly understandable by machines; hence, they must be converted into numerical form before being processed by Machine Learning models. Techniques such as one-hot encoding, word embeddings, or other numerical transformation methods are used to convert text into a machine-readable format, enabling models to learn from and predict based on the data provided.

Machine Learning algorithms, along with neural networks, have been proposed for many decades. However, during the early days, the lack of sufficient data and limited computational power hindered their practical application. Today, the scenario has drastically changed. Companies generate vast amounts of data, and computational resources have become extraordinarily powerful and accessible. Services like Google Cloud and Amazon Web Services (AWS) provide scalable computational power, allowing users to run complex Machine Learning algorithms without the need for setting up their own hardware infrastructure. This has led to an exponential growth in the demand for Machine Learning and Deep Learning applications [69].

Understanding Machine Learning and Deep Learning is crucial in this data-rich environment. The ability to harness data through these technologies enables companies to make informed predictions, optimize processes, and drive innovation. The more effectively Machine Learning models are trained and utilized, the greater the value they can bring to an organization, ultimately contributing to increased profitability and competitive advantage.

Types of Machine Learning:

- **Supervised Learning**: This approach involves training a Machine Learning model on a labeled dataset, where the output labels are known. The model learns to map inputs to outputs based on this labeled data. During training, the model predictions are compared with the actual labels to evaluate its performance and adjust its parameters accordingly. This iterative process continues until the model achieves satisfactory accuracy. An example of supervised learning is predicting house prices. Given a dataset with features like square footage, number of bedrooms, and location, along with corresponding house prices, the model learns to predict the price of a house based on these features. By comparing its predictions with the actual prices, the model improves over time.
- **Unsupervised Learning**: In this approach, the model is trained on data without labeled outputs. The goal is for the model to identify patterns, structures, or relationships within the data. One common application of unsupervised learning is customer segmentation, where customers are grouped based on their behavior, such as purchase history or browsing patterns, without predefined labels. This helps businesses tailor marketing strategies to different customer segments.
- **Semi-Supervised Learning**: This technique combines elements of both supervised and unsupervised learning. It uses a small amount of labeled data along with a larger amount of unlabeled data. The model is initially trained on the labeled data, and then it generalizes its learning to the unlabeled data. This approach is useful when obtaining labeled data is expensive or time-consuming. An example of semi-supervised learning is text document classification. A model is first trained on a subset of documents with known categories. Then, it is used to classify the remaining documents, which do not have predefined labels, based on the patterns it has learned.

3.2 Gradient Descent

Gradient Descent (GD) is a foundational first-order optimization algorithm widely used in Machine Learning (ML) and Deep Learning (DL) for minimizing cost or loss functions. This iterative method works by iteratively adjusting the parameters of a model to find the local minimum or maximum of a given function, thereby improving the model accuracy. For instance, in linear regression, Gradient Descent helps minimize the difference between predicted and actual values by updating the weights and biases. Its significance and simplicity make it a fundamental topic in nearly all introductory Machine Learning courses, as it provides a clear understanding of how models learn and improve from data [70].

The Gradient Descent algorithm is not universally applicable to all functions. For it to work effectively, the function must meet two specific criteria: it must be differentiable and convex.

Mathematically:

- A function is differentiable if it has a derivative at each point in its domain. This means the function must be smooth and continuous without any sharp corners or discontinuities. Differentiability is crucial because Gradient Descent relies on the gradient (the derivative) to determine the direction and magnitude of the steps to take towards the minimum or maximum. If the function is not differentiable, the algorithm cannot compute the gradient accurately, leading to unreliable optimization results.

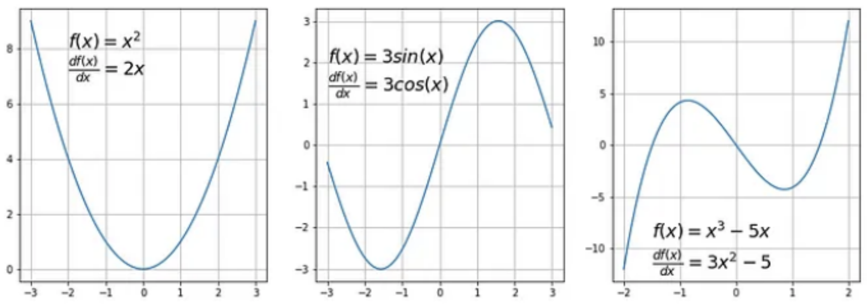


Figure 1: Examples of differentiable functions. Image from [70].

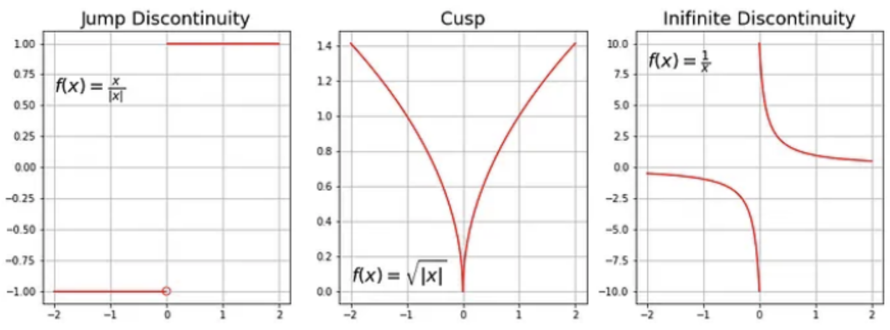


Figure 2: Examples of non-differentiable functions [70].

- A function is convex if the line segment between any two points on the function graph lies above or on the graph. Formally, a function $f(x)$ is said to be convex if for all x_1 and x_2 in its domain, and for all λ in the interval $[0,1]$, the following inequality holds:

$$f(\lambda x_1 + (1-\lambda)x_2) \leq \lambda f(x_1) + (1-\lambda)f(x_2) \tag{1}$$

  Convexity ensures that any local minimum is also a global minimum, making it easier for Gradient Descent to find the optimal solution.

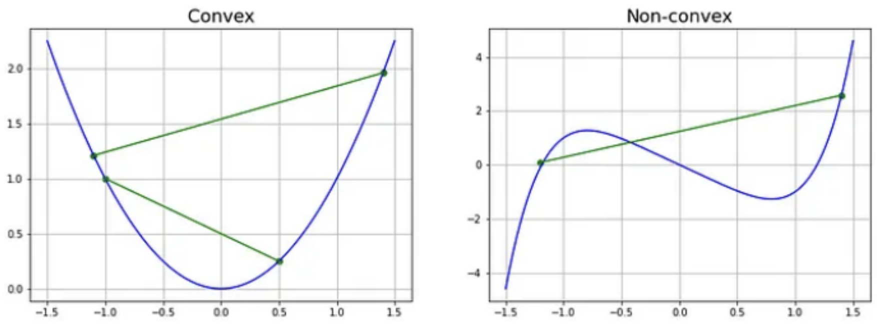


Figure 3: Examples of convex and non-convex functions [70].

Gradient Descent is an optimization algorithm used to minimize a function by iteratively moving towards the steepest descent direction. At each step, it calculates the gradient of the function at the current point, scales it by a learning rate, and updates the position by moving in the opposite direction of the gradient. This process ensures that the algorithm moves towards the function minimum. The update rule can be expressed as:

$$\mathbf{a}_n = \mathbf{a}_{n-1} - \eta \nabla f(\mathbf{a}_{n-1}) \tag{2}$$

where:

- $\mathbf{a}_{n-1}$ is the current position.
- η is the learning rate.
- $\nabla f(\mathbf{a}_{n-1})$ is the gradient of the function at the current position.

By repeatedly applying this update rule, Gradient Descent converges to the minimum of the function. Furthermore, for maximizing the given function, the algorithm would instead add the scaled gradient [70].

The learning rate, denoted as η, is a crucial hyperparameter in gradient descent algorithms that determines the step size at each iteration while moving toward a minimum of the loss function. The choice of learning rate significantly affects the performance of the algorithm:

- **Small Learning Rate**: If the learning rate is too small, the convergence process will be slow, potentially requiring many iterations to reach the optimal point. In some cases, the algorithm may even hit the maximum number of iterations before converging.

- **Large Learning Rate**: If the learning rate is too large, the algorithm might not converge to the optimal point. Instead, it could overshoot the minimum, causing the algorithm to oscillate or even diverge entirely, failing to find a satisfactory solution.

Thus, selecting an appropriate learning rate is essential for the efficient and effective training of Machine Learning models [70].

3.2.1 AdamW Optimizer

Adam combines the strengths of both Stochastic Gradient Descent (SGD) [71] with Momentum and Root Mean Square Propagation (RMSProp). It calculates the exponential moving averages of the gradient first and second moments. Like SGD with Momentum, it fully helps smooth out the updates to accelerate convergence. Similar to RMSProp, Adam uses adaptive learning rates for each parameter, allowing smaller steps for steep gradients and larger steps for flatter areas. Additionally, Adam adjusts these learning rates by considering both the mean of the first moment (the gradient) and the mean of the second moment (the squared gradient). The parameter update rule in Adam is given by [1]:

$$\theta_{t+1} = \theta_t - \eta \frac{\hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon} \tag{3}$$

$$m_t = \beta_1 m_{t-1} + (1 - \beta_1) \nabla_\theta f(\theta_t) \tag{4}$$

$$v_t = \beta_2 v_{t-1} + (1 - \beta_2)(\nabla_\theta f(\theta_t))^2 \tag{5}$$

where:

- θ is the parameter to be updated
- η is the initial learning rate
- β_1 is a decay rate for the first moment, with a default value of 0.9
- β_2 is a decay rate for the second moment, with a default value of 0.999
- ϵ is a small number used to avoid division by zero
- $\nabla_\theta f(\theta)$ is the gradient of the loss function $f(\theta)$

The exponential moving average of the squared gradients v_t in the last equation is sometimes referred to as the second moment or uncentered variance, due to the fact that the mean of the gradients is not subtracted. Additionally, m_t is often called the first moment or mean, representing the exponential moving average of the gradients, which captures the cumulative history of gradients. What is more, in Adam optimizer, the direction of the update is determined by normalizing the first moment with respect to the second moment [72].

Initially, the moving averages m_0 and v_0 are set to zero vectors. This initialization causes the moment estimates to be biased towards zero, particularly during the initial timesteps, especially when the decay rates β_1 and β_2 are close to 1. To address this issue, bias-corrected estimates

[1] `https://www.linkedin.com/pulse/understanding-adam-adamw-dsaisolutions-ileof/`

$\hat{m}_t$ and $\hat{v}_t$ are calculated. This correction helps control the weights while approaching the global minimum, preventing high oscillations. The formulas for the bias corrections are as follows [72]:

$$\hat{m}_t = \frac{m_t}{1 - \beta_1^t} \tag{6}$$

$$\hat{v}_t = \frac{v_t}{1 - \beta_2^t} \tag{7}$$

If no correction is applied, with $\beta_1 = 0.9$ and $\beta_2 = 0.999$, for the initial step, there are $m_1 = 0.1 g_t$ and $v_1 = 0.001 g_t^2$, causing the denominator in the update rule to become large. However, with bias-corrected estimates, the denominator becomes less sensitive to β_1 and β_2. The default values of β_1 and β_2 are set to 0.9 and 0.999, respectively, to retain as much historical gradient information as possible.

In practice, β_2 is usually set much closer to 1 than β_1, following the recommendation of the authors ($\beta_2 = 0.999$, $\beta_1 = 0.9$). Consequently, the update coefficient $1 - \beta_2 = 0.001$ is significantly smaller than $1 - \beta_1 = 0.1$. By accounting for both the first and second moments when updating parameters, Adam improves optimization efficiency, especially in complex models with numerous parameters.

However, Adam can be sensitive to the initial learning rate and other hyperparameters, which can affect its convergence and stability. Moreover, it may overfit, particularly with small datasets.

To address the generalization issue, Ilya Loshchilov and Frank Hutter introduced AdamW, an improved version of Adam, in their 2019 paper "Decoupled Weight Decay Regularization" [2]. Unlike Adam, which implicitly links weight decay to the learning rate, AdamW decouples weight decay from the optimization process.

This separation allows the learning rate and weight decay to be optimized independently. As a result, adjusting the learning rate does not necessitate recalculating the optimal weight decay, leading to more stable and effective optimization.

3.3 Deep Learning

Deep Learning is a subfield of Machine Learning that focuses on algorithms inspired by the structure and function of the brain neural networks. Unlike traditional Machine Learning algorithms that often require manual feature extraction and rely on simpler linear or non-linear models, Deep Learning employs complex architectures known as Artificial Neural Networks (ANNs). These neural networks are designed to automatically and hierarchically extract features from raw data, making them particularly powerful for tasks involving large and unstructured datasets.

[2] `https://openreview.net/forum?id=Bkg6RiCqY7`

Deep Learning models, particularly those involving deep neural networks (DNNs), consist of multiple layers of interconnected neurons. Each neuron in a layer processes input data, applies a transformation using weights and biases, and passes the output to the next layer. The hierarchical nature of these layers allows Deep Learning models to learn increasingly abstract and complex representations of the data as it moves through the network. This capability enables Deep Learning models to perform exceptionally well on tasks such as Image and Speech Recognition, Natural Language Processing, and Game Playing.

One of the key advantages of Deep Learning is its ability to learn from vast amounts of data without the need for extensive manual feature engineering. For instance, in Image Recognition tasks, traditional Machine Learning approaches might require handcrafted features such as edges, textures, and shapes. In contrast, Deep Learning models can automatically learn to detect these features directly from the pixel data [73].

Deep Learning has achieved remarkable success across various domains, driven by advancements in computational power (especially GPUs), large datasets, and novel network architectures like Convolutional Neural Networks (CNNs), Recurrent Neural Networks (RNNs), and Transformers [2]. CNNs, for example, are particularly well-suited for image-related tasks, while RNNs and Transformers [2] excel in Sequential Data Processing, such as Language Modeling and Translation.

3.3.1 Artificial Neural Networks (ANN)

An artificial neural network (ANN) is composed of interconnected nodes, also known as artificial neurons or units, organized into layers. These layers include the input layer, one or more hidden layers, and the output layer. Because of the numerous layers, ANNs are sometimes referred to as deep neural networks when they have many hidden layers [74].

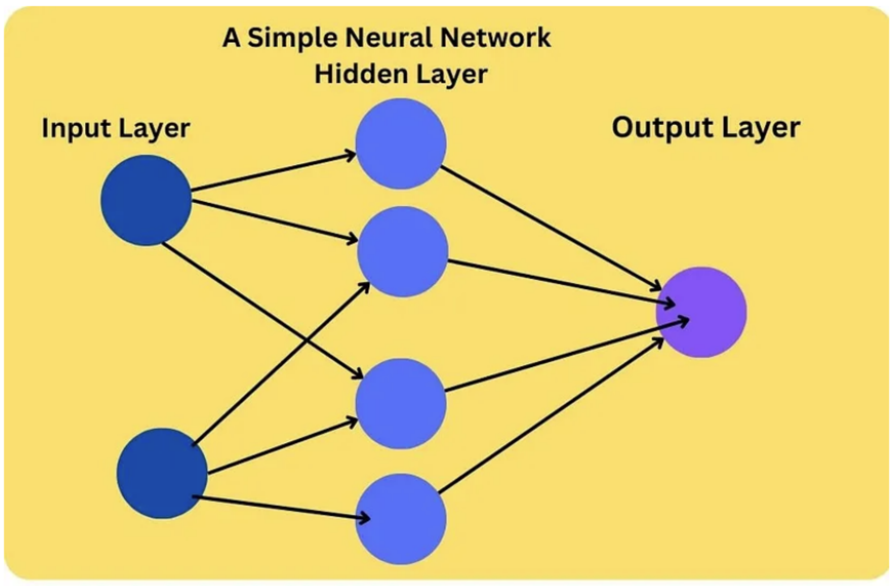


Figure 4: A simple Artificial Neural Network [74].

The three main layers in most ANNs are:

- **Input Layer**: This layer receives the raw input data, which can be in various forms such as images, text, or numerical values.
- **Hidden Layers**: Located between the input and output layers, these layers are responsible for processing the input data through weighted connections and activation functions.
- **Output Layer**: The output layer produces the final results of the neural network's processing, such as classifications or numerical predictions.

Training the neural network involves adjusting the weights of the connections to minimize the error in the output. Artificial neural networks have been successfully applied to a wide range of tasks, including Image Recognition, Natural Language Processing, Speech Recognition, Recommendation Systems, Financial Predictions, and many others. Their ability to learn and adapt from data makes them a powerful tool in Machine Learning and Artificial Intelligence.

3.3.2 Multi-Layer Perceptron (MLP)

Multilayer Perceptrons (MLPs) were initially inspired by the Perceptron, a supervised Machine Learning algorithm designed for binary classification. The original Perceptron was only capable of handling linearly separable data. To overcome this limitation, the Multilayer Perceptron was introduced, enabling the handling of both linearly and non-linearly separable data [75].

A Multilayer Perceptron is a type of neural network that belongs to the class of feed-forward neural networks. In these networks, the neurons in one layer are connected to the neurons in the subsequent layer in a forward manner, without any loops.

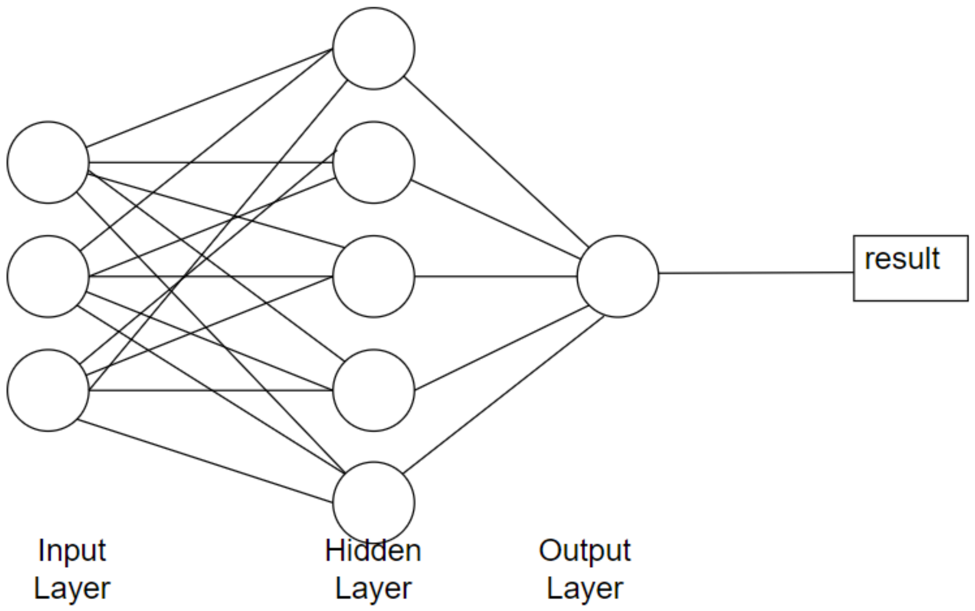


Figure 5: Multilayer Perceptron [75].

Furthermore, MLP is also a type of Artificial Neural Network (ANN) characterized by a layered architecture of interconnected neurons designed to process data through multiple stages. The fundamental structure of an MLP comprises an input layer, one or more hidden layers, an output layer, activation functions, and a set of weights and biases:

- **Input Layer**: This is the initial layer that receives input data, which can be numerical values, images, or other types of structured data.
- **Hidden Layers**: Positioned between the input and output layers, these layers are responsible for processing the input data through complex computations. There is no fixed limit to the number of hidden layers, though MLPs usually feature a modest number to balance complexity and computational efficiency.
- **Output Layer**: The final layer that delivers the output results derived from the data processed throughout the network.

A defining feature of MLPs is the use of backpropagation, a supervised learning technique for training neural networks. Backpropagation involves adjusting the weights in the network by propagating the error from the output back through the network. This iterative process fine-tunes the network parameters, enhancing performance and minimizing errors.

Due to their straightforward design, MLPs typically require shorter training times to learn data representations and produce outputs. However, they often necessitate more powerful computing resources than standard computers, particularly devices equipped with Graphics Processing Units (GPUs), to handle the intensive computations involved.

MLPs have proven to be versatile and efficient tools in various applications, from image recognition and natural language processing to financial forecasting and beyond. Their ability to learn from data and improve over time underscores their importance in the realm of Machine Learning and Artificial Intelligence.

3.3.3 Generative Adversarial Network (GAN)

Generative Adversarial Networks (GANs) [1] have revolutionized the field of Artificial Intelligence, capturing widespread attention for their ability to generate and enhance data in innovative ways. They are pivotal in applications such as creating photorealistic images, transforming styles in artwork, and generating lifelike human faces. Introduced in a groundbreaking 2014 NeurIPS paper[1], GANs are specialized Machine Learning systems designed to mimic specific data distributions [76].

At the heart of a GAN are two dynamically interacting neural networks: the Generator, which creates data, and the Discriminator, which evaluates the authenticity of the generated data. This dynamic interaction forms an adversarial training process, conceptualized as a competitive game. Here, the Generator strives to produce data so realistic that the Discriminator cannot reliably tell it apart from genuine data, achieving a 50% deception rate.

A Generative Adversarial Network comprises two neural networks that engage in simultaneous adversarial training:

- **Generator**: This network transforms random noise into coherent data outputs, such as images. Its objective is to generate data that closely resembles real-world data.
- **Discriminator**: This network evaluates both real data and data generated by the Generator, attempting to distinguish between the two. It outputs the likelihood that the provided data is real.

Throughout training, the Generator continuously improves its ability to create data that the Discriminator cannot distinguish from real data. Concurrently, the Discriminator enhances its skill in differentiating authentic data from the generated data. This adversarial game leads to the Generator producing progressively more realistic data over time.

Generative Adversarial Networks (GANs) employ loss functions to train both the generator and the discriminator, ensuring they both improve over time. The loss function helps with adjusting the weights of these models during training to enhance their performance. Both the generator and the discriminator utilize the binary cross-entropy loss, which can be expressed as:

$$L(y, p) = -(y \log(p) + (1 - y) \log(1 - p)) \tag{8}$$

where:

- $L(y, p)$ is the loss value.
- y is the true label (either 0 or 1).
- p is the predicted probability of the sample belonging to class 1.

