University of Western Australia

Data Science Research Project

Quantum Transformers: From Circuit Design to Performance Comparison

Wei Chun Nicholas Choong

Supervised by
Assoc. Prof. Wei Liu
Assoc. Prof. Du Huynh
Prof. Jingbo Wang
Prof. Fredrick Cadet

October 2024

Abstract

This dissertation investigates the integration of quantum variational circuits (VQCs) into transformer models to explore the potential advantages of quantum computing in natural language processing (NLP) tasks. By incorporating both basic and strong VQCs, along with encoding methods such as amplitude and angle encoding, we compare the performance of quantum transformers to classical transformers. Extensive experiments were conducted using datasets like IMDb, Amazon, and Yelp to evaluate models with and without embedding layers. The results show that quantum models, particularly those using amplitude encoding, can outperform classical models in certain configurations, achieving superior accuracy. Additionally, we have successfully reduced the training time by leveraging efficient quantum computing frameworks. This work demonstrates the viability of quantum machine learning models and provides insights into their potential applications in complex NLP tasks. The study also discusses future directions for improving quantum transformer architectures, including further optimisations in encoding strategies and circuit designs.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivations

Over the past few decades, the exponential growth of deep learning has led to groundbreaking advancements that have not only transformed computer science and permeated numerous other disciplines and industries. The profound impact of deep learning is evident in its integration into various aspects of daily life—from enhancing virtual assistants and powering autonomous vehicles to revolutionising healthcare diagnostics and financial analytics. Deep learning models have achieved superhuman performance in tasks such as computer vision, natural language processing (NLP), and generative artificial intelligence (AI), enabling the creation of human-like outputs across multiple media, including text, images, audio, and video.

This influence was further emphasised in 2024 when John J. Hopfield and Geoffrey E. Hinton were jointly awarded the Nobel Prize in Physics for their seminal contributions to artificial neural networks. Hopfield's pioneering work on associative memory models and Hinton's revolutionisation of the backpropagation algorithm laid the foundational architectures of modern deep learning. The recognition by the Nobel Committee not only highlights the interdisciplinary nature of their work but also cements the significance of deep learning in advancing scientific and technological progress.

Despite these remarkable achievements, the rapid evolution of deep learning has brought several limitations to the forefront, particularly concerning the computational and financial resources required to train increasingly complex models. As model architectures grow in depth and breadth, the demand for vast amounts of data and processing power escalates, presenting significant barriers to scalability and broader application. Classical hardware, while powerful, faces inherent limitations in handling the ever-growing demands of cutting-edge AI models. This is especially evident in emerging fields like multimodal learning, artificial general intelligence, and protein folding, where the sheer scale of computations challenges even the most advanced classical systems.

In response to these challenges, quantum computing has emerged as a promising paradigm

that could potentially overcome the limitations of classical computing. By leveraging the principles of quantum mechanics, quantum computing offers the potential to process information in fundamentally new ways, providing exponential speedups for certain tasks. Quantum phenomena such as superposition and entanglement allow quantum computers to explore vast computational spaces more efficiently than classical bits, opening up new avenues for tackling problems previously considered intractable. The implications of quantum computing extend across various sectors, including cryptography, optimisation, and artificial intelligence.

At the intersection of deep learning and quantum computing lies the potential for a new class of models: quantum deep learning models, particularly quantum transformers. Inspired by classical transformer models—which have revolutionised NLP, computer vision, and generative AI—these quantum counterparts aim to enhance capabilities by exploiting the unique properties of qubits, such as superposition, entanglement, and quantum parallelism. Quantum transformers hold the promise of processing information more efficiently, enabling more scalable and resource-efficient models. This presents exciting possibilities for advancing tasks like language translation, image generation, and even more complex applications that push the boundaries of AI.

Recent developments have seen claims that quantum transformers can outperform classical models in specific scenarios, achieving comparable or superior results with fewer resources. For instance, the work of Cherrat et al. [2] suggests that quantum vision transformers can attain accuracies that meet or exceed those of classical vision transformers. However, these claims face significant challenges. Quantum hardware remains in developmental stages, making the implementation of quantum algorithms complex and resource-intensive. Additionally, training quantum neural networks, including quantum transformers, often encounters the problem of barren plateaus—regions in the optimisation landscape where gradients vanish, hindering effective training.

Moreover, the theoretical and practical aspects of integrating quantum computing with deep learning are still under active research. Issues such as error correction in quantum systems, decoherence, and the scalability of quantum circuits pose substantial hurdles. The quantum machine learning community is actively exploring algorithms and architectures that can mitigate these problems, aiming to unlock the full potential of quantum-enhanced AI.

The convergence of quantum computing and deep learning represents a frontier with immense potential but also significant uncertainty. As both fields continue to evolve, their intersection could lead to breakthroughs that redefine computational capabilities and transform industries. Understanding the current landscape, the challenges, and the opportunities is crucial for advancing this emerging area of research.

The integration of quantum mechanics into deep learning models also raises philosophical and ethical considerations. The prospect of machines that can process information in fundamentally new ways prompts questions about the future of AI, the nature of intelligence, and the potential societal impacts. There is a growing discourse on how quantum-enhanced AI could influence

areas such as data privacy, security, and employment, highlighting the need for interdisciplinary collaboration in addressing these concerns.

Historically, technological advancements have often led to paradigm shifts in society. The steam engine ignited the Industrial Revolution, electricity transformed daily life and industry, and the internet revolutionised communication and information access. Similarly, the fusion of quantum computing and AI could bring forth a new era of innovation. Industries such as pharmaceuticals might see accelerated drug discovery processes, finance could benefit from more sophisticated modeling and risk assessment, and logistics could achieve unprecedented optimisation levels.

In summary, the advent of quantum computing offers a promising avenue to address the limitations faced by classical deep learning models. Quantum transformers, by harnessing the unique properties of quantum mechanics, could potentially revolutionise AI by enabling more efficient and powerful models. As we stand at the cusp of this new era, exploring the interplay between quantum computing and deep learning is not only intriguing but also essential for the future of computational science.

The journey toward realising the full potential of quantum transformers is fraught with challenges but also rich with opportunities. Continued research and innovation in this field could lead to significant breakthroughs that redefine the limits of machine learning and computation. The implications of such advancements are vast, promising transformative applications across industries and disciplines, and ushering in a new chapter in the story of technological progress.

1.2 Research Objectives

The convergence of quantum computing and deep learning presents a frontier ripe with potential for significant advancements in artificial intelligence. This research seeks to explore this intersection by focusing on quantum transformers. The primary objectives of this study are outlined as follows:

1. **Design and