The discriminator objective is to correctly classify real samples as real and fake samples, generated by the generator, as fake. The loss function for the discriminator is typically represented as:

$$L_D = -\frac{1}{2}\mathbb{E}_{x\sim p_{data}(x)}[\log(D(x))] - \frac{1}{2}\mathbb{E}_{z\sim p_z(z)}[\log(1 - D(G(z)))] \tag{9}$$

where:

$$\mathbb{E}_{x\sim p_{data}(x)}[f(x)] \approx \frac{1}{N}\sum_{i=1}^{N} f(x_i) \tag{10}$$

$$\mathbb{E}_{z\sim p_z(z)}[f(z)] \approx \frac{1}{M}\sum_{i=1}^{M} f(z_i) \tag{11}$$

In these equations:

- x_i are samples from the real dataset.
- N is the number of samples from the real dataset.
- z_i are samples from the noise distribution.

- M is the number of samples from the noise distribution.

The first term on the right-hand side penalizes the discriminator for misclassifying real data, while the second term penalizes the discriminator for misclassifying the fake data produced by the generator.

The goal of the generator in a Generative Adversarial Network (GAN) is to produce samples that the discriminator cannot distinguish from real data. The generator loss function is designed to penalize the generator when the discriminator correctly identifies its outputs as fake. Mathematically, the generator loss can be expressed as [76]:

$$L_G = -\frac{1}{2}\mathbb{E}_{z\sim p_z(z)}\left[\log(D(G(z)))\right] \tag{12}$$

Here, z represents the noise input to the generator drawn from a prior distribution $p_z(z)$, $G(z)$ is the generated sample, and $D(G(z))$ is the discriminator probability that $G(z)$ is a real sample.

The combined GAN loss, also known as the minimax loss, encapsulates the adversarial nature of GAN [**?**]raining. In this setup, the generator and discriminator engage in a two-player minimax game where the discriminator aims to maximize its ability to classify real and fake data correctly, while the generator strives to minimize the discriminator ability by generating realistic data. The combined loss is given by:

$$L_{GAN} = \min_G \max_D (L_D + L_G) \tag{13}$$

Here, L_D is the loss for the discriminator, and L_G is the loss for the generator.

Gradient penalty is a technique used to stabilize the training by penalizing the gradients if they become too steep. This can help in stabilizing the training and avoiding issues like mode collapse. The gradient penalty term is defined as:

$$GP = \lambda\mathbb{E}_{\hat{x}\sim P_{\hat{x}}}\left[(\|\nabla_{\hat{x}}D(\hat{x})\|_2 - k)^2\right] \tag{14}$$

Where:

- $\mathbf{GP}$ represents the gradient penalty term.
- λ is a hyperparameter that controls the strength of the penalty.
- The gradient component $\nabla_{\hat{x}}D(\hat{x})$ is the gradient of the discriminator's output with respect to its input $\hat{x}$.
- $\mathbf{P}_{\hat{\mathbf{x}}}$ represents the distribution of interpolated samples between real and generated data.
- $\mathbf{k}$ is a target norm for the gradient, often set to 1.

The purpose of introducing a gradient penalty is to enforce a constraint on the gradient of the discriminator output. This constraint ensures that the discriminator does not become overly confident, which can lead to sharp decision boundaries and unstable training. By keeping the gradient close to a target norm k, typically set to 1, the gradient penalty helps maintain a smooth decision boundary.

The hyperparameter λ plays a crucial role in this process. A higher value of λ results in a stronger penalty for deviations from the target gradient norm, while a lower value of λ results in a weaker penalty. Proper tuning of λ is essential for the stability and performance of the GAN.

The interpolated samples $\hat{x}$ are generated by taking convex combinations of real and generated samples. This interpolation helps in evaluating the gradient penalty across the data distribution, promoting better generalization and stability in the training process.

The discriminator loss with gradient penalty can be incorporated as follows:

$$L_D = \mathbb{E}_{x \sim p_{\text{data}}(x)} [\log(D(x))] + \mathbb{E}_{z \sim p_z(z)} [\log(1 - D(G(z)))] + GP \tag{15}$$

This loss function consists of the usual GAN discriminator loss components for real and generated data, plus the gradient penalty term to regularize the discriminator behavior.

The discriminator loss with gradient penalty in GANs integrates standard loss components for real and generated data, augmented by a gradient penalty term to regulate the discriminator behavior. By penalizing large gradients, this method promotes smoother behavior in the discriminator, contributing to a more stable training process for both the generator and the discriminator. The gradient penalty helps prevent sharp changes in the discriminator output, which can otherwise lead to instability and issues like mode collapse, where the generator produces limited and repetitive outputs.

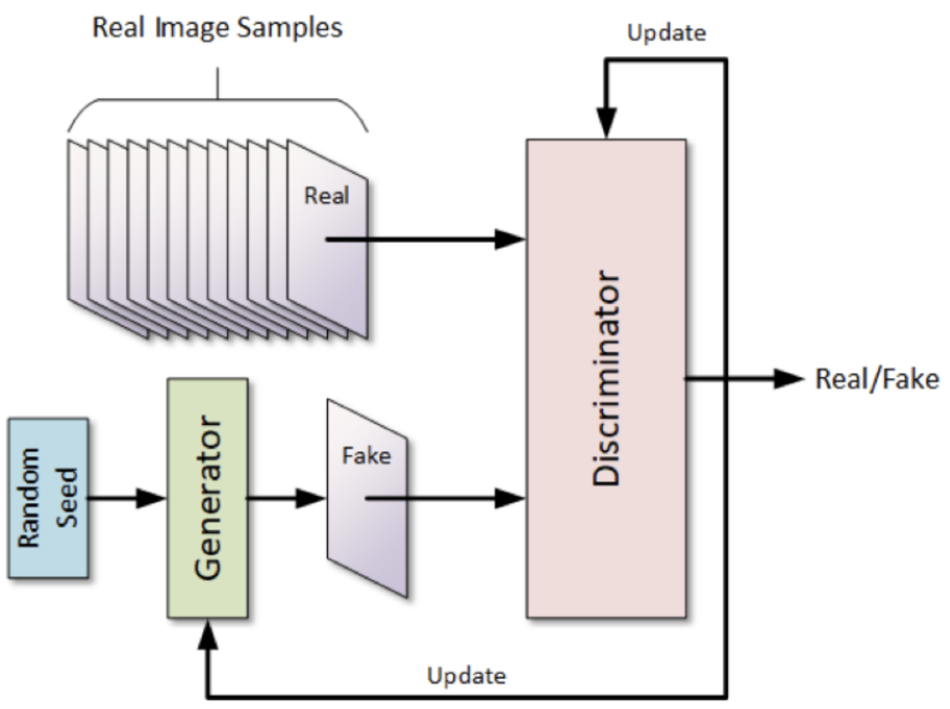


Figure 6: The Generative Adversarial Network Architecture [77].

GAN are not only remarkable for their ability to create high-quality data, but also for their versatility across a range of complex tasks. Their unique adversarial training mechanism ensures that the generated data is continually refined, pushing the boundaries of what is possible in Machine Learning and Artificial Intelligence.

3.3.4 Visual Transformer (ViTs)

Originally designed for Natural Language Processing, the Transformer [2] architecture has been successfully adapted to other domains.

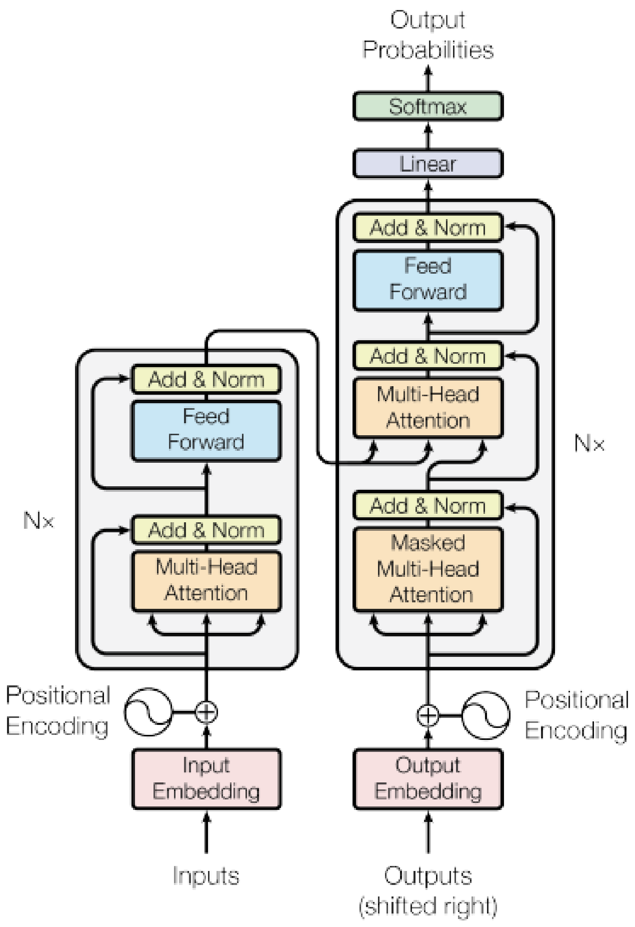


Figure 7: The Transformer Architecture [2].

A notable example is the Vision Transformer (ViT) [3], which adapts the Transformer for Computer Vision tasks. In ViTs, an image is divided into a sequence of patches, each linearly embedded and treated as input tokens for a stack of Transformer layers, collectively referred to as the Transformer Encoder.

The ViT has achieved state-of-the-art performance on various image classification benchmarks, showcasing the Transformer architecture versatility and effectiveness across different domains.

Visual Transformer (ViT) is a Deep Learning architecture that leverages the self-attention mechanism of Transformers to process and understand visual data such as images.

By dividing an input image into smaller patches, ViT allows each patch to attend to others, leading to state-of-the-art performance in various computer vision tasks

The usual workflow of Visual Transformers in Image Classification task include [78]:

1. Decomposing the Image into Patches
   - **Input Image**: Consider an input image with dimensions $H \times W \times N$, where H is the height, W is the width, and N represents the number of channels (such as 3 for RGB images).
   - **Patch Flattening**: The image is divided into smaller patches, each of size 16×16 pixels. For an RGB image, this results in each patch being flattened into a vector of 768 elements (since $16 \times 16 \times 3 = 768$). This transformation converts the spatial information of the image patches into a form suitable for further processing by the Vision Transformer.
2. Patch Embedding
   - **Linear Transformation**: Each flattened patch undergoes a trainable linear transformation that maps it into a D-dimensional embedding space. This operation, which is akin to a fully connected layer, enriches the representation of each patch, making it more suitable for subsequent processing by the transformer.
   - **Spatial Information Encoding**: Since the transformer architecture does not inherently preserve the spatial arrangement of patches, additional spatial information is incorporated into the patch embeddings. This is achieved by adding positional encodings, which can be either fixed or learned parameters, to the embedded patches. These encodings ensure that the model retains information about the position of each patch within the original image.
3. Visual Transformer Encoder
   - **Self-Attention Mechanism**: At the heart of the transformer model lies the self-attention mechanism, which efficiently evaluates the relevance of different patches in relation to each other for a specific task. This mechanism determines the amount of focus each part of the image should receive when encoding a particular patch, enhancing the model ability to understand complex relationships within the image.
   - **Multi-head Attention**: The encoder employs multi-head attention, running several self-attention mechanisms in parallel. This approach allows the model to simultaneously capture diverse aspects and patterns in the image, enriching the representation of each patch with a variety of contextual information.
   - **Feed-forward Network**: Following the attention computation, each patch representation is processed through a feed-forward neural network (FFNN). This network is applied identically and independently to each position, further refining the encoded information and enabling the model to learn intricate features.
   - **Residual Connections and Normalization**: To ensure a stable training process and facilitate the accumulation of knowledge from each layer, the transformer encoder incorporates residual connections around each sub-layer. These are followed

by layer normalization, which helps maintain consistent gradients and improves the overall training dynamics, ensuring that each layer effectively builds upon the previous ones.

4. Creating the Output
   - **CLS Token**: To adapt transformer architecture for image classification, a dedicated classification token (CLS) is prefixed to the sequence of embedded patches. This token encapsulates comprehensive information aggregated from the entire image after undergoing transformation through the encoder. Utilized by a classifier, typically a straightforward linear layer, it facilitates accurate prediction of the image class by leveraging enriched contextual understanding encoded within the transformer model.
5. Training
   - **Optimizing Model Parameters**: During training, the transformer model is fine-tuned end-to-end using an appropriate loss function, commonly cross-entropy for classification tasks. This function calculates the discrepancy between predicted and actual class labels, guiding the optimization of parameters across the linear projection, transformer encoder, and classifier. The objective is to systematically minimize this loss on a training dataset, ensuring the model effectively learns to classify images with high accuracy.

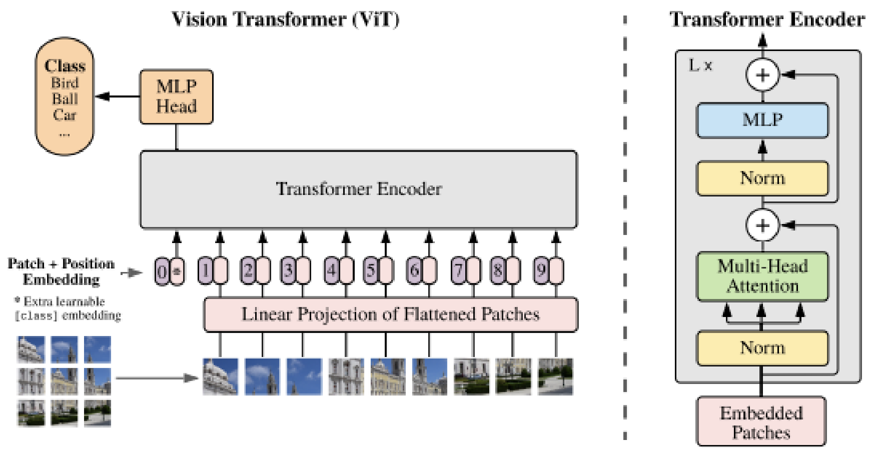


Figure 8: The Visual Transformer Neural Architecture [2, 3].

The transformation at each layer can be defined as [49]:

$$Z = X + \text{LayerNorm}(\text{MHA}(X, X, X)) \tag{16}$$

$$X' = Z + \text{LayerNorm}(\text{MLP}(Z)) \tag{17}$$

where $X \in \mathbb{R}^{N \times D}$ is a Transformer layer matrix input and $X' \in \mathbb{R}^{N \times D}$ is the output.

The attention mechanism is a vital component of the Transformer architecture, enabling the model to focus on specific parts of the input sequence when generating each output element. Given a query matrix $Q \in \mathbb{R}^{N \times D_k}$, a key matrix $K \in \mathbb{R}^{M \times D_k}$, and a value matrix $V \in \mathbb{R}^{M \times D_v}$, the attention function is defined as [49]:

$$\text{Attention}(Q, K, V) = \text{softmax}\left(\frac{QK^T}{\sqrt{D_k}}\right)V \tag{18}$$

where D_k is the dimension of the keys, used as a scaling factor to prevent the dot products from becoming excessively large.

Self-attention is a specialized form of attention where the query, key, and value matrices are all derived from the same input matrix X. Within the Transformer, self-attention permits each position in the input sequence to attend to all other positions in the preceding layer.

Multi-head attention augments the attention mechanism, enabling the model to concurrently attend to information from diverse representation subspaces at different positions. Rather than executing a single attention function, multi-head attention projects the queries, keys, and values h times using different learned linear projections. These projections are processed in parallel, their results concatenated, and the concatenated output is then linearly projected once more. Mathematically, multi-head attention is expressed as [49]:

$$\text{MHA}(Q, K, V) = \text{Concat}(\text{head}_1, \ldots, \text{head}_h)W^O \tag{19}$$

$$\text{head}_i = \text{Attention}(QW_i^Q, KW_i^K, VW_i^V) \tag{20}$$

Here, $W_i^Q \in \mathbb{R}^{D_x \times D_k}$, $W_i^K \in \mathbb{R}^{D_x \times D_k}$, $W_i^V \in \mathbb{R}^{D_x \times D_v}$, and $W^O \in \mathbb{R}^{hD_v \times D_x}$ are learnable parameter matrices.

3.3.5 Learning Rate Schedulers

When training deep networks, it is beneficial to gradually reduce the learning rate as the training progresses. This concept stems from the idea that a high learning rate imparts high kinetic energy to the model, causing its parameter vector to oscillate chaotically. Consequently, the model struggles to stabilize within the deeper and narrower valleys of the loss function, which represent local minima. Conversely, if the learning rate is too low, the system exhibits low kinetic energy, leading it to settle into the shallower and narrower valleys of the loss function, often getting stuck in suboptimal minima.

In essence, starting with a higher learning rate helps the model explore the loss landscape broadly, avoiding premature convergence. As training advances, lowering the learning rate allows the model to fine-tune its parameters and settle into more optimal solutions, thus achieving better performance. This dynamic adjustment helps in navigating the complex loss surfaces typical in deep learning, striking a balance between exploration and exploitation [79].

Examples of Learning Rate Scheduling [80]:

- **ExponentialLR**: Adjusts the learning rate by dividing it by a factor each epoch or evaluation period.
- **CyclicLR**: Oscillates the learning rate cyclically between specified minimum and maximum values.
- **StepLR**: Decreases the learning rate by multiplying it with a decay factor every fixed number of epochs or evaluation periods.
- **MultiStepLR**: Reduces the learning rate by a factor when the training reaches predefined milestones.
- **ReduceLROnPlateau**: Dynamically reduces the learning rate if a monitored metric fails to improve for a certain number of evaluations.
- **CosineAnnealingLR**: Starts with a high learning rate and smoothly decreases it following a cosine curve, periodically restarting the cycle.

3.3.6 Weight Decay

Weight decay, also known as L2 regularization, is an essential technique in deep learning to enhance model performance. By penalizing large weights, it offers several key benefits [81]:

- **Mitigates Overfitting:** Large weights often cause a model to memorize training data, hindering its ability to generalize to new examples. Weight decay discourages large weights, promoting the learning of smaller, more generalizable weights that capture the fundamental patterns in the data.
- **Enhances Model Stability:** Large weights can destabilize training and increase sensitivity to data noise. Weight decay stabilizes the training process by keeping weights manageable, thus enhancing the model robustness and reducing overfitting.
- **Encourages Feature Sharing:** By promoting similar weights across different neurons, weight decay fosters feature sharing. This leads to a more efficient network where multiple neurons can utilize the same features, often resulting in fewer required parameters.
- **Boosts Generalization in Overparameterized Models:** Modern deep learning models often contain more parameters than training data, a scenario known as overparameterization. Weight decay helps manage the complexity of such models, improving their ability to generalize.

Weight decay adds a penalty to the loss function, proportional to the sum of the squared weights. This penalty encourages the model to prefer smaller weights during training. Implementation of weight decay typically follows one of two approaches:

- **L2 Regularization:** Adds a term to the loss function proportional to the sum of the squared weights.
- **Optimizer-based Weight Decay:** Alters the optimizer update rule to include a decay factor that incrementally reduces the weights.

Choosing the right weight decay parameter involves balancing overfitting prevention with performance optimization. The optimal value varies based on model size, complexity, training data, and learning rate. Techniques such as grid search, hyperparameter optimization, and cross-validation are useful for identifying the best weight decay value.

3.3.7 Cross Validation

Cross-Validation (CV) is a cornerstone of evaluating learning models which provides the possibility to approximate model performance on unseen data not used while training. When training a model, the dataset is divided into two primary subsets: training and testing. The training set encompasses all the examples from which the model learns, while the testing set emulates real-world scenarios where the model performance is assessed [82].

Validating a learning model before deploying it in production involves ensuring it can make accurate predictions on data it has never encountered. This unseen data represents any type of information that the model hasn't been trained on. Ideally, during testing, this data flows directly into the model across numerous iterations. However, in practice, access to genuinely new data is often restricted or unavailable in a fresh environment.

A common approach is the 80-20 rule, where 80% of the data is used for training and 20% for testing. Despite its popularity, this method is prone to creating a seemingly perfect split that might artificially inflate model accuracy, while failing to replicate the same performance in real-world scenarios. The accuracy achieved in such cases can often be attributed to mere chance. It is worth noting that the 80-20 split is not a hard and fast rule; other ratios like 70-30 or 75-25 are frequently used as well.

Tuning model hyperparameters is another crucial step in optimizing an algorithm to uncover the hidden patterns within a dataset. However, performing this tuning on a simple training-testing split is generally discouraged. Model performance is highly sensitive to hyperparameters, and adjusting them based on a fixed data split can lead to overfitting, reducing the model generalizability.

To avoid overfitting during hyperparameter tuning, some suggest dividing the dataset into three parts: training, validation, and testing, for example, using 70% for training, 20% for validation, and 10% for testing. Yet, with small datasets, maximizing the data for training is often necessary. Moreover, such splits can introduce bias if important examples are disproportionately allocated to the training or validation sets. Cross-validation (CV) is a valuable technique to mitigate these issues by ensuring a more balanced evaluation.

The process of splitting data for training and testing can be fraught with difficulties. The testing set may not share similar properties with the training set, causing instability. Even though random sampling theoretically gives each sample an equal chance of being included in the testing set, a single split might still result in instability when the experiment is repeated with a different division.

3.3.8 Random Search

Random Search is a hyperparameter optimization technique in machine learning that samples a defined number of hyperparameter combinations from specified distributions at random. These distributions represent sets or ranges of possible parameter values [83].

This method efficiently navigates a subset of the hyperparameter space, making it faster and more resource-efficient compared to exhaustive techniques like grid search.

Although Random Search may not guarantee the discovery of the absolute optimal hyperparameters, it is highly effective at identifying good configurations, particularly in scenarios involving large or complex parameter spaces.

The Random Search process begins by initializing random hyperparameter values from the defined search space, denoted as point x. The value of the cost function is then evaluated at x. Subsequently, another set of random hyperparameter values, labeled y, is generated, and the cost function is computed at this new point y.

If the cost function value at y is lower than that at x, the algorithm updates the starting point to y by setting $x = y$. This iterative process continues: generating new random points, comparing cost function values, and updating the starting point if a better set of hyperparameters is found. The procedure repeats until a stopping criterion specified by the user is met [84].

Some advantages of random search in Machine Learning over grid search are:

- Random search tends to yield better results than grid search, especially when the dimensionality and the number of hyperparameters are high.
- Random search allows for limiting the number of hyperparameter combinations, whereas grid search exhaustively evaluates all possible combinations.
- Typically, random search achieves superior results compared to grid search in fewer iterations.

3.4 Ordinary Differential Equations (ODE)

Differential equations are pivotal in capturing and describing the dynamics of various physical, biological, and engineering systems. Among these, Ordinary Differential Equations (ODEs) hold a special place. ODEs are equations involving functions of a single independent variable and their derivatives. They are termed ordinary to distinguish them from partial differential equations, which involve multiple independent variables.

An ODE is formulated as an equation involving an unknown function $f(x)$, its derivatives, and the independent variable x. It is being assumed that $f(x)$ is differentiable up to order k. The general form of an ODE of order k is expressed as [85]:

$$G\left(x, f(x), \frac{df}{dx}, \dots, \frac{d^k f}{dx^k}\right) = 0, \tag{21}$$

where the function G includes the highest derivative $\frac{d^k f}{dx^k}$. The order of the ODE is determined by the highest derivative present. The degree of the ODE is the power of the highest derivative after clearing any fractional powers.

An ODE is called linear if G is a linear function of $f(x)$ and its derivatives:

$$a_k(x)\frac{d^k f}{dx^k} + a_{k-1}(x)\frac{d^{k-1} f}{dx^{k-1}} + \cdots + a_1(x)\frac{df}{dx} + a_0(x)f = h(x). \tag{22}$$

Otherwise, it is nonlinear.

The general form provided is often referred to as the implicit form. In an explicit form, the highest-order derivative is expressed as a function of lower-order derivatives:

$$\frac{d^k f}{dx^k} = F\left(x, f(x), \frac{df}{dx}, \ldots, \frac{d^{k-1} f}{dx^{k-1}}\right). \tag{23}$$

To uniquely determine a solution to an ODE, initial conditions are necessary. These conditions specify the value of the function and its derivatives at a particular point. For a first-order ODE, an initial condition might look like:

$$f(x_0) = f_0, \tag{24}$$

where x_0 is the initial point, and f_0 is the value of the function at x_0.

For a second-order ODE, two initial conditions are typically required:

$$f(x_0) = f_0 \quad \text{and} \quad \left.\frac{df}{dx}\right|_{x=x_0} = f_1. \tag{25}$$

Let us consider a specific example of an ODE for the function $f(x)$:

$$\frac{d^2 f}{dx^2} = 5\left[1 + \left(\frac{df}{dx}\right)^{\frac{2}{3}}\right]^{\frac{1}{3}}. \tag{26}$$

This equation is a nonlinear explicit ODE of degree 3 and order 2. The nonlinearity arises from the fractional power of the first derivative term.

Solving an ODE involves finding the function $f(x)$ that satisfies the equation over the domain of x (e.g., $\mathbb{R}$). Various methods can be employed depending on the nature of the ODE, including analytical techniques, such as separation of variables, integrating factors, and numerical approaches, such as Euler's method, Runge-Kutta methods.

ODEs are essential tools in modeling and understanding the behavior of dynamic systems. Their solutions provide insight into how systems evolve over time under various conditions.

3.5 Runge-Kutta Methods

Runge-Kutta methods are a family of iterative methods used to solve initial value problems for ordinary differential equations (ODEs). These methods vary in their order of accuracy, with higher-order methods providing more accurate solutions. Here,the formulations for the 1st, 2nd, 3rd, and 4th order Runge-Kutta methods are being presented [85].

The simplest Runge-Kutta method is the 1st order method, also known as Euler's Method. It is given by:

$$y_{i+1} = y_i + hf(y_i, x_i). \tag{27}$$

This method has a local truncation error of $O(h^2)$ and a global truncation error of $O(h)$.

The 2nd order Runge-Kutta method improves upon Euler's method by incorporating an additional intermediate step. One common form is the Heun's method, given by:

$$k_{1i} = f(y_i, x_i), \tag{28}$$

$$k_{2i} = f(y_i + hk_{1i}, x_i + h), \tag{29}$$

$$y_{i+1} = y_i + \frac{h}{2}(k_{1i} + k_{2i}). \tag{30}$$

Another popular 2nd order method is the midpoint method:

$$k_{1i} = f(y_i, x_i), \tag{31}$$

$$k_{2i} = f(y_i + \frac{h}{2}k_{1i}, x_i + \frac{h}{2}), \tag{32}$$

$$y_{i+1} = y_i + hk_{2i}. \tag{33}$$

Both methods have a local truncation error of $O(h^3)$ and a global truncation error of $O(h^2)$.

The 3rd order Runge-Kutta method provides even greater accuracy. It is formulated as follows:

$$k_{1i} = f(y_i, x_i), \tag{34}$$

$$k_{2i} = f\left(y_i + \frac{h}{2}k_{1i}, x_i + \frac{h}{2}\right), \tag{35}$$

$$k_{3i} = f\left(y_i - hk_{1i} + 2hk_{2i}, x_i + h\right), \tag{36}$$

$$y_{i+1} = y_i + \frac{h}{6}(k_{1i} + 4k_{2i} + k_{3i}). \tag{37}$$

This method has a local truncation error of $O(h^4)$ and a global truncation error of $O(h^3)$.

The 4th order Runge-Kutta method is the most widely used due to its high accuracy and relatively simple implementation. It is given by:

$$k_{1i} = f(y_i, x_i), \tag{38}$$

$$k_{2i} = f\left(y_i + \frac{h}{2}k_{1i}, x_i + \frac{h}{2}\right), \tag{39}$$

$$k_{3i} = f\left(y_i + \frac{h}{2}k_{2i}, x_i + \frac{h}{2}\right), \tag{40}$$

$$k_{4i} = f(y_i + hk_{3i}, x_i + h), \tag{41}$$

$$y_{i+1} = y_i + \frac{h}{6}(k_{1i} + 2k_{2i} + 2k_{3i} + k_{4i}). \tag{42}$$

This method has a local truncation error of $O(h^5)$ and a global truncation error of $O(h^4)$.

The Runge-Kutta methods of various orders offer a range of options for solving initial value problems in ODEs, with higher-order methods providing greater accuracy. The choice of method depends on the desired balance between computational complexity and accuracy.

3.5.1 ODE Interpretation of Transformer Layers

Residual networks can be viewed as an Euler discretization of the solutions to Ordinary Differential Equations (ODEs). Building on this concept, delving deeper into the relationship between Transformers and numerical ODE methods it can be demonstrated that a residual block of layers within a Transformer can be interpreted as a higher-order solution to an ODE [50].

Let $Y^m := [y_1^m, y_2^m, \dots, y_L^m]$ represent the input to the m-th transformer layer, where L denotes the sequence length. The multi-head attention sublayer can be mathematically expressed as [47]:

$$\hat{y}_i^m = y_i^m + G(y_i^m, Y^m), \quad 1 \leq i \leq L, \tag{43}$$

where G represents the multi-head dot-product attention function.

The output from this sublayer, denoted by $\hat{Y}^m := [\hat{y}_1^m, \hat{y}_2^m, \ldots, \hat{y}_L^m]$, is then fed into the feedforward network sublayer. The transformation in this sublayer is given by:

$$y_i^{m+1} = \hat{y}_i^m + H(\hat{y}_i^m), \quad 1 \leq i \leq L, \tag{44}$$

where H represents the sequence of linear mappings and activation functions.

The operations in equations 43 and 44 can be viewed as a numerical integration scheme over the time interval $[m, m+1]$ of the following ODE, using the Lie-Trotter splitting method. [86, 87]:

$$\frac{dy_i}{dt} = H(y_i) + G(y_i, Y). \tag{45}$$

This interpretation provides a deeper understanding of the Transformer internal mechanisms by linking them to the well-established principles of numerical integration.

Writing $H(y_i)$ and $G(y_i, Y)$ in terms of a new function, namely $F(y_i, Y) = H(y_i) + G(y_i, Y)$, and keeping in mind that the function $F(y_i, Y)$ is re-used in a block and the input Y is shared within the block, Runge-Kutta blocks can be computed as follows:

$$y_i(t+1) = y_i(t) + \sum_{j=1}^{n} \gamma_j F_{ij} \tag{46}$$

$$F(y_i, Y) = F_i \tag{47}$$

$$F_{ij} = F_i(y_i + \sum_{p=1}^{j-1} \beta_{jp} F_{ip}, Y) \tag{48}$$

Where β is a coefficient which can be determined by the Taylor series of $y_i(t+1)$ This makes the neural network more parameter-efficient [50].

Thus, the definiton of Euler's method is (RK1):

$$y_i(t+1) = y_i(t) + F_{i1} \tag{49}$$

$$F_{i1} = F_i(y_i(t), Y) \tag{50}$$

One can define a second order Runge-Kutta block as (RK2):

$$y_i(t+1) = y_i(t) + \frac{1}{2}(F_{i1} + F_{i2}) \tag{51}$$

$$F_{i1} = F_i(y_i(t), Y) \tag{52}$$

$$F_{i2} = F_i(y_i(t) + F_{i1}, Y) = F_i(y_i(t) + F_i(y_i(t), Y), Y) \tag{53}$$

And a third order Runge-Kutta block as (RK3):

$$y_i(t+1) = y_i(t) + \frac{1}{9}(2F_{i1} + 3F_{i2} + 4F_{i3}) \tag{54}$$

$$F_{i1} = F_i(y_i(t), Y) \tag{55}$$

$$F_{i2} = F_i(y_i(t) + \frac{1}{2}F_{i1}, Y) = F_i(y_i(t) + \frac{1}{2}F_i(y_i(t), Y), Y) \tag{56}$$

$$F_{i3} = F_i(y_i(t) + \frac{3}{4}F_{i2}, Y) = F_i(y_i(t) + \frac{3}{4}F_i(y_i(t) + \frac{1}{2}F_i(y_i(t), Y), Y), Y) \tag{57}$$

Likewise, a fourth order Runge-Kutta block can be written as (RK4):

$$y_i(t+1) = y_i(t) + \frac{1}{6}(F_{i1} + 2F_{i2} + 2F_{i3} + F_{i4}) \tag{58}$$

$$F_{i1} = F_i(y_i(t), Y) \tag{59}$$

$$F_{i2} = F_i(y_i(t) + \frac{1}{2}F_{i1}, Y) = F_i(y_i(t) + \frac{1}{2}F_i(y_i(t), Y), Y) \tag{60}$$

$$F_{i3} = F_i(y_i(t) + \frac{1}{2}F_{i2}, Y) = F_i(y_i(t) + \frac{1}{2}F_i(y_i(t) + \frac{1}{2}F_i(y_i(t), Y), Y), Y) \tag{61}$$

$$F_{i4} = F_i(y_i(t) + F_{i3}, Y) = F_i(y_i(t) + F_i(y_i(t) + \frac{1}{2}F_i(y_i(t) + \frac{1}{2}F_i(y_i(t), Y), Y), Y), Y) \tag{62}$$

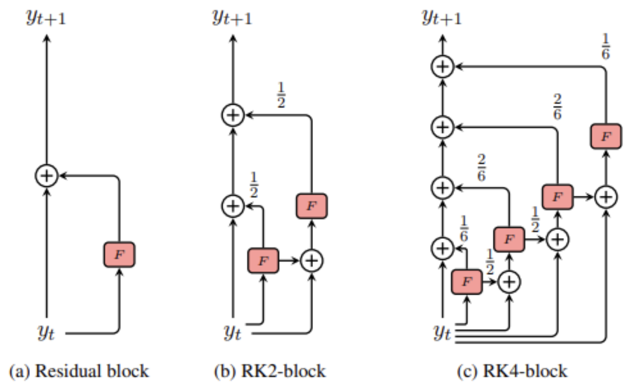


Figure 9: Different Runge-Kutta ODEs Blocks [50].

However, further optimizations can still be made for deeper models in terms of minimizing the vanishing gradient effects. Considering RK4:

$$y_i(t+1) = y_i(t) + \frac{1}{6}(F_{i1} + 2F_{i2} + 2F_{i3} + F_{i4}) \tag{63}$$

Let $\mathcal{E}$ be the loss of training, L be the number blocks of the model, and y_{iL} be the model output. The gradient of $\mathcal{E}$ at y_{it}, using the Chain Rule, is:

$$\frac{\partial \mathcal{E}}{\partial y_{it}} = \frac{\partial \mathcal{E}}{\partial y_{iL}} \cdot \frac{\partial y_{iL}}{\partial y_{iL-1}} \cdot \frac{\partial y_{iL-1}}{\partial y_{iL-2}} \cdots \frac{\partial y_{it+1}}{\partial y_{it}} \tag{64}$$

$$\frac{\partial \mathcal{E}}{\partial y_{it}} = \frac{\partial \mathcal{E}}{\partial y_{iL}} \cdot \prod_{k=t}^{L-1} \frac{\partial y_{ik+1}}{\partial y_{ik}} \tag{65}$$

where

$$\begin{aligned}\frac{\partial y_{ik+1}}{\partial y_{ik}} = 1 + \frac{1}{6}\frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}} + \\ \frac{1}{3}\frac{\partial F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y)}{\partial y_{ik} + \frac{1}{2}F_i(y_{ik}, Y)} \cdot \left(1 + \frac{1}{2}\frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}}\right) + \\ \frac{1}{3}\frac{\partial F_i(y_{ik} + \frac{1}{2}F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y), Y)}{\partial y_{ik} + \frac{1}{2}F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y)} \cdot \\ \left[1 + \frac{1}{2}\frac{\partial F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y)}{\partial y_{ik} + \frac{1}{2}F_i(y_{ik}, Y)} \cdot \left(1 + \frac{1}{2}\frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}}\right)\right] + \\ \frac{1}{6}\frac{\partial F_i(y_{ik} + F_i(y_{ik} + \frac{1}{2}F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y), Y), Y)}{\partial y_{ik} + F_i(y_{ik} + \frac{1}{2}F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y), Y)} \cdot \\ \left\{1 + \frac{\partial F_i(y_{ik} + \frac{1}{2}F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y), Y)}{\partial y_{ik} + \frac{1}{2}F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y)} \cdot \right. \\ \left.\left[1 + \frac{1}{2}\frac{\partial F_i(y_{ik} + \frac{1}{2}F_i(y_{ik}, Y), Y)}{\partial y_{ik} + \frac{1}{2}F_i(y_{ik}, Y)} \cdot \left(1 + \frac{1}{2}\frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}}\right)\right]\right\}\end{aligned} \tag{66}$$

This result can lead to a higher risk of gradient vanishing when L is big.

The issue seems to stem from the small coefficients of F_i, specifically, $\gamma_1 = \gamma_4 = \frac{1}{6}$ and $\gamma_2 = \gamma_3 = \frac{1}{3}$. A practical approach is to empirically set $\gamma_i = 1$, $\beta_p = 1$ for removing the product factors that are less than 1 during gradient computation. Furthermore, experiments shown that eliminating F_{i2} and Fi_3 blocks can lead to a faster convergence and a small accuracy increase.

However, this adjustment is not theoretically supported by standard Runge-Kutta methods. Eq. 63 with the updated coefficients can be written as:

$$y_i(t+1) = y_i(t) + F_{i1} + F_{i4} \tag{67}$$

Then, with all of the transformations, the gradient of loss will become:

$$\frac{\partial \mathcal{E}}{\partial y_{it}} = \frac{\partial \mathcal{E}}{\partial y_{iL}} \cdot \prod_{k=t}^{L-1} f_{ik} \tag{68}$$

where

$$\begin{aligned}f_{ik} = 1 + \frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}} + \frac{\partial F_i(y_{ik} + F_i(y_{ik}, Y), Y)}{\partial y_{ik} + F_i(y_{ik}, Y)} \cdot \left(1 + \frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}}\right) + \\ \frac{\partial F_i(y_{ik} + F_i(y_{ik} + F_i(y_{ik}, Y), Y), Y)}{\partial y_{ik} + F_i(y_{ik} + F_i(y_{ik}, Y), Y)} \cdot \\ \left[1 + \frac{\partial F_i(y_{ik} + F_i(y_{ik}, Y), Y)}{\partial y_{ik} + F_i(y_{ik}, Y)} \cdot \left(1 + \frac{\partial F_i(y_{ik}, Y)}{\partial y_{ik}}\right)\right]\end{aligned} \tag{69}$$

Understanding the transformer layer operations through the lens of numerical integration opens up new avenues for improving their efficiency and effectiveness by leveraging advanced techniques from numerical analysis.

3.6 Quantum Computing

Quantum computing revolutionizes information processing by using qubits instead of classical bits. Unlike classical bits, qubits can exist simultaneously in multiple states thanks to superposition. The state of n qubits is described by a unit vector $|\psi\rangle$ in the Hilbert space $\mathbb{C}^{2^n}$. Here, the ket $|\cdot\rangle$ denotes a column vector and the bra $\langle\cdot|$ denotes a row vector.

An n-level quantum system is described by an n-dimensional Hilbert space $\mathcal{H}^n$. A quantum pure state is represented by a unit complex vector $\mathbf{v}$ in $\mathcal{H}^n$. This state can be depicted as a density matrix $\rho = \mathbf{v}\mathbf{v}^\dagger$, where $\mathbf{v}^\dagger$ is the conjugate transpose of $\mathbf{v}$. For a mixture of pure states $\{\mathbf{v}_i\}_{i=1}^n$ with probabilities $\{p_i\}_{i=1}^n$ summing to 1, the mixed state density matrix is $\rho = \sum_{i=1}^n p_i \mathbf{v}_i \mathbf{v}_i^\dagger$. [7].

The evolution of a closed quantum system is governed by unitary transformations [19]. The state $\mathbf{v}_t$ at time t evolves to $\mathbf{v}_{t+\Delta t}$ at time $t + \Delta t$ via a unitary matrix $U \in \mathcal{H}$, such that $U_{\Delta t}\mathbf{v}_t = \mathbf{v}_{t+\Delta t}$. This matrix U satisfies $UU^\dagger = I$. Considering a density matrix ρ, the system evolution is expressed as $U\rho_t U^\dagger = \rho_{t+\Delta t}$.

A quantum circuit manipulates qubits through a series of quantum gates, transforming their states via unitary operations. Mathematically, this is represented as $U|\psi\rangle$, where U is a $2^n \times 2^n$ unitary matrix. The circuit typically concludes with the measurement of qubits, revealing the final state of the system.

In this context, R_X, R_Y and R_Z gates are used, which rotates a qubit around the X, Y and Z axes, and CNOT gates, which flips the state of a target qubit conditionally on the state of a control qubit being $|1\rangle$. The Hadamard gate H helps with the creation of entanglement, along with CNOT.

These gates are defined by the following matrices:

$$H = \frac{1}{\sqrt{2}} \begin{bmatrix} 1 & 1 \\ 1 & -1 \end{bmatrix}$$

$$R_X(\theta) = \begin{bmatrix} \cos(\theta/2) & -i\sin(\theta/2) \\ -i\sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

$$R_Y(\theta) = \begin{bmatrix} \cos(\theta/2) & -\sin(\theta/2) \\ \sin(\theta/2) & \cos(\theta/2) \end{bmatrix}$$

$$R_Z(\theta) = \begin{bmatrix} e^{-i\theta/2} & 0 \\ 0 & e^{i\theta/2} \end{bmatrix}$$

$$\text{CNOT} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 1 & 0 \end{bmatrix}$$

3.6.1 Quantum Machine Learning

Quantum Machine Learning (QML) leverages Quantum Computing to enhance classical Machine Learning models. By integrating quantum circuits into classical algorithms, QML exploits unique properties in Quantum Mechanics to improve performance and capabilities.

Notable QML models include Quantum Support Vector Machines [88], Quantum Nearest-Neighbor Algorithms [89], Quantum Nearest Centroid Classifiers [90], and Quantum Artificial Neural Networks [91], as well as Quantum Graph Neural Networks [92]. These models often use Variational Quantum Algorithms [10], or Variational Quantum Circuits (VQCs) [13], where rotation angles are optimized alongside classical parameters [49].

Variational Quantum Algorithms (VQAs) have emerged to address various challenges in quantum chemistry and combinatorial optimization. Key examples include the Variational Quantum Eigensolver (VQE) and the Quantum Approximate Optimization Algorithm (QAOA). VQAs operate by optimizing parameters within a quantum circuit to minimize a given objective function, often represented by the Hamiltonian of the system [93].

The structure of a VQA can be distilled into four key components:

1. **Objective Function:** Often derived from the Hamiltonian, guiding the evolution of a quantum state. The objective function $C(\theta)$ is typically given by:

   $$C(\theta) = \langle \psi(\theta) | \hat{H} | \psi(\theta) \rangle$$

   where $|\psi(\theta)\rangle$ is the quantum state prepared by the parameterized quantum circuit and $\hat{H}$ is the Hamiltonian of the system.
2. **Parameterized Quantum Circuit (PQC):** A quantum circuit with tunable parameters, evolving the initial state $|0\rangle$ through a sequence of quantum gates:

   $$|\psi(\theta)\rangle = U(\theta)|0\rangle$$

   where $U(\theta)$ is the unitary operator representing the quantum circuit.
3. **Measurement Scheme:** Extracting information from the quantum state is crucial for evaluating the objective function. This often involves measuring the expectation value of the Hamiltonian:

   $$\langle \hat{H} \rangle = \sum_i h_i \langle \psi(\theta) | \hat{O}_i | \psi(\theta) \rangle$$

   where $\hat{O}_i$ are observables and h_i are coefficients.
4. **Classical Optimizer:** A classical algorithm (e.g., gradient descent) is used to adjust the parameters of the PQC to minimize the objective function:

   $$\theta^{(t+1)} = \theta^{(t)} - \eta \nabla_\theta C(\theta)$$

   where η is the learning rate.

The NISQ era, a term coined by John Preskill, describes the current phase of Quantum Computing characterized by devices with a limited number of qubits and significant noise levels.

These devices do not yet support full-scale quantum error correction, limiting the depth and complexity of executable quantum circuits. NISQ algorithms often adopt a hybrid approach, combining quantum circuits for quantum-intensive computations with classical processes for the remainder.

Quantum neural networks aim to merge Quantum Mechanics with neural network architectures. These networks leverage quantum phenomena such as superposition and entanglement to perform computations that may be infeasible for classical neural networks. The structure of a Quantum Neural Network can be represented as:

- **Quantum Layers:** Each layer consists of quantum gates applied to qubits:

  $$|\psi_l\rangle = U_l(\theta_l)|\psi_{l-1}\rangle$$

  where $U_l(\theta_l)$ is the unitary transformation for layer l.
- **Activation Function:** Measurement and classical post-processing serve as activation functions:

  $$\sigma(x) = \text{Measurement}(U(\theta)x)$$
- **Training:** Training involves optimizing the parameters θ using a classical optimizer:

  $$\theta \to \theta - \eta \nabla_\theta \mathcal{L}$$

  where $\mathcal{L}$ is the loss function.

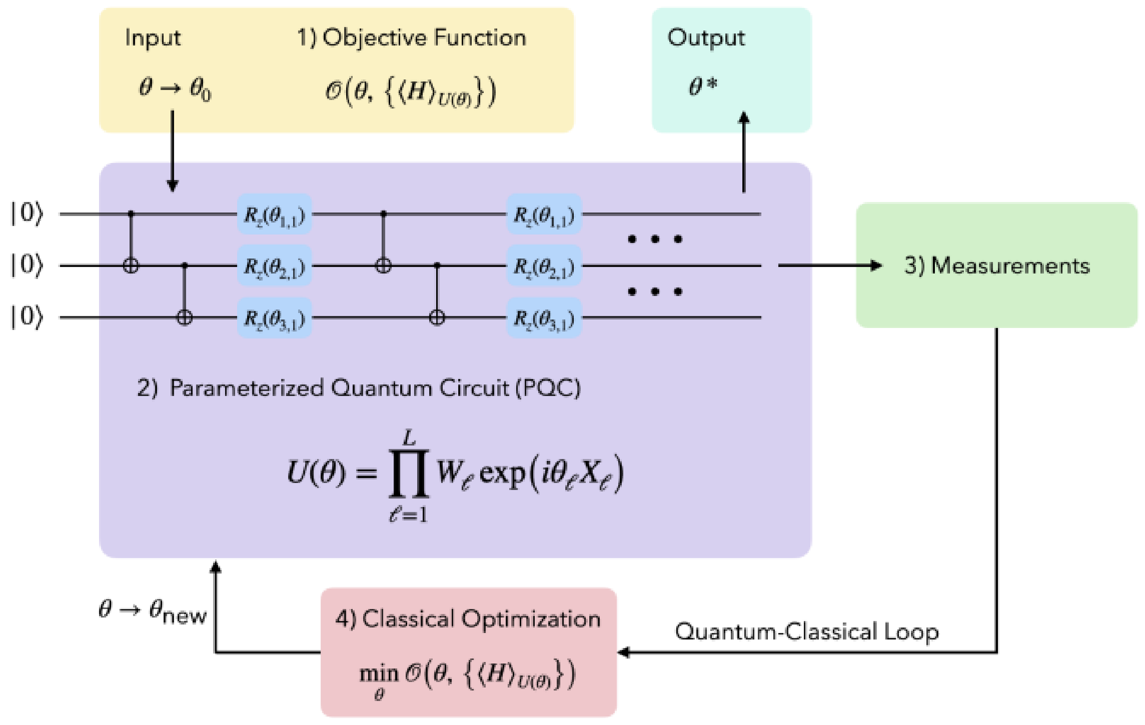

Input
$\theta \to \theta_0$
1) Objective Function
$\mathcal{O}(\theta, \{\langle H\rangle_{U(\theta)}\})$
Output
$\theta*$
$|0\rangle$
$|0\rangle$
$|0\rangle$
$R_z(\theta_{1,1})$
$R_z(\theta_{2,1})$
$R_z(\theta_{3,1})$
$R_z(\theta_{1,1})$
$R_z(\theta_{2,1})$
$R_z(\theta_{3,1})$
3) Measurements
2) Parameterized Quantum Circuit (PQC)
$U(\theta) = \prod_{\ell=1}^{L} W_\ell \exp(i\theta_\ell X_\ell)$
$\theta \to \theta_{\text{new}}$
4) Classical Optimization
$\min_\theta \mathcal{O}(\theta, \{\langle H\rangle_{U(\theta)}\})$
Quantum-Classical Loop

Figure 10: The Variational Quantum Algorithm Structure [93].

4 Proposed Solution

In this chapter, the feasibility and potential of integrating quantum principles into Deep Learning Optimization are being presented. Specifically, neural hybrid architectures based on ODE (Ordinary Differential Equation)-inspired Quantum Visual Transformer (QViTs) network configurations are examined. Additionally, a generative Quantum Neural Network is designed, QRKT-GAN, leveraging qubits in varying quantities, from small to large. The primary aim of this thesis is not to tailor specific architectural components for particular problems in Computer Vision (CV) or Natural Language Processing (NLP). Instead, it focuses on experimenting with hybrid frameworks using widely recognized, consistent, and versatile datasets, complemented by robust classical optimization techniques.

To achieve this, the proposed ODE-QViTs networks are hybrid models that blend classical and Quantum Computing principles. These configurations aim to demonstrate how Quantum Mechanics can enhance the efficiency and performance of Deep Learning models. QRKT-GAN, the proposed generative Quantum Neural Network, leverages a Quantum Runge-Kutta method of 4th order within its architecture, aiming to explore the potential of Variational Quantum Circuits (VQC) in the context of ODEs.

The implementation leverages a variety of advanced technologies:

- **JAX [94] and Flax [95]**: These are used for Just-In-Time (JIT) compilation and advanced numerical computing. They facilitate efficient computation with Deep Learning layers.
- **Tensorflow Quantum [96]**: This tool provide a platform for building and training quantum models.
- **Qiskit [97]**: This is used for simulating and running quantum algorithms.
- **Pytorch [98]**: A popular deep learning framework that supports the implementation and training of complex neural networks.
- **Tensor Circuit [17]**: It is used for efficiently simulating Variational Quantum Circuits in a non-quantum environment leveraging Noisy Intermediate-Scale Quantum (NISQ) era.

Training the networks relies on strong classical techniques to ensure robustness and reliability in performance.

For the implementation of Quantum Visual Transformers Encoders and Unmasked Decoders, several key parameters are being considered [49]:

- **Patch Size**: Determines the size of the input data chunks.
- **Hidden Size**: Defines the dimensionality of the hidden layers within the transformer.
- **ODE-Transformer Blocks**: The design of these blocks is inspired by [50], integrating ODE principles into the transformer architecture.
- **Quantum Attention Heads**: These replace classical attention heads to leverage quantum parallelism.
- **Hidden QMLP Size**: Specifies the size of the hidden layers in the Quantum Multi-Layer Perceptron (QMLP).

These parameters are crucial for optimizing the QViTs architecture and ensuring it can effectively process and learn from the input data.

The QRKT-GAN generative hybrid network incorporates the following components:

- **Generator and Discriminator Architectures**: Both are based on QViTs. The generator architecture specifically includes a Quantum ODE lock, namely a Runge-Kutta 4th order one. This design is intended to investigate the utilization of VQCs in the context of ODEs.

The proposed model is shown below:

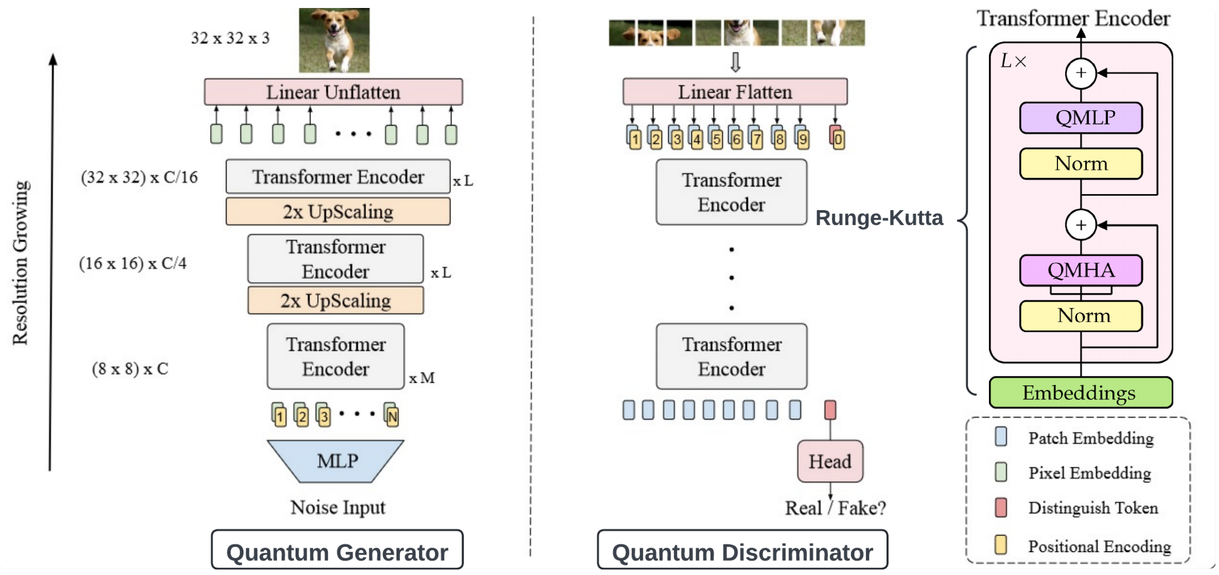
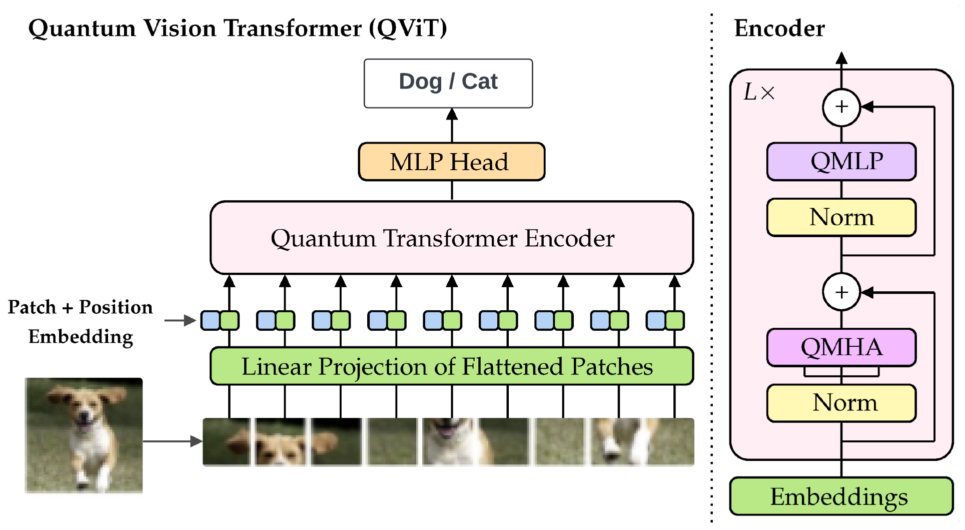


Figure 11: The QRKT-GAN Neural Architecture Structure. Image inspired from [49, 53].

The used Quantum Visual Transformer Neural Architecture is described as:



Quantum Vision Transformer (QViT)
Dog / Cat
MLP Head
Quantum Transformer Encoder
Patch + Position Embedding
Linear Projection of Flattened Patches
Encoder
L×
QMLP
Norm
QMHA
Norm
Embeddings

Figure 12: The Visual Quantum Transformer used in QRKT-GAN. Image inspired from [49].

And the VQC used for implementing the QMLP and Quantum Attention Heads is defined as:

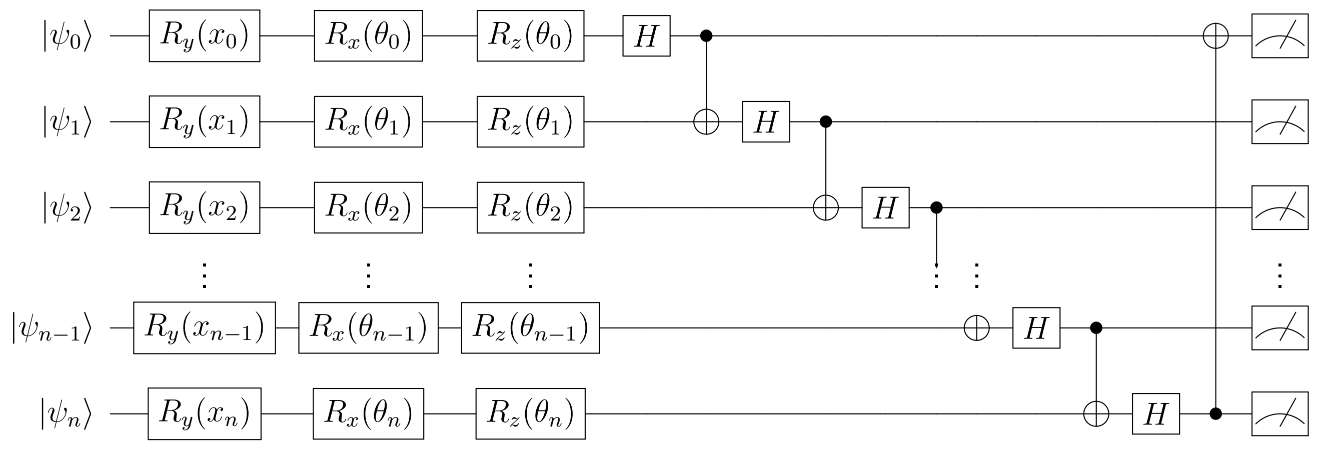


Figure 13: The Variational Quantum Circuit used in QRKT-GAN

Each component of the feature vector $\mathbf{x} = (x_0, \ldots, x_{n-1}, x_n)$ is embedded into the quantum state of qubits by translating them into specific rotation angles. Following this, two layers of single-qubit rotations parameterized by $\boldsymbol{\theta} = (\theta_0, \ldots, \theta_{n-1}, \theta_n)$ is applied to each qubit. These parameters, namely the qurons, are not static; they are optimized and learned alongside other model parameters during the training process.

In contrast with the Variational Quantum Circuit (VQC) from [49], the modified VQC presented here achieves a modest increase in accuracy, typically around 1-2%. This improvement is attributed to the inclusion of additional R_y, R_z and H gates. These gates are believed to enhance the circuit ability to capture information from the input features more effectively, allowing for a richer extraction of semantics from the patches.

The R_y gate is a rotation around the Y-axis of the Bloch sphere. It helps in adjusting the quantum state in a way that can represent the data more accurately. The R_z gate is a rotation around the Z-axis of the Bloch sphere. It fine-tunes the phase of the quantum state, which can be crucial for encoding complex data patterns. By incorporating these gates, the circuit can navigate the state space more freely and capture more nuanced features of the input data. This potentially leads to better learning and generalization.

The Hadamard gates are used to create superposition states, which are essential for quantum parallelism. They enable the circuit to consider multiple states simultaneously, thus increasing the computational efficiency. These gates also facilitate the creation of Bell entangled states when combined with controlled-NOT (CNOT) gates. Entanglement is a unique quantum property that correlates the states of qubits, making it possible to represent more complex patterns and relationships within the data. The entanglement helps in capturing deeper semantic relationships within the data patches, which traditional classical methods might miss. This can lead to a more robust and accurate model.

The added R_y and R_z gates, along with the Hadamard gates, likely contribute to the model improved performance through the following mechanisms:

- **Enhanced Feature Representation:** The additional gates allow the quantum circuit to transform and represent the input data in a higher-dimensional space more effectively. This transformation helps in capturing more intricate patterns and features from the data.
- **Better Generalization:** The improved feature extraction and entanglement help the model generalize better to unseen data, reducing overfitting and improving accuracy.
- **Quantum Parallelism and Entanglement:** These quantum properties allow the model to explore multiple hypotheses simultaneously and capture complex dependencies within the data, which is often challenging for classical models.

The proposed enhancements to the VQC demonstrate a small but significant increase in accuracy, highlighting the potential benefits of leveraging additional quantum gates and entanglement in quantum circuits. The goal of QRKT-GAN is to generate realistic and high-quality synthetic data, leveraging the strengths of Quantum Computing to enhance generative models.

This chapter highlights the proposed hybrid architectural configurations, emphasizing their potential to integrate Quantum Computing principles into Deep Learning optimization. By using advanced technologies and innovative designs, such as ODE-based QViTs and QRKT-GAN, this research aims to push the boundaries of what is achievable in Deep Learning through the incorporation of Quantum Mechanics. The experiments will utilize popular and versatile datasets to validate the effectiveness of these hybrid models, showcasing their capability to address complex optimization problems in an energy-efficient manner.

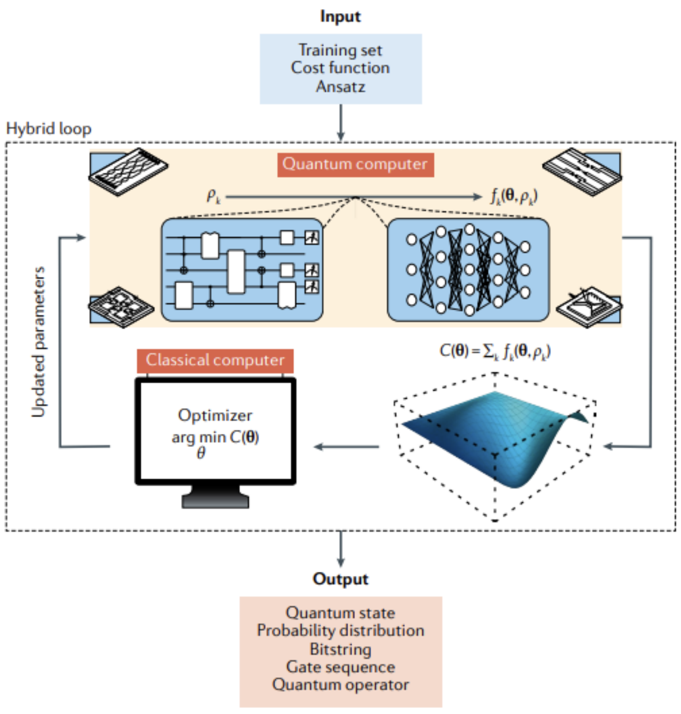


Figure 14: Quantum Machine Learning Summary [10]

5 Experiments

This chapter presents the methodologies used during the training and evaluation of the ODE models for classification tasks in computer vision (CV) and natural language processing (NLP) before the generation task. The datasets utilized in these experiments and the specific configurations of the QVits models are detailed. The final section of this chapter explores the generative capabilities of the QRKT-GAN compared to its classical baseline, TransGAN. All models are evaluated using consistent metrics, with an additional metric for the hybrid models: the number of qubits employed in learning the parameters of the qurons in the Variational Quantum Circuits (VQCs).

To begin with, the classification tasks in CV and NLP are approached using ODE models. These models were trained and evaluated on a range of benchmark datasets. For CV tasks, datasets such as MNIST and CIFAR-10 were used. For NLP task, IMDB for sentiment analysis was employed. Each dataset provided unique challenges and opportunities for the ODE models to demonstrate their robustness and adaptability. In terms of QVits configurations, various architectures were explored. The configurations varied in terms of the depth and width of the neural networks.

The generative capabilities of the QRKT-GAN is rigorously compared with the classical baseline TransGAN. The QRKT-GAN incorporates quantum-inspired techniques to enhance generative performance. The dataset used is CIFAR-10 for general object generation. Both models were evaluated based on standard generative metrics, such as Inception Score (IS) and Fréchet Inception Distance (FID).

An additional metric for the hybrid models is the number of qubits used in the VQCs. This metric is crucial as it provides insight into the quantum resource requirements for achieving specific performance levels. The evaluation highlighted that while QRKT-GAN required fewer parameters than TransGAN, it utilized a significant number of qubits, indicating a trade-off between classical and quantum resources.

Furthermore, classical and hybrid architectures are configured with a comparable number of components listed in chapter 4. Given the fact that the input and output states of a Variational Quantum Circuit (VQC) share the same dimensions, the qubit count must match the size of the neural network corresponding layers. The designed and proposed VQC has $2n$ parameters compared to the classical fully connected layer $n^2 + n$ parameters.

The dimensions are kept minimal to limit qubit requirements, thereby reducing simulation time and enabling execution on current quantum hardware. A batch size of 65 is employed, with training spanning 65 epochs using the AdamW optimizer [28], incorporating gradient clipping at norm 1, and a learning rate scheduler that transitions from a linear warmup over 5000 steps (0 to 10^{-3}) to a cosine decay [80]. A random exhaustively hyperparameter search [83] is conducted to optimize parameters for the classical baseline, which are then applied to the QViT.

The classical components of the model, along with the classical baseline, are implemented and trained using JAX [94] and Flax [95, 49]. TensorCircuit [17] is utilized for the implementation, training, and execution of the VQCs via numerical simulation on a classical computer. This approach allows the quantum model to be trained for multiple epochs in a relatively decent duration. This represents a significant improvement over previous efforts, such as the work by Di Sipio et al. [6], which required around 100 hours to train a similar hybrid Transformer model for a single epoch, despite having a larger sample size.

5.1 MNIST Digit Classification

The MNIST [15] dataset is a large database of handwritten digits commonly used for training various Image Processing systems. This consistent dataset contains 60,000 training images and 10,000 testing images, each of which is a 28x28 grayscale image of a single digit from 0 to 9. MNIST serves as a standard benchmark in Machine Learning due to its simplicity and the ease with which models can achieve high accuracy. It is particularly useful for validating new algorithms and models.

In this experiment, the dataset is strategically divided into three distinct parts to ensure comprehensive model evaluation and optimization. The training split, containing 54,000 images, provides a robust foundation for the model to learn diverse patterns and features. For hyperparameter tuning and model refinement, a validation split of 6,000 images is used during the Random Search process. This helps in selecting the best-performing model configuration. Lastly, to rigorously test the model generalization capability, 10,000 images are reserved in the test set, which remains unseen during the training phase. This approach ensures that the model is not only well-trained but also performs effectively on new, unseen data, reflecting its true predictive power.

Some extracted samples are shown below:

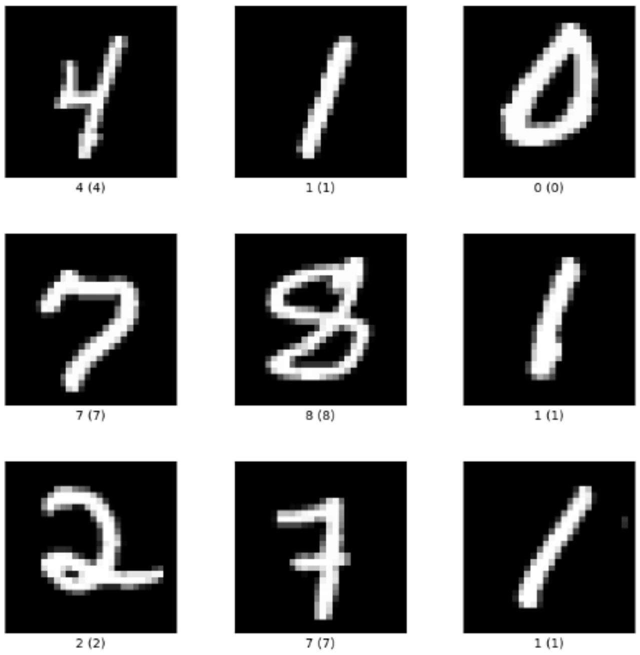


Figure 15: Examples from MNIST[1]

[1]https://www.tensorflow.org/datasets/catalog/mnist

The evolution of the loss and AUC score during **one classical ODE block**-based models learning, computed at the end of each epoch, are shown:

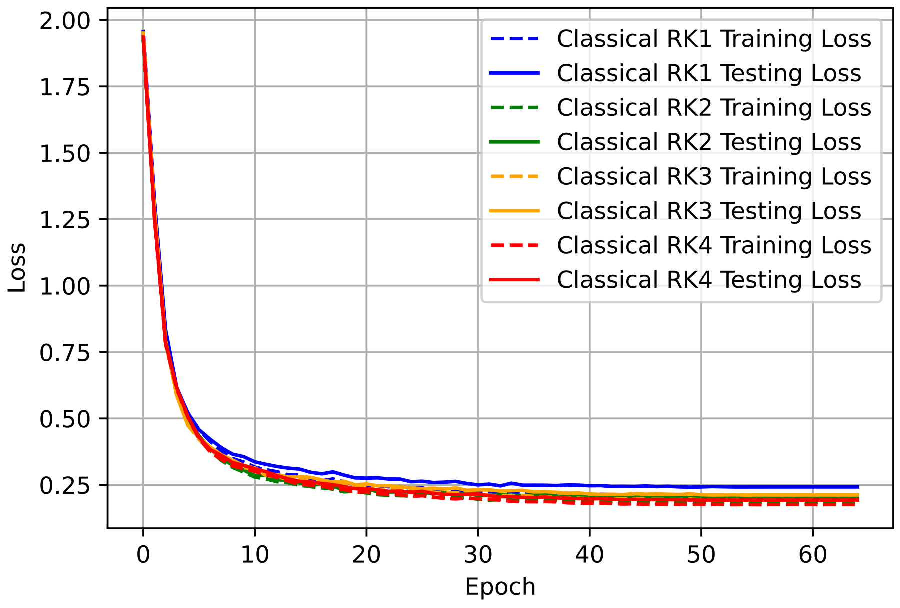


Figure 16: Cross-entropy loss evolution during learning

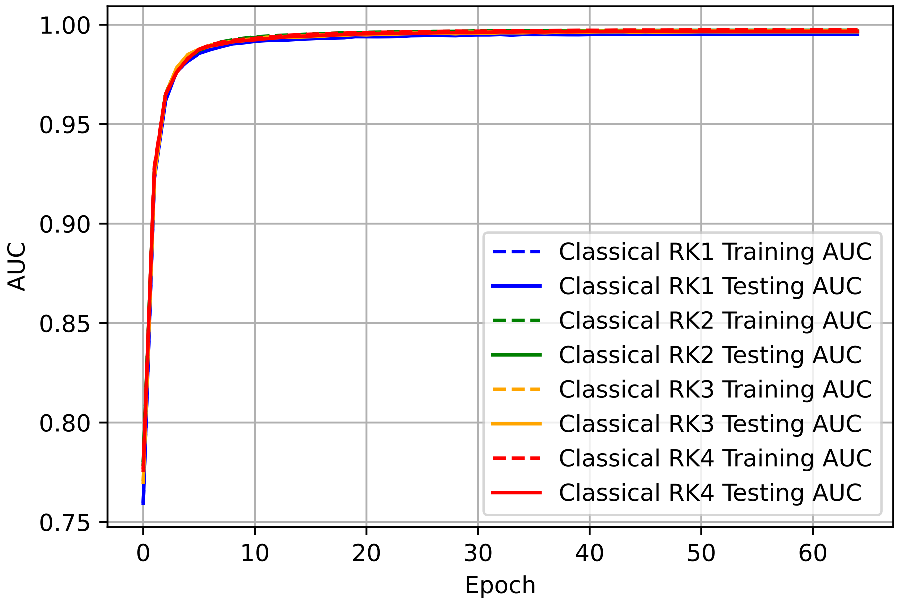


Figure 17: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Patch Size**: 14
- **Hidden Size**: 6
- **Classical ODE-Transformer Blocks**: 1
- **Classical Attention Heads**: 2
- **Hidden MLP Size**: 3

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
RK1	1544.82	94%	93%	50	3733	-
RK2	1524.14	94%	94%	55	3946	-
RK3	1539.63	94%	94%	51	4159	-
RK4	1560.66	95%	94%	51	4372	-

Table 1: MNIST metrics for the classical baselines

Observations:

The F1 Scores are consistently high across all models, with RK1 scoring slightly lower at 93%. RK2, RK3, and RK4 all achieve a 94% F1 Score, indicating a good balance between precision and recall for these models.

The epoch at which the best Area Under the Curve (AUC) is achieved varies slightly. RK2 reaches its best AUC performance at epoch 55, which is higher than the others. RK1 achieves its best AUC at epoch 50, while RK3 and RK4 both reach their best performance at epoch 51.

The number of parameters increases progressively from RK1 to RK4. RK1 has the fewest parameters at 3733, while RK4 has the most at 4372. This indicates that RK4 is a more complex model compared to the others, which may contribute to its slightly better performance metrics.

And between the optimized RK4 and traditional RK4:

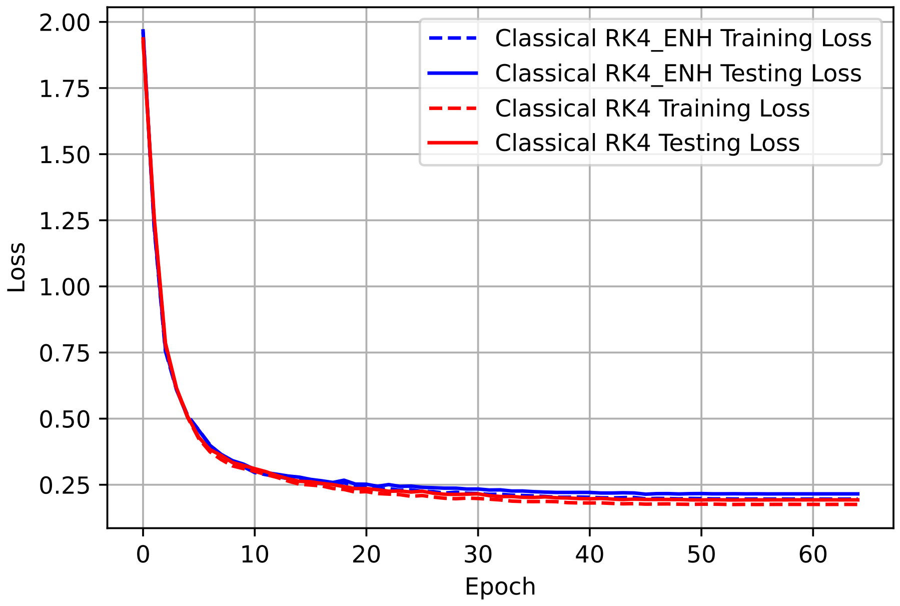


Figure 18: Cross-entropy loss evolution during learning

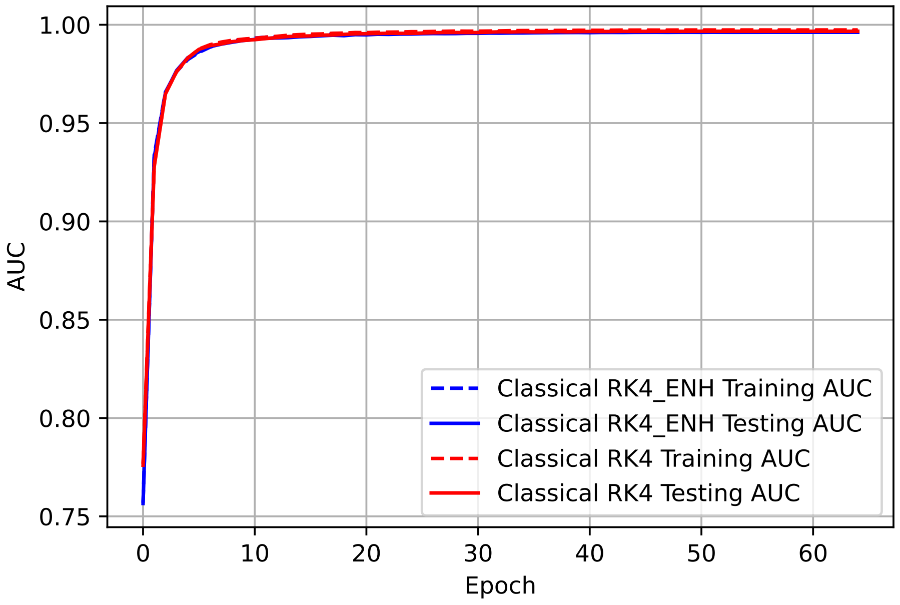


Figure 19: AUC score evolution during learning

For the MNIST classification task, the optimized variant does not demonstrate any significant improvement in training loss, testing loss, or AUC. This lack of impact could be attributed to the relatively small dimensionality of the problem, where further adjustments yield negligible effects. Consequently, the inherent simplicity of the dataset may limit the benefits of optimization techniques.

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
RK4	1560.66	95%	94%	51	4372	-
RK4_ENH	1562.54	94%	94%	49	4372	-

Table 2: MNIST metrics for the classical baselines

Observations:

The training times for RK4 and RK4_ENH are very close, with RK4_ENH taking slightly longer. This suggests that the enhancements in RK4_ENH do not significantly increase the training time, making it comparable in terms of computational requirements.

RK4 achieves a slightly higher accuracy of 95% compared to 94% for RK4_ENH. This indicates that the base RK4 model is marginally better at correctly classifying the MNIST dataset images.

Both models have an identical F1 Score of 94%, demonstrating a consistent balance between precision and recall. This suggests that the enhancement does not significantly affect the F1 Score.

RK4 reaches its best AUC at epoch 51, whereas RK4_ENH reaches it at epoch 49. This slight difference indicates that RK4_ENH achieves its optimal performance slightly earlier in the training process compared to RK4.

Both models have the same number of parameters, indicating that the enhancements in RK4_ENH do not involve changes in the model complexity or size.

The evolution of the loss and AUC score during **three classical ODE blocks**-based models learning, computed at the end of each epoch, are shown:

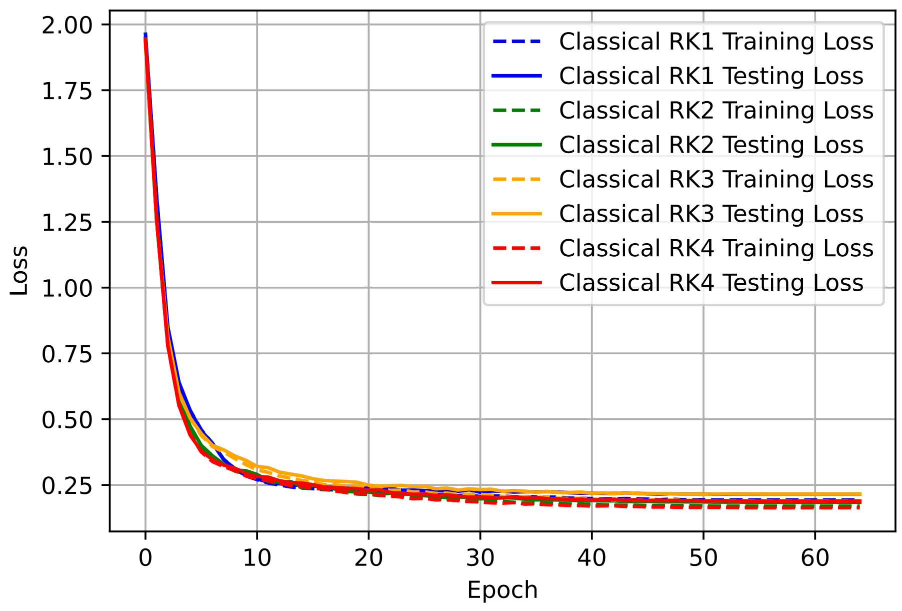


Figure 20: Cross-entropy loss evolution during learning

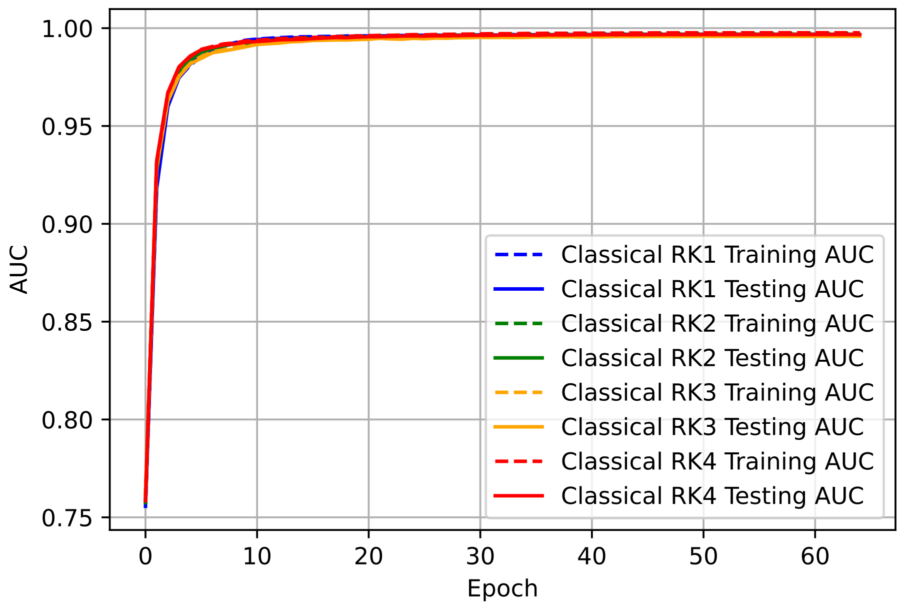


Figure 21: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Patch Size**: 14
- **Hidden Size**: 6
- **Classical ODE-Transformer Blocks**: 3
- **Classical Attention Heads**: 2
- **Hidden MLP Size**: 3

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
RK1	1582.73	94%	94%	50	3472	-
RK2	1689.24	95%	95%	53	4693	-
RK3	1850.70	94%	94%	55	5332	-
RK4	1925.89	95%	95%	51	5971	-

Table 3: MNIST metrics for the classical baselines

Observations:

Both RK2 and RK4 achieve the highest scores, demonstrating that higher-order methods can offer slight improvements in classification performance.

RK1 and RK3 both achieve an accuracy and F1 score of 94%, which is marginally lower than RK2 and RK4. This suggests that while all methods are effective, the second and fourth-order methods have a slight edge in performance.

Training time increases with the order of the Runge-Kutta method. RK1, being the simplest and least computationally intensive method, has the shortest training time. As the order increases, so does the complexity and the computational load, resulting in longer training times: RK2, RK3, and RK4.

The epoch at which the model achieves its best Area Under the Curve (AUC) varies slightly among the methods. RK1 and RK4 reach their peak AUC at around 50 and 51 epochs respectively, while RK2 and RK3 reach it slightly later at 53 and 55 epochs.

More complex methods involve more parameters, which can contribute to improved performance but also add to the model complexity and training time.

And between the optimized RK4 and traditional RK4:

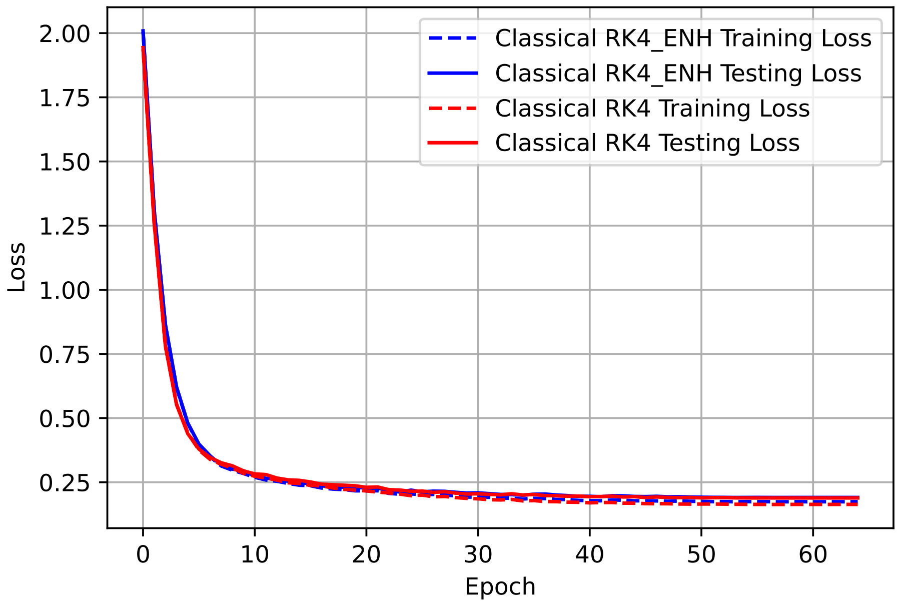


Figure 22: Cross-entropy loss evolution during learning

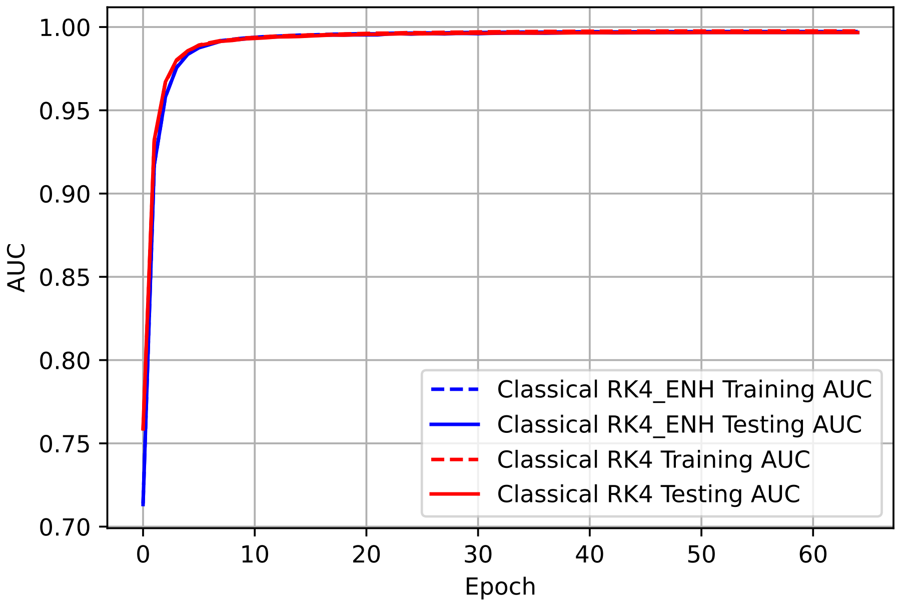


Figure 23: AUC score evolution during learning

Like before, The optimized variant does not show any significant improvement in training loss, testing loss, or AUC. This lack of impact could be due to the relatively small dimensionality of the problem, where further adjustments have minimal effects. Consequently, the inherent simplicity of the dataset may limit the advantages of optimization techniques.

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
RK4	1925.89	95%	95%	51	5971	-
RK4_ENH	1842.04	95%	95%	54	5971	-

Table 4: MNIST metrics for the classical baselines

Observations:

The training time for the enhanced RK4 method is slightly less than the standard RK4, suggesting improved computational efficiency.

Both RK4 and RK4_ENH achieve an accuracy of 95%, indicating that the enhancement does not compromise the classification performance.

Both methods achieve an F1 score of 95%, reaffirming that the enhancement maintains the model ability to balance precision and recall.

The enhanced model reaches its peak AUC slightly later, at the 54th epoch. This might indicate a different learning curve or optimization process.

These methods have the same number of parameters, indicating that the enhancement does not involve increasing the model complexity in terms of parameters.

The evolution of the loss and AUC score during **three ODE quantum blocks**-based models learning, computed at the end of each epoch, are shown:

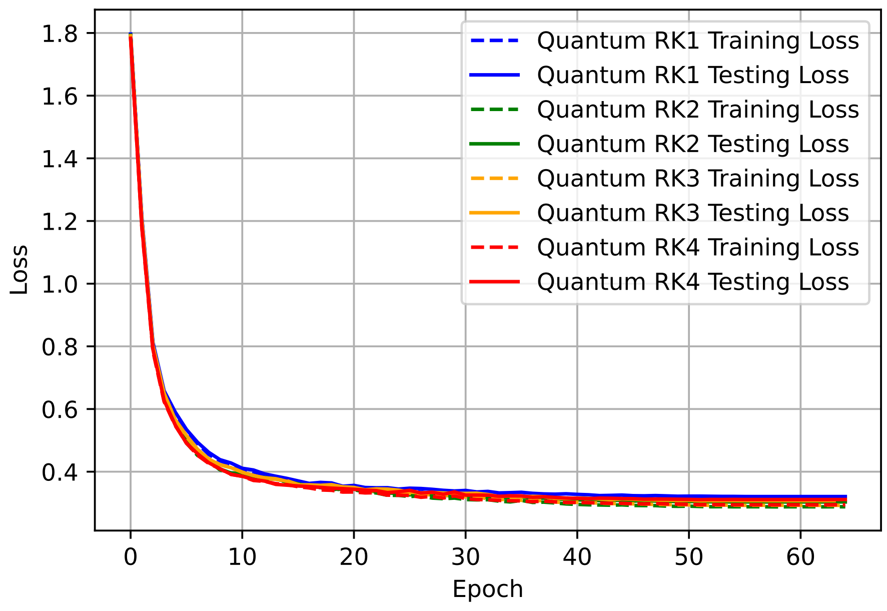


Figure 24: Cross-entropy loss evolution during learning

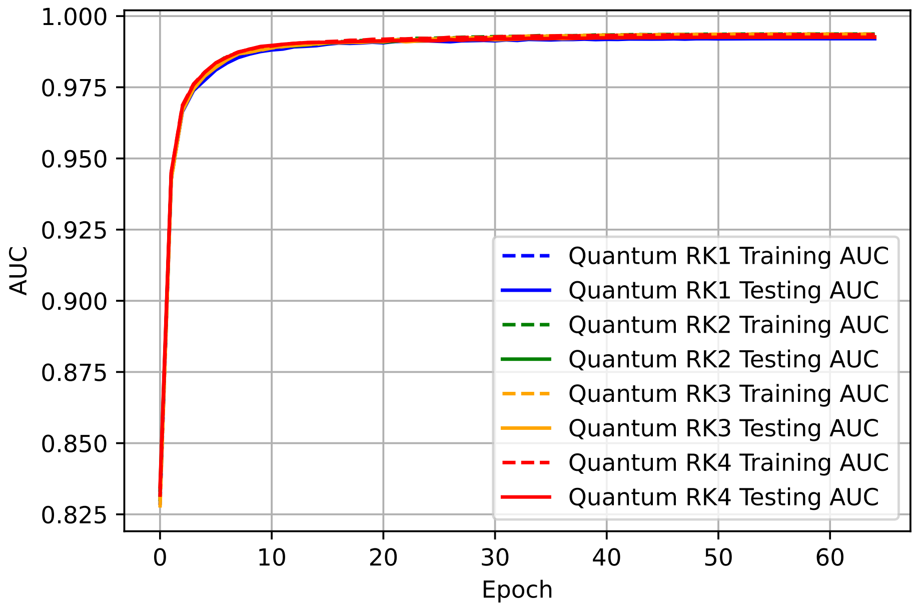


Figure 25: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Patch Size**: 14
- **Hidden Size**: 6
- **Quantum ODE-Transformer Blocks**: 3
- **Quantum Attention Heads**: 2
- **Hidden QMLP Size**: 3

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
QRK1	2051.72	91%	91%	56	2872	114
QRK2	2442.47	91%	91%	47	3088	195
QRK3	2945.24	91%	91%	52	3304	276
QRK4	3594.99	91%	91%	52	3520	357

Table 5: MNIST metrics for the quantum configurations

Observations:

Training time increases with the order of the quantum Runge-Kutta method, reflecting the increasing computational complexity.

All quantum methods achieve an accuracy of 91%. This consistency suggests that the order of the method does not significantly impact accuracy in the quantum implementations.

Similar to accuracy, the F1 score is 91% for all quantum methods, indicating consistent performance in balancing precision and recall.

The best AUC epoch varies, with QRK2 reaching its peak performance earliest at 47 epochs, while QRK1 reaches it the latest at 56 epochs.

The number of parameters increases with the order of the method, which is expected as higher-order methods typically involve more complex calculations.

Higher-order methods require significantly more qubits, indicating a substantial increase in quantum resource requirements.

And between the optimized RK4 and traditional RK4:

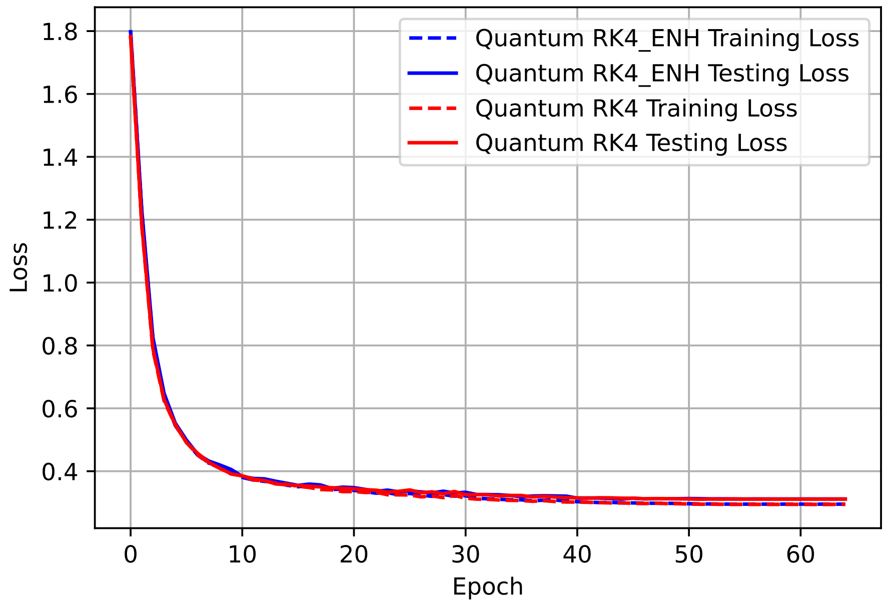


Figure 26: Cross-entropy loss evolution during learning

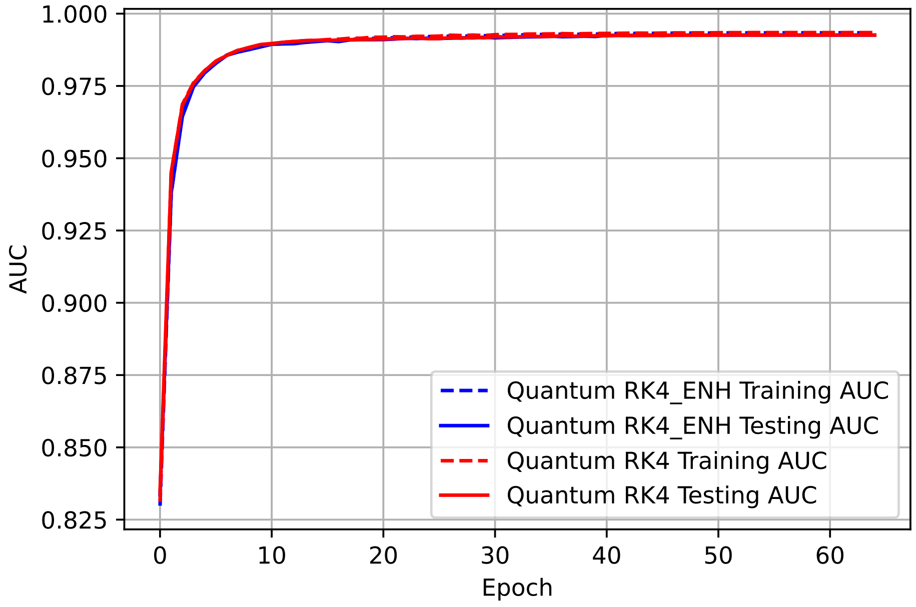


Figure 27: AUC score evolution during learning

Similar to previous observations, the optimized variant fails to exhibit significant improvements in training loss, testing loss, or AUC. This lack of noticeable impact might stem from the relatively small dimensionality of the problem, where further adjustments produce negligible effects. As a result, the inherent simplicity of the dataset may constrain the potential benefits of optimization techniques.

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
QRK4	3594.99	91%	91%	52	3520	357
QRK4_ENH	3539.44	91%	91%	48	3520	357

Table 6: MNIST metrics for the quantum configurations

Observations:

The enhanced QRK4 method has a slightly reduced training time, suggesting improved computational efficiency.

Both QRK4 and QRK4_ENH achieve an accuracy of 91%, indicating that the enhancement does not impact the classification performance negatively.

Both methods have an F1 score of 91%, which means the enhanced method maintains the same balance between precision and recall as the standard method.

The enhanced model reaches its peak AUC earlier, at the 48th epoch, indicating potentially faster convergence.

Both methods have the same number of parameters, suggesting that the enhancement does not involve increasing the model complexity in terms of parameters.

Both methods require the same number of qubits, indicating that the enhancement does not increase quantum resource requirements.

The evolution of the loss and AUC score during **three ODE quantum and classical blocks**-based models learning, computed at the end of each epoch, are shown:

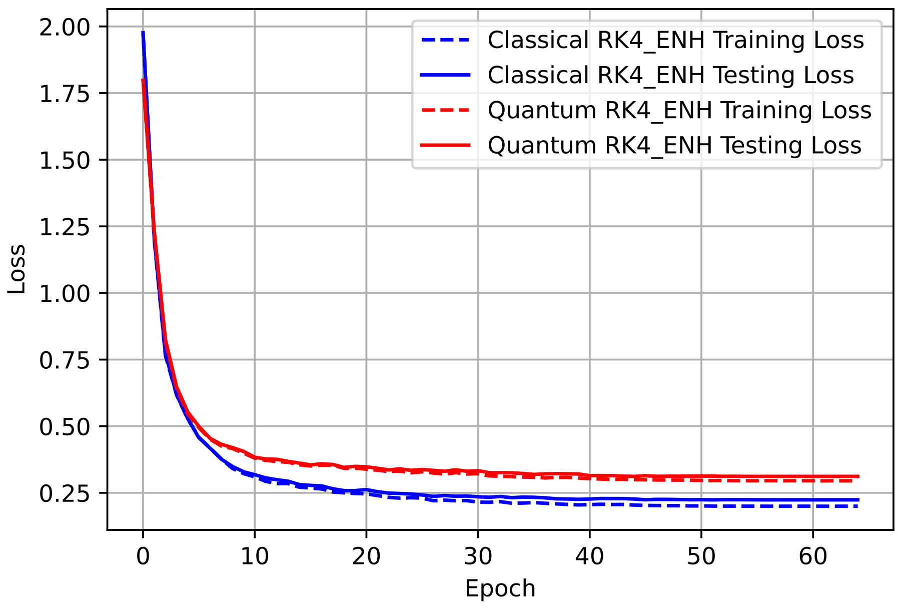


Figure 28: Cross-entropy loss evolution during learning

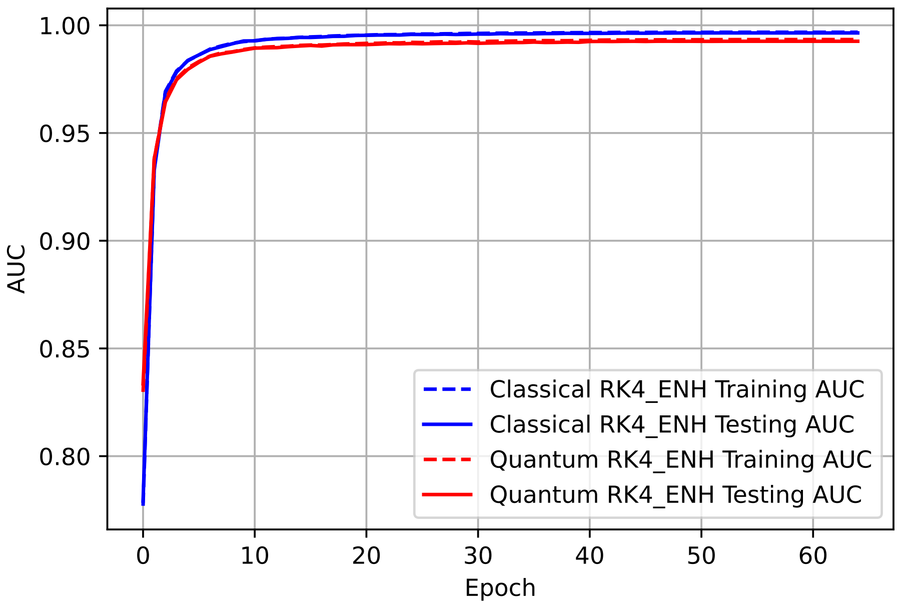


Figure 29: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Patch Size**: 14
- **Hidden Size**: 6
- **Classical and Quantum ODE-Transformer Blocks**: 3
- **Classical and Quantum Attention Heads**: 2
- **Hidden QMLP Size**: 3

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
RK4_ENH	1842.04	95%	95%	54	5971	-
QRK4_ENH	3539.44	91%	91%	48	3520	357

Table 7: MNIST metrics for the quantum and classical configurations

Observations:

The training time for the enhanced quantum QRK4 method is significantly longer than the classical method.

The accuracy of the enhanced quantum method is lower compared to the classical method.

The F1 score of the enhanced quantum method is consistent with its accuracy, indicating a balanced precision and recall.

The enhanced quantum method achieves its best AUC performance earlier, suggesting faster convergence.

The quantum method has fewer parameters, indicating potentially lower model complexity.

5.2 CIFAR-10 Object Classification

CIFAR-10 is another widely used dataset in the field of Computer Vision. It consists of 60,000 32x32 color images in 10 different classes, with 6,000 images per class. The classes include airplanes, cars, birds, cats, deer, dogs, frogs, horses, ships, and trucks. The dataset is divided into 50,000 training images and 10,000 testing images. CIFAR-10 is more complex than MNIST due to its color images and diverse range of object classes, making it a suitable challenge for evaluating the performance of more advanced models.

In this experiment, the dataset is meticulously divided into three distinct segments to ensure thorough model evaluation and optimization. The training split, comprising 45,000 images, offers a solid foundation for the model to learn diverse patterns and features. A validation split of 5,000 images is employed during the Random Search process for hyperparameter tuning and model refinement, facilitating the selection of the best-performing model configuration. Finally, to rigorously assess the model generalization capability, 10,000 images are set aside in the test set, remaining unseen during the training phase. This method ensures that the model is not only well-trained but also demonstrates effective performance on new, unseen data, accurately reflecting its true predictive power.

Some extracted samples are shown below:

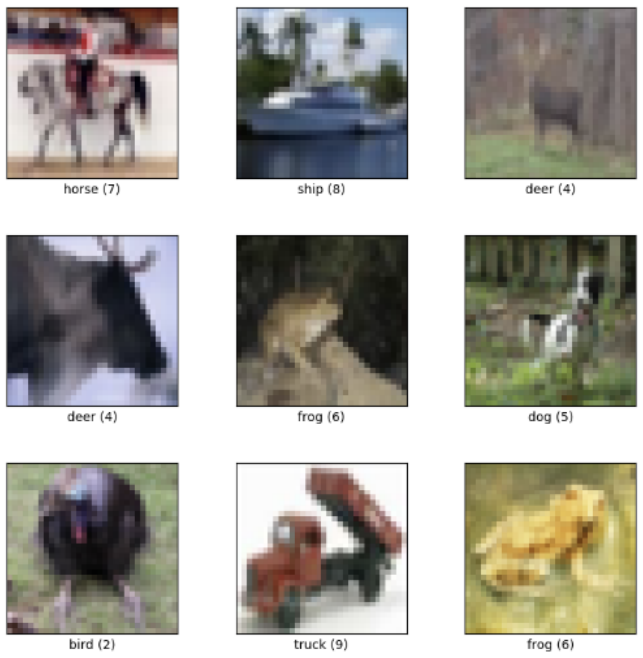


Figure 30: Examples from CIFAR-10[2]

[2]https://www.tensorflow.org/datasets/catalog/cifar10

The evolution of the loss and AUC score during **one classical ODE block**-based models learning, computed at the end of each epoch, are shown:

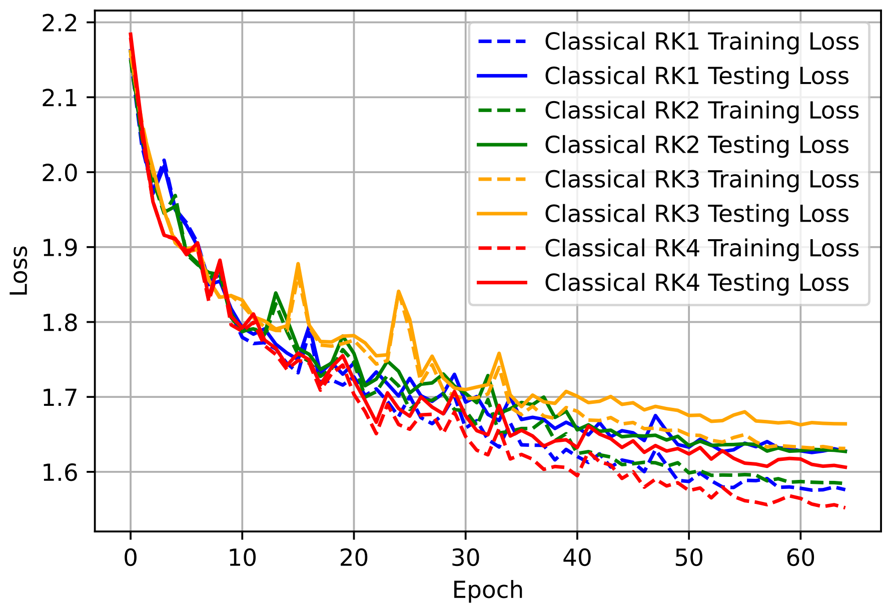


Figure 31: Cross-entropy loss evolution during learning

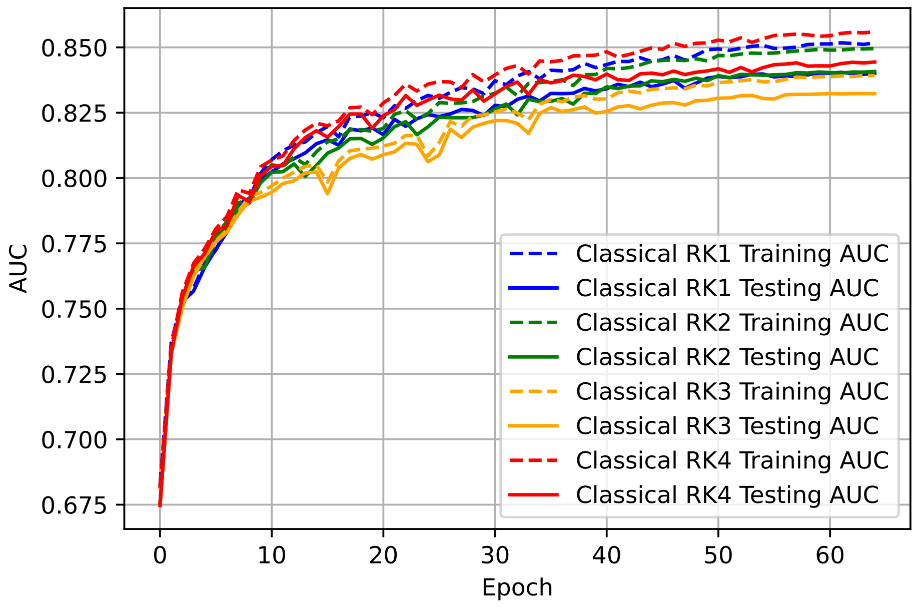


Figure 32: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Patch Size**: 16
- **Hidden Size**: 12
- **Classical ODE-Transformer Blocks**: 1
- **Classical Attention Heads**: 6
- **Hidden MLP Size**: 6

ODE	Train Time (s)	Accuracy	F1 Score	Best AUC Epoch	# Parameters	# Qubits
RK1	1568.55	41%	41%	62	28894	-
RK2	1521.45	40%	40%	65	29680	-
RK3	1464.02	38%	38%	61	28084	-
RK4	1685.04	42%	42%	65	33634	-

Table 8: CIFAR-10 metrics for the classical configurations

Observations:

Training time varies slightly among the methods, with RK3 having the shortest training time and RK4 the longest.

The accuracy ranges from 38% to 42%, with RK4 achieving the highest accuracy.

The F1 score mirrors the accuracy, indicating a consistent balance between precision and recall across methods.

The best AUC epoch varies, with RK2 and RK4 achieving their peak performance later than RK1 and RK3.

The number of parameters increases with the order of the method, with RK4 having the most parameters.

And between the optimized RK4 and traditional RK4:

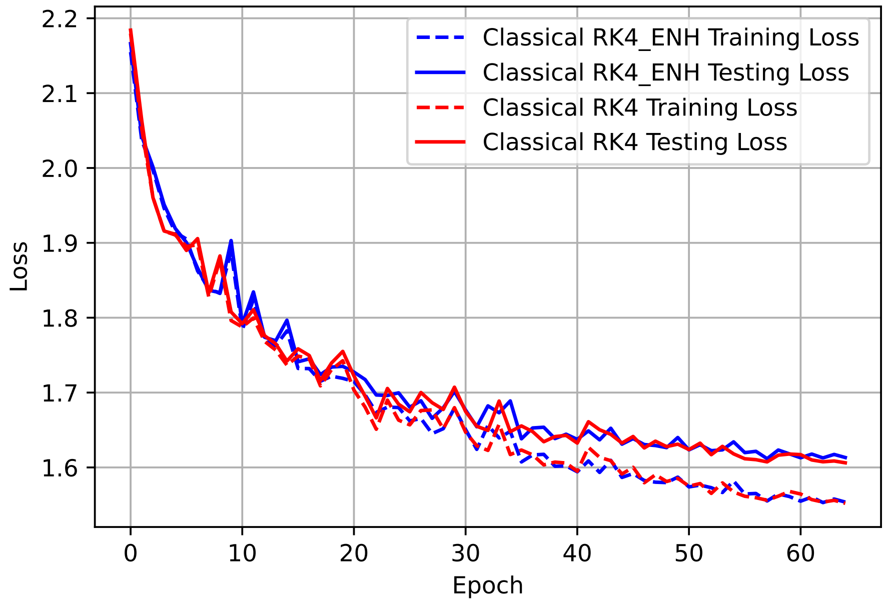


Figure 33: Cross-entropy loss evolution during learning

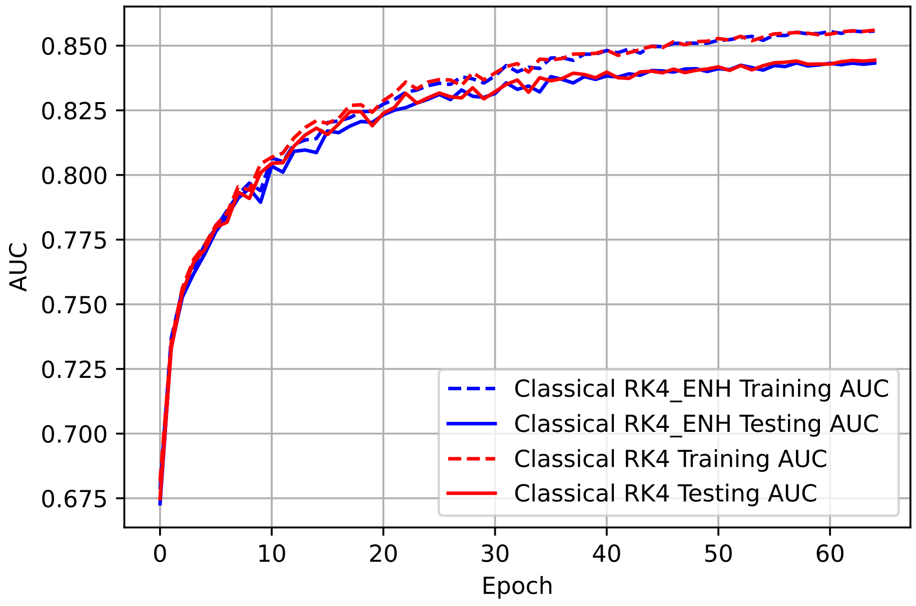


Figure 34: AUC score evolution during learning

For the CIFAR-10 dataset, a more complex problem compared to MNIST, the optimized variant demonstrates noteworthy differences. Although it does not significantly improve training loss, testing loss, or AUC, it converges faster than the standard version. This optimized model requires fewer parameters and trains more quickly, showcasing efficiency gains. The increased complexity of CIFAR-10 means further adjustments have a more pronounced impact, but the inherent challenges of the dataset still limit substantial improvements in accuracy metrics. Nonetheless, the optimized version faster convergence and reduced parameter count highlight valuable advancements in model performance within this more intricate problem space.

ODE	**Train Time (s)**	**Accuracy**	**F1 Score**	**Best AUC Epoch**	**# Parameters**	**# Qubits**
RK4	1685.04	42%	42%	65	33634	-
RK4_ENH	1572.41	41%	41%	58	31252	-

Table 9: CIFAR-10 metrics for the classical configurations

Observations:

The enhanced RK4 method has a reduced training time, suggesting improved computational efficiency.

The enhanced RK4 method shows a slight decrease in accuracy compared to the standard RK4 method.

The F1 score of the enhanced RK4 method is consistent with its accuracy, indicating a balanced precision and recall.

The enhanced RK4 method achieves its best AUC performance earlier, suggesting faster convergence.

The enhanced RK4 method has fewer parameters, indicating a reduction in model complexity.

The evolution of the loss and AUC score during **one ODE quantum and classical blocks**-based models learning, computed at the end of each epoch, are shown:

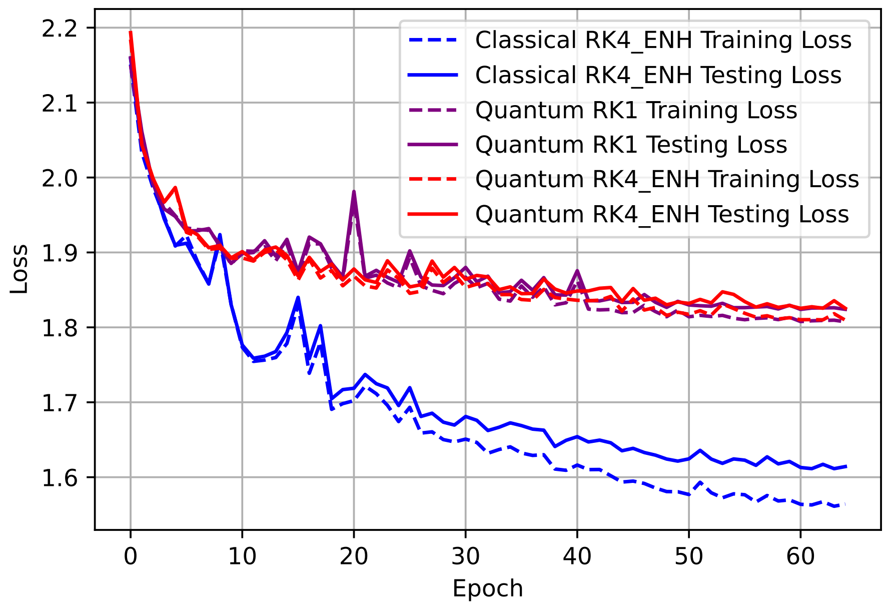


Figure 35: Cross-entropy loss evolution during learning

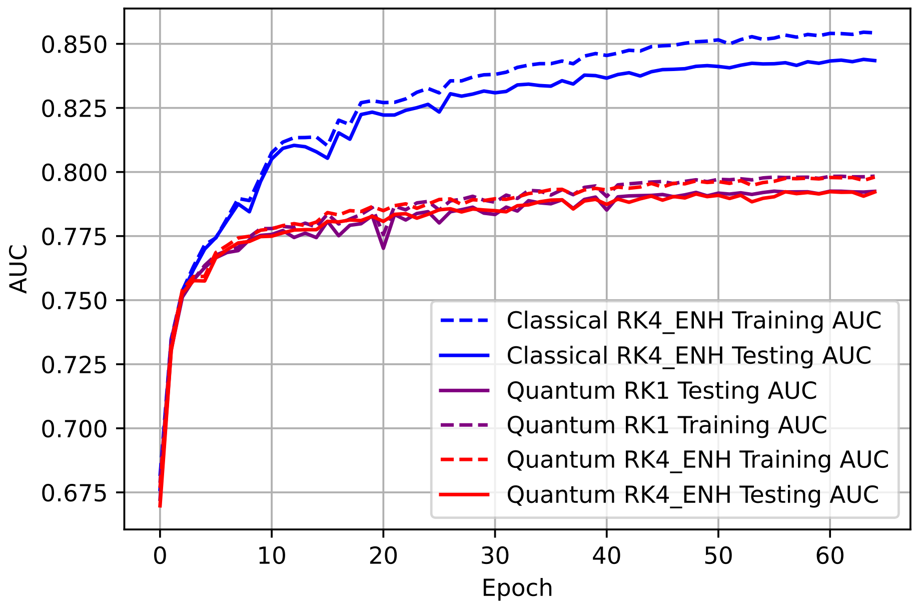


Figure 36: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Patch Size**: 16
- **Hidden Size**: 12
- **Classical and Quantum ODE-Transformer Blocks**: 1
- **Classical and Quantum Attention Heads**: 6
- **Hidden QMLP Size**: 6

ODE	**Train Time (s)**	**Accuracy**	**F1 Score**	**Best AUC Epoch**	**# Parameters**	**# Qubits**
RK4_ENH	1685.04	42%	42%	65	33634	-
QRK4_ENH	16724.79	34%	33%	61	20590	390
QRK1	10909.40	33%	33%	56	20590	336

Table 10: CIFAR-10 metrics for the quantum and classical configurations

Observations:

The training time for the enhanced quantum QRK4 method is significantly longer than the classical method. The training time for the first-order quantum RK method is shorter than QRK4_ENH but still much longer than RK4_ENH.

The accuracy of the enhanced quantum method is lower compared to the classical method. The first-order quantum RK method has slightly lower accuracy than the QRK4_ENH method.

The F1 score of the first-order quantum RK method matches its accuracy, indicating balanced precision and recall.

The first-order quantum RK method achieves its best AUC performance the earliest among the methods listed.

The number of parameters is the same for both quantum methods.

The quantum methods require a significant number of qubits, highlighting the substantial quantum resources needed.

5.3 IMDb Sentiment Analysis

IMDb is a widely used dataset in Natural Language Processing, particularly for sentiment analysis tasks. It comprises 50,000 movie reviews, equally divided into 25,000 training reviews and 25,000 testing reviews. Each review is labeled as either positive or negative, providing a binary classification challenge. The dataset complexity arises from the nuances of human language, including sarcasm, idioms, and varying lengths of reviews, making it an excellent benchmark for evaluating advanced sentiment analysis models.

In this experiment, the IMDb dataset is meticulously divided into three distinct segments to ensure thorough model evaluation and optimization. The training split, comprising 22,500 reviews, provides a robust foundation for the model to learn diverse linguistic patterns and sentiment cues. A validation split of 2,500 reviews is utilized during the Random Search process for hyperparameter tuning and model refinement, enabling the selection of the best-performing model configuration. Finally, to rigorously assess the model generalization capability, 25,000 reviews are reserved in the test set, remaining unseen during the training phase. This method ensures that the model is not only well-trained but also demonstrates effective performance on new, unseen data, accurately reflecting its true predictive power in sentiment analysis tasks.

Some extracted samples are shown below:

Label	Text
0 (neg)	I have been known to fall asleep during films, but this is usually due to a combination of things including, really tired, being warm and comfortable on the settee and having just eaten a lot. However on this occasion I fell asleep because the film was rubbish. The plot development was constant. Constantly slow and boring. Things seemed to happen, but with no explanation of what was causing them or why. I admit, I may have missed part of the film, but I watched the majority of it and everything just seemed to happen of its own accord without any real concern for anything else. I can't recommend this film at all.
1 (pos)	This is a film which should be seen by anybody interested in, effected by, or suffering from an eating disorder. It is an amazingly accurate and sensitive portrayal of bulimia in a teenage girl, its causes and its symptoms. The girl is played by one of the most brilliant young actresses working in cinema today, Alison Lohman, who was later so spectacular in 'Where the Truth Lies'. I would recommend that this film be shown in all schools, as you will never see a better on this subject. Alison Lohman is absolutely outstanding, and one marvels at her ability to convey the anguish of a girl suffering from this compulsive disorder. If barometers tell us the air pressure, Alison Lohman tells us the emotional pressure with the same degree of accuracy [...]

Table 11: Movie Reviews[3]

[3] https://www.tensorflow.org/datasets/catalog/imdb_reviews

The evolution of the loss and AUC score during **one classical ODE block**-based models learning, computed at the end of each epoch, are shown:

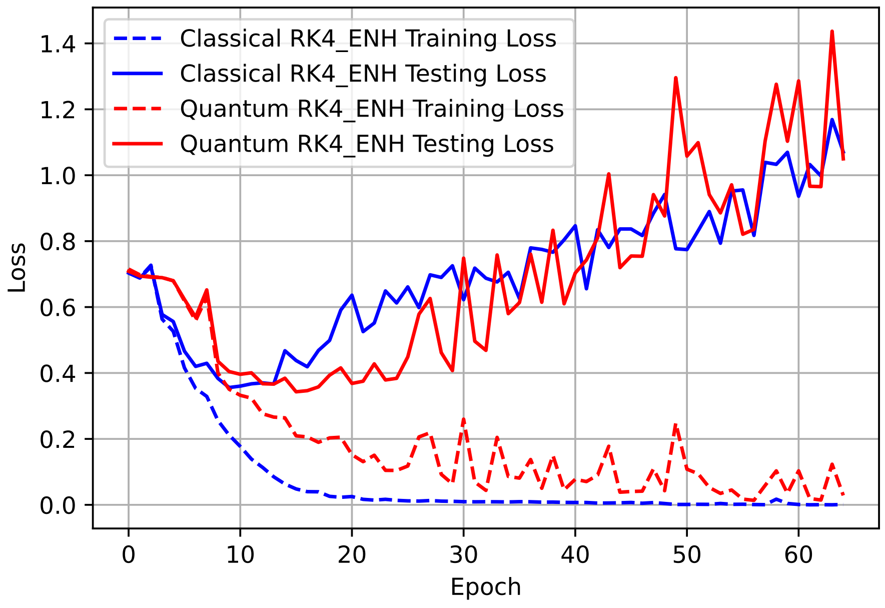


Figure 37: Cross-entropy loss evolution during learning

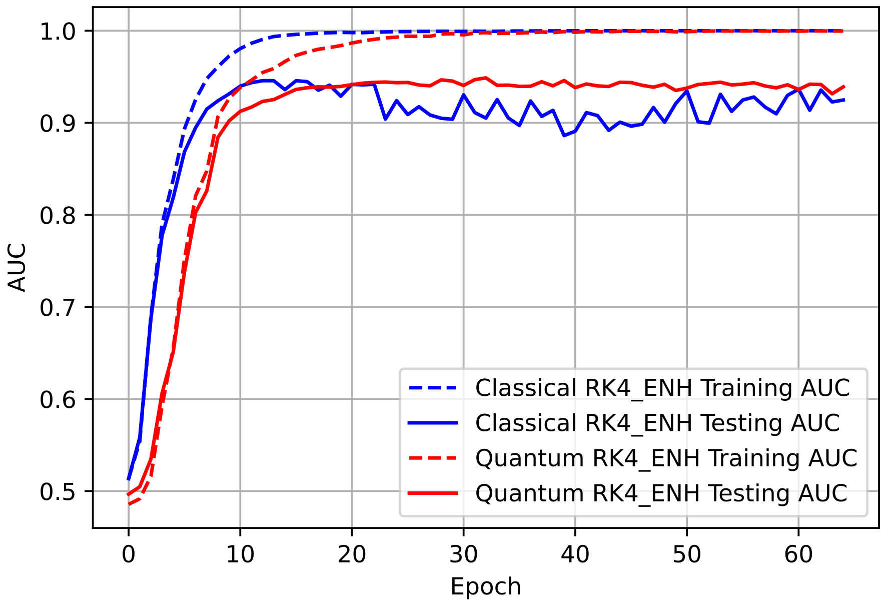


Figure 38: AUC score evolution during learning

This implementation showcases the following components and parameters:

- **Max sequence length**: 512
- **Classical / Quantum Hidden Size**: 6 / 12
- **Classical / Quantum ODE-Transformer Blocks**: 1 / 1
- **Classical / Quantum Attention Heads**: 2 / 6
- **Hidden MLP Size**: 3 / 6

ODE	**Train Time (s)**	**Accuracy**	**F1 Score**	**Best AUC Epoch**	**# Parameters**	**# Qubits**
RK4_ENH	3328.61	85%	85%	13	499316	-
QRK4_ENH	9033.13	**85**%	**85**%	33	**243896**	141

Table 12: IMDb metrics for the classical configurations

The quantum-enhanced configuration (QRK4_ENH) takes significantly longer to train, almost three times as long as the classical configuration (RK4_ENH). This suggests that while quantum methods may have computational benefits, they currently require more training time, possibly due to the complexity of quantum computations and the need for more iterations to converge.

Despite the increased training time for the quantum-enhanced model, it does not result in higher accuracy or F1 score compared to the classical model. This indicates that the quantum enhancement does not compromise the model performance on these metrics.

The best epoch for the Area Under the Curve (AUC) metric occurs earlier for the classical configuration compared to the quantum-enhanced one. This could imply that the classical model reaches its optimal performance faster during training, whereas the quantum-enhanced model requires more epochs to achieve its best performance.

The quantum-enhanced configuration has approximately 51% fewer parameters than the classical configuration. This significant reduction in the number of parameters is a crucial observation, as it highlights the potential of quantum methods in optimizing model complexity and reducing resource requirements without sacrificing performance.

5.4 QRKT-GAN and TransGAN Synthetic Data Generation

This section provides a detailed comparison of the synthetic data generation capabilities of two distinct Generative Adversarial Networks (GANs): TransGAN and QRKT-GAN. While both aim to generate high-quality images, their underlying architectures and training processes differ significantly.

The training hyperparameters for both GANs are summarized as follows [99]:

- Learning Rate for Generator: 0.0001
- Learning Rate for Discriminator: 0.0001
- Latent Dimension: 1024
- Batch Size for Generator: 32
- Batch Size for Discriminator: 32
- Number of Epochs: 40
- Weight Decay: 1e-3
- Dropout Rate: 0.5
- Number of Critic Steps: 5
- Maximum Iterations: 500000
- Learning Rate Decay: True

The architectural details are as follows:

- Image Size: 32
- Initial Size: 8
- Patch Size: 4
- Embedding Dimension: 384
- Optimizer: 'Adam'
- Loss Function: "wgangp_eps"
- Phi: 1
- Beta1: 0
- Beta2: 0.99
- Data Augmentation: True

Due to high computational limitations, QRKT-GAN has weaker architectural components for both the Generator and Discriminator. Both QRKT-GAN and TransGAN were trained for only 40 epochs, owing to the very high generation time and memory consumption for QRKT-GAN. The training utilized the Nvidia A100-PCIE-40GB GPU cluster. For QRKT-GAN, the data augmentation techniques specified were applied to mitigate some of these limitations.

However, the ODE-based Transformer block using optimized RK4 could be applied to the Generator network, further enhancing the quality of images and the obtained scores.

The Discriminator, unfortunately, doesn't have any ODE blocks in its composition due to high memory consumption during training. It exclusively employs Encoders integrated with Variational Quantum Circuits (VQCs).

The performance metrics for QRKT-GAN and TransGAN are summarized in the table below:

Metric	QRKT-GAN	TransGAN
Inception Score (IS)	74.89	52.31
Fréchet Inception Distance (FID)	6.78	46.97

Table 13: Performance Metrics for TransGAN and QRKT-GAN on CIFAR-10 using 40 epochs

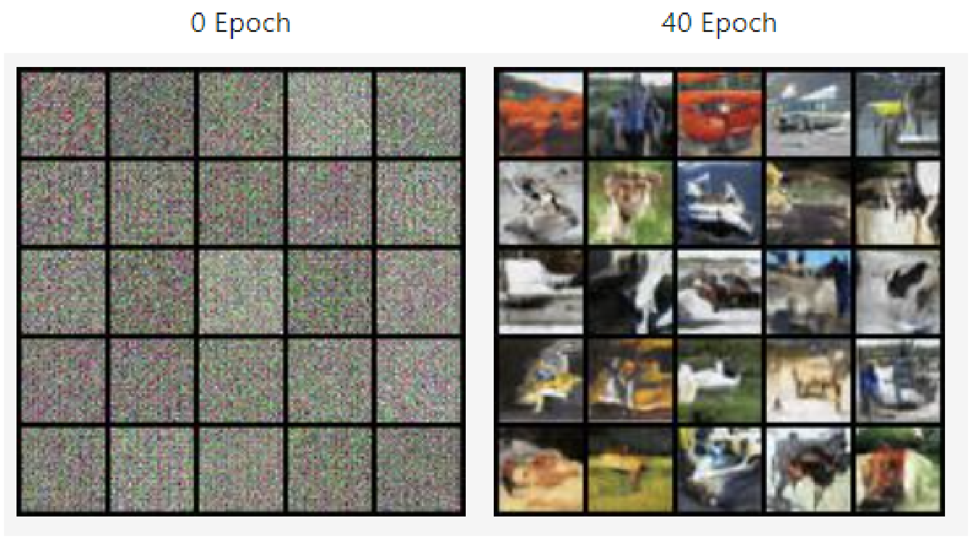


Figure 39: TransGAN Generated Examples [99]

Future investigations are numerous and essential to advancing the field. One crucial area is the development of fault-tolerant quantum hardware, which would allow for more reliable and scalable implementations of quantum GANs. Additionally, testing QRKT-GAN on various quantum architectures, including those based on trapped ions, superconducting qubits, and photonic systems, could provide deeper insights into the most effective configurations.

Further research should also explore the use of Quantum Graph Networks (QGNs) and Quantum Random Walks for optimizing network-based models. Quantum Annealing, particularly for combinatorial optimization problems, presents another promising avenue for enhancing GAN performance.

While QRKT-GAN currently faces significant limitations, ongoing advancements in Quantum Computing hardware and algorithms hold the promise of achieving Quantum Artificial Intelligence that can surpass classical methods in generating high-quality synthetic data.

6 Conclusions and Future Work

The comprehensive experiments and theoretical discussions presented in this thesis underscore the transformative potential of Quantum Computing in optimizing Deep Learning models. These results reveal that quantum techniques can significantly reduce the number of trainable parameters, thereby decreasing the energy consumption required for training such models. This is particularly crucial in domains where computational resources and energy efficiency are critical constraints.

The research primarily focused on Variational Quantum Circuits (VQCs) and Variational Quantum Algorithms (VQAs), illustrating their capacity to compress neural networks without compromising performance. These quantum methodologies offer a promising alternative to traditional Deep Learning optimization techniques, especially in scenarios demanding high efficiency and lower energy footprints.

However, the journey of integrating Quantum Computing with Artificial Intelligence is just beginning, and there are numerous avenues for future research and exploration. A pivotal area for future investigation is the development and implementation of fault-tolerant quantum hardware. The current generation of quantum devices is plagued by high error rates and issues related to decoherence, which limit their practical applications. Achieving fault tolerance in quantum hardware will be a significant milestone, enabling more reliable and extensive testing of quantum-enhanced AI models on real-world tasks.

Beyond improving hardware, future research should focus on testing these quantum models on diverse quantum architectures. Different quantum computing paradigms, such as those based on trapped ions, superconducting qubits, and photonic systems, offer unique advantages and challenges. Exploring these architectures will provide a more comprehensive understanding of how quantum computing can be leveraged to enhance AI.

Moreover, future work should include extensive benchmarking and testing of VQCs and VQAs on practical, real-world problems. This involves assessing their scalability, robustness, and generalizability across different domains. By conducting these tests, researchers can validate the practical utility of quantum-enhanced AI models and identify any limitations that need to be addressed.

Based on the insights and experimental results presented in this thesis, it is evident that Quantum Computing is poised to play a pivotal role in the future of Artificial Intelligence. The integration of quantum mechanics and AI could give rise to a stronger Quantum Artificial Intelligence, a new paradigm that leverages the unique properties of quantum systems to achieve capabilities beyond those of classical AI. This future holds the promise of more efficient, powerful, and intelligent systems capable of solving complex problems that are currently intractable.

In summary, while significant challenges remain, the potential for Quantum Computing to revolutionize AI is immense. Continued research and development in this area are essential to realize the full potential of Quantum Artificial Intelligence. By exploring diverse quantum architectures, advanced quantum techniques, and practical applications, it can be ensured that Quantum Computing becomes a cornerstone of future advancements in Artificial Intelligence.

7 Bibliography

[1] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. Generative adversarial nets. In *Advances in neural information processing systems*, pages 2672–2680, 2014.

[2] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. *Advances in neural information processing systems*, 30, 2017.

[3] Alexey Dosovitskiy, Lucas Beyer, Alexander Kolesnikov, Dirk Weissenborn, Xiaohua Zhai, Thomas Unterthiner, Mostafa Dehghani, Matthias Minderer, Georg Heigold, Sylvain Gelly, Jakob Uszkoreit, and Neil Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale. *arXiv preprint arXiv:2010.11929*, 2020.

[4] Xue-Wen Chen and Xiaotong Lin. Big data deep learning: challenges and perspectives. *IEEE access*, 2:514–525, 2014.

[5] Petro Liashchynskyi and Pavlo Liashchynskyi. Grid search, random search, genetic algorithm: a big comparison for nas. *arXiv preprint arXiv:1912.06059*, 2019.

[6] Riccardo Di Sipio, Jia-Hong Huang, Samuel Yen-Chi Chen, Stefano Mangini, and Marcel Worring. The dawn of quantum natural language processing. In *ICASSP 2022-2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, pages 8612–8616. IEEE, 2022.

[7] Zipeng Fan, Jing Zhang, Peng Zhang, Qianxi Lin, and Hui Gao. Quantum-inspired neural network with runge-kutta method. In *Proceedings of the AAAI Conference on Artificial Intelligence*, volume 38, pages 17977–17984, 2024.

[8] Geoffrey E Hinton, Alex Krizhevsky, and Sida D Wang. Transforming auto-encoders. In *Artificial Neural Networks and Machine Learning–ICANN 2011: 21st International Conference on Artificial Neural Networks, Espoo, Finland, June 14-17, 2011, Proceedings, Part I 21*, pages 44–51. Springer, 2011.

[9] Liyuan Liu, Xiaodong Liu, Jianfeng Gao, Weizhu Chen, and Jiawei Han. Understanding the difficulty of training transformers. *arXiv preprint arXiv:2004.08249*, 2020.

[10] Marco Cerezo, Andrew Arrasmith, Ryan Babbush, Simon C Benjamin, Suguru Endo, Keisuke Fujii, Jarrod R McClean, Kosuke Mitarai, Xiao Yuan, Lukasz Cincio, et al. Variational quantum algorithms. *Nature Reviews Physics*, 3(9):625–644, 2021.

[11] Elena Voita, David Talbot, Fedor Moiseev, Rico Sennrich, and Ivan Titov. Analyzing multi-head self-attention: Specialized heads do the heavy lifting, the rest can be pruned. *arXiv preprint arXiv:1905.09418*, 2019.

[12] Marius-Constantin Popescu, Valentina E Balas, Liliana Perescu-Popescu, and Nikos Mastorakis. Multilayer perceptron and neural networks. *WSEAS Transactions on Circuits and Systems*, 8(7):579–588, 2009.

[13] Marcello Benedetti, Erika Lloyd, Stefan Sack, and Mattia Fiorentini. Parameterized quantum circuits as machine learning models. *Quantum Science and Technology*, 4(4):043001, 2019.

[14] Alex Krizhevsky. Learning multiple layers of features from tiny images. Technical report, 2009.

[15] Yann LeCun, Corinna Cortes, and CJ Burges. Mnist handwritten digit database. *ATT Labs [Online]. Available: http://yann.lecun.com/exdb/mnist*, 2, 2010.

[16] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. Learning word vectors for sentiment analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*, pages 142–150, Portland, Oregon, USA, June 2011. Association for Computational Linguistics.

[17] Shi-Xin Zhang, Jonathan Allcock, Zhou-Quan Wan, Shuo Liu, Jiace Sun, Hao Yu, Xing-Han Yang, Jiezhong Qiu, Zhaofeng Ye, Yu-Qin Chen, Chee-Kong Lee, Yi-Cong Zheng, Shao-Kai Jian, Hong Yao, Chang-Yu Hsieh, and Shengyu Zhang. Tensorcircuit: a quantum software framework for the nisq era. *Quantum*, 7:912, February 2023.

[18] Jonathan Wei Zhong Lau, Kian Hwee Lim, Harshank Shrotriya, and Leong Chuan Kwek. Nisq computing: where are we and where do we go? *AAPPS bulletin*, 32(1):27, 2022.

[19] Michael A Nielsen and Isaac L Chuang. *Quantum computation and quantum information*, volume 2. Cambridge university press Cambridge, 2001.

[20] Hsin-Yuan Huang, Michael Broughton, Masoud Mohseni, Ryan Babbush, Sergio Boixo, Hartmut Neven, and Jarrod R McClean. Power of data in quantum machine learning. *Nature communications*, 12(1):2631, 2021.

[21] Yun He, Ziwei Zhu, Yin Zhang, Qin Chen, and James Caverlee. Infusing disease knowledge into bert for health question answering, medical inference and disease name recognition. *arXiv preprint arXiv:2010.03746*, 2020.

[22] Yuanzhi Li and Yang Yuan. Convergence analysis of two-layer neural networks with relu activation. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017.

[23] Jin Xu, Zishan Li, Bowen Du, Miaomiao Zhang, and Jing Liu. Reluplex made more practical: Leaky relu. In *2020 IEEE Symposium on Computers and communications (ISCC)*, pages 1–7. IEEE, 2020.

[24] Dan Hendrycks and Kevin Gimpel. Gaussian error linear units (gelus). *arXiv preprint arXiv:1606.08415*, 2016.

[25] Weiyang Liu, Yandong Wen, Zhiding Yu, and Meng Yang. Large-margin softmax loss for convolutional neural networks. *arXiv preprint arXiv:1612.02295*, 2016.

[26] Jun Han and Claudio Moraga. The influence of the sigmoid function parameters on the speed of backpropagation learning. In *International workshop on artificial neural networks*, pages 195–201. Springer, 1995.

[27] Babak Zamanlooy and Mitra Mirhassani. Efficient vlsi implementation of neural networks with hyperbolic tangent activation function. *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, 22(1):39–48, 2013.

[28] Zijun Zhang. Improved adam optimizer for deep neural networks. In *2018 IEEE/ACM 26th international symposium on quality of service (IWQoS)*, pages 1–2. Ieee, 2018.

[29] Léon Bottou. Large-scale machine learning with stochastic gradient descent. In *Proceedings of COMPSTAT'2010: 19th International Conference on Computational StatisticsParis France, August 22-27, 2010 Keynote, Invited and Contributed Papers*, pages 177–186. Springer, 2010.

[30] Long Wen, Liang Gao, Xinyu Li, and Bing Zeng. Convolutional neural network with automatic learning rate scheduler for fault classification. *IEEE Transactions on Instrumentation and Measurement*, 70:1–12, 2021.

[31] Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization. *arXiv preprint arXiv:1711.05101*, 2017.

[32] Nitish Srivastava, Geoffrey Hinton, Alex Krizhevsky, Ilya Sutskever, and Ruslan Salakhutdinov. Dropout: a simple way to prevent neural networks from overfitting. *The journal of machine learning research*, 15(1):1929–1958, 2014.

[33] Ivan Georgiev Koprinkov. The quantum superposition principle: a reconsideration, 2023.

[34] Ryszard Horodecki, Paweł Horodecki, Michał Horodecki, and Karol Horodecki. Quantum entanglement. *Reviews of Modern Physics*, 81(2):865–942, June 2009.

[35] Hai-Long Shi, Si-Yuan Liu, Xiao-Hui Wang, Wen-Li Yang, Zhan-Ying Yang, and Heng Fan. Coherence depletion in the grover quantum search algorithm. *Physical Review A*, 95(3):032307, 2017.

[36] Thomas E O'Brien, Brian Tarasinski, and Barbara M Terhal. Quantum phase estimation of multiple eigenvalues for small-scale (noisy) experiments. *New Journal of Physics*, 21(2):023022, 2019.

[37] Yaakov S Weinstein, MA Pravia, EM Fortunato, Seth Lloyd, and David G Cory. Implementation of the quantum fourier transform. *Physical review letters*, 86(9):1889, 2001.

[38] Stephan Gulde, Mark Riebe, Gavin PT Lancaster, Christoph Becher, Jürgen Eschner, Hartmut Häffner, Ferdinand Schmidt-Kaler, Isaac L Chuang, and Rainer Blatt. Implementation of the deutsch–jozsa algorithm on an ion-trap quantum computer. *Nature*, 421(6918):48–50, 2003.

[39] Abolfazl Farahani, Sahar Voghoei, Khaled Rasheed, and Hamid R. Arabnia. A brief review of domain adaptation, 2020.

[40] Geoffrey Hinton, Oriol Vinyals, and Jeff Dean. Distilling the knowledge in a neural network, 2015.

[41] Daniel Berrar et al. Cross-validation., 2019.

[42] Mohamad Zaim Awang Pon and Krishna Prakash KK. Hyperparameter tuning of deep learning models in keras. *Sparklinglight Transactions on Artificial Intelligence and Quantum Computing (STAIQC)*, 1(1):36–40, 2021.

[43] Joseph O Ogutu, Torben Schulz-Streeck, and Hans-Peter Piepho. Genomic selection using regularized linear regression models: ridge regression, lasso, elastic net and their extensions. In *BMC proceedings*, volume 6, pages 1–6. Springer, 2012.

[44] Jose M Alvarez and Mathieu Salzmann. Learning the number of neurons in deep networks, 2018.

[45] Karl Weiss, Taghi M Khoshgoftaar, and DingDing Wang. A survey of transfer learning. *Journal of Big data*, 3:1–40, 2016.

[46] Stuart Hadfield, Zhihui Wang, Bryan O'gorman, Eleanor G Rieffel, Davide Venturelli, and Rupak Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, 2019.

[47] Yaofeng Desmond Zhong, Tongtao Zhang, Amit Chakraborty, and Biswadip Dey. A neural ode interpretation of transformer layers. *arXiv preprint arXiv:2212.06011*, 2022.

[48] Nikolas P Breuckmann and Xiaotong Ni. Scalable neural network decoders for higher dimensional quantum codes. *Quantum*, 2:68, 2018.

[49] Marçal Comajoan Cara, Gopal Ramesh Dahale, Zhongtian Dong, Roy T. Forestano, Sergei Gleyzer, Daniel Justice, Kyoungchul Kong, Tom Magorsch, Konstantin T. Matchev, Katia Matcheva, and Eyup B. Unlu. Quantum vision transformers for quark-gluon classification. *Axioms*, 13(5):323, May 2024.

[50] Bei Li, Quan Du, Tao Zhou, Yi Jing, Shuhan Zhou, Xin Zeng, Tong Xiao, Jingbo Zhu, Xuebo Liu, and Min Zhang. Ode transformer: An ordinary differential equation-inspired model for sequence generation. *arXiv preprint arXiv:2203.09176*, 2022.

[51] John Charles Butcher. A history of runge-kutta methods. *Applied numerical mathematics*, 20(3):247–260, 1996.

[52] Shengyu Zhao, Zhijian Liu, Ji Lin, Jun-Yan Zhu, and Song Han. Differentiable augmentation for data-efficient gan training. In *Conference on Neural Information Processing Systems (NeurIPS)*, 2020.

[53] Yifan Jiang, Shiyu Chang, and Zhangyang Wang. Transgan: Two pure transformers can make one strong gan, and that can scale up. *Advances in Neural Information Processing Systems*, 34:14745–14758, 2021.

[54] Jean-Baptiste Cordonnier, Andreas Loukas, and Martin Jaggi. On the relationship between self-attention and convolutional layers, 2020.

[55] Alan C. Santos. O computador quântico da ibm e o ibm quantum experience. *Revista Brasileira de Ensino de Física*, 39(1), September 2016.

[56] Gil Kalai, Yosef Rinott, and Tomer Shoham. Google's quantum supremacy claim: Data, documentation, and discussion, 2023.

[57] Mariia Mykhailova. Teaching quantum computing using microsoft quantum development kit and azure quantum. In *2023 IEEE International Conference on Quantum Computing and Engineering (QCE)*. IEEE, September 2023.

[58] Justin A. Reyes, Dan C. Marinescu, and Eduardo R. Mucciolo. Simulation of quantum many-body systems on amazon cloud. *Computer Physics Communications*, 261:107750, April 2021.

[59] Vedran Dunjko and Hans J Briegel. Machine learning & artificial intelligence in the quantum domain: a review of recent progress. *Reports on Progress in Physics*, 81(7):074001, 2018.

[60] Yagnik Chatterjee, Eric Bourreau, and Marko J Rančić. Solving various np-hard problems using exponentially fewer qubits on a quantum computer. *Physical Review A*, 109(5):052441, 2024.

[61] Martin Fürer. Solving np-complete problems with quantum search. In *Latin American Symposium on Theoretical Informatics*, pages 784–792. Springer, 2008.

[62] Amira Abbas, David Sutter, Christa Zoufal, Aurélien Lucchi, Alessio Figalli, and Stefan Woerner. The power of quantum neural networks. *Nature Computational Science*, 1(6):403–409, 2021.

[63] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[64] Juyong Kim, Yookoon Park, Gunhee Kim, and Sung Ju Hwang. SplitNet: Learning to semantically split deep networks for parameter reduction and model parallelization. In Doina Precup and Yee Whye Teh, editors, *Proceedings of the 34th International Conference on Machine Learning*, volume 70 of *Proceedings of Machine Learning Research*, pages 1866–1874. PMLR, 06–11 Aug 2017.

[65] Basheer Qolomany, Majdi Maabreh, Ala Al-Fuqaha, Ajay Gupta, and Driss Benhaddou. Parameters optimization of deep learning models using particle swarm optimization. In *2017 13th International Wireless Communications and Mobile Computing Conference (IWCMC)*, pages 1285–1290, 2017.

[66] Chai Wah Wu. Prodsumnet: reducing model parameters in deep neural networks via product-of-sums matrix decompositions. *arXiv preprint arXiv:1809.02209*, 2018.

[67] Alireza Azadbakht, Saeed Reza Kheradpisheh, Ismail Khalfaoui-Hassani, and Timothée Masquelier. Drastically reducing the number of trainable parameters in deep cnns by inter-layer kernel-sharing. *arXiv preprint arXiv:2210.14151*, 2022.

[68] Adriano Barenco, Charles H Bennett, Richard Cleve, David P DiVincenzo, Norman Margolus, Peter Shor, Tycho Sleator, John A Smolin, and Harald Weinfurter. Elementary gates for quantum computation. *Physical review A*, 52(5):3457, 1995.

[69] Suhas Maddali. A brief overview of machine learning. `https://towardsdatascience.com/a-brief-overview-of-machine-learning-20abc68cbd4e`, 2022.

[70] Robert Kwiatkowski. Gradient descent algorithm. `https://towardsdatascience.com/gradient-descent-algorithm-a-deep-dive-cf04e8115f21`, 2021.

[71] Léon Bottou. Stochastic gradient descent tricks. In *Neural Networks: Tricks of the Trade: Second Edition*, pages 421–436. Springer, 2012.

[72] Fabio M. Graetz. Why adamw matters. `https://towardsdatascience.com/why-adamw-matters-736223f31b5d`, 2018.

[73] Anshu Mishra. Deep learning fundamental. `https://medium.datadriveninvestor.com/deep-learning-fundamental-important-concepts-59d7ae90901b`, 2019.

[74] Morjina akter. Artificial neural network(ann). `https://medium.com/@morjina/artificial-neural-network-ann-74eae97980ea`, 2023.

[75] baeldung. Multi-layer perceptron vs. deep neural network. `https://www.baeldung.com/cs/mlp-vs-dnn`, 2023.

[76] Marco Del Pra. Generative adversarial networks. `https://medium.com/@marcodelpra/generative-adversarial-networks-dba10e1b4424`, 2023.

[77] BRYON MOYER. Generative adversarial network (gan). `https://semiengineering.com/knowledge_centers/artificial-intelligence/neural-networks/generative-adversarial-network-gan/`, 2021.

[78] Tauseef Ahmad. Vision transformers. `https://medium.com/@tauseefahmad12/vision-transformers-4c6116f7f0ce`, 2024.

[79] Shreenidhi Sudhakar. Learning rate scheduler. `https://towardsdatascience.com/learning-rate-scheduler-d8a55747dd90`, 2017.

[80] Cloud Factory. Learning rate scheduler - overview. https://wiki.cloudfactory.com/docs/mp-wiki/scheduler/overview-of-learning-rate-schedulers-in-ml, 2024.

[81] Sujatha Mudadla. Weight decay in deep learning. https://medium.com/@sujathamudadla1213/weight-decay-in-deep-learning-8fb8b5dd825c, 2023.

[82] Mohammed Alhamid. What is cross-validation? https://towardsdatascience.com/what-is-cross-validation-60c01f9d9e75, 2020.

[83] M. Hammad Hassan. Random search. https://medium.com/@hammad.ai/tuning-model-hyperparameters-with-random-search-f4c1cc88f528, 2023.

[84] Binoy. Random search in machine learning. https://www.scaler.com/topics/machine-learning/random-search-in-machine-learning/, 2023.

[85] Vahid Shahrezaei. Ordinary differential equations. https://bookdown.org/vshahrez/lecture-notes/introduction-to-ordinary-differential-equations.html, 2021.

[86] Yiping Lu, Zhuohan Li, Di He, Zhiqing Sun, Bin Dong, Tao Qin, Liwei Wang, and Tie-Yan Liu. Understanding and improving transformer from a multi-particle dynamic system point of view. *arXiv preprint arXiv:1906.02762*, 2019.

[87] Subhabrata Dutta, Tanya Gautam, Soumen Chakrabarti, and Tanmoy Chakraborty. Redesigning the transformer architecture with insights from multi-particle dynamical systems. *Advances in Neural Information Processing Systems*, 34:5531–5544, 2021.

[88] Patrick Rebentrost, Masoud Mohseni, and Seth Lloyd. Quantum support vector machine for big data classification. *Physical review letters*, 113(13):130503, 2014.

[89] Nathan Wiebe, Ashish Kapoor, and Krysta M Svore. Quantum nearest-neighbor algorithms for machine learning. *Quantum information and computation*, 15(3-4):318–358, 2015.

[90] Sonika Johri, Shantanu Debnath, Avinash Mocherla, Alexandros Singk, Anupam Prakash, Jungsang Kim, and Iordanis Kerenidis. Nearest centroid classification on a trapped ion quantum computer. *npj Quantum Information*, 7(1):122, 2021.

[91] Ajit Narayanan and Tammy Menneer. Quantum artificial neural network architectures and components. *Information Sciences*, 128(3-4):231–255, 2000.

[92] Weihua Hu, Muhammed Shuaibi, Abhishek Das, Siddharth Goyal, Anuroop Sriram, Jure Leskovec, Devi Parikh, and C Lawrence Zitnick. Forcenet: A graph neural network for large-scale quantum calculations. *arXiv preprint arXiv:2103.01436*, 2021.

[93] Yunfei Wang and Junyu Liu. A comprehensive review of quantum machine learning: from nisq to fault tolerance, 2024.

[94] James Bradbury, Roy Frostig, Peter Hawkins, Matthew James Johnson, Chris Leary, Dougal Maclaurin, George Necula, Adam Paszke, Jake VanderPlas, Skye Wanderman-Milne, et al. Jax: composable transformations of python+ numpy programs. 2018.

[95] Jonathan Heek, Anselm Levskaya, Avital Oliver, Marvin Ritter, Bertrand Rondepierre, Andreas Steiner, and Marc van Zee. Flax: A neural network library and ecosystem for jax. *Version 0.3*, 3:14–26, 2020.

[96] Michael Broughton, Guillaume Verdon, Trevor McCourt, Antonio J Martinez, Jae Hyeon Yoo, Sergei V Isakov, Philip Massey, Ramin Halavati, Murphy Yuezhen Niu, Alexander Zlokapa, et al. Tensorflow quantum: A software framework for quantum machine learning. *arXiv preprint arXiv:2003.02989*, 2020.

[97] Andrew Cross. The ibm q experience and qiskit open-source quantum computing software. In *APS March meeting abstracts*, volume 2018, pages L58–003, 2018.

[98] Sagar Imambi, Kolla Bhanu Prakash, and GR Kanagachidambaresan. Pytorch. *Programming with TensorFlow: Solution for Edge Computing Applications*, pages 87–104, 2021.

[99] Ahmet Sarıgün. Re-Implementation of TransGAN: Two Transformers Can Make One Strong GAN, March 2024.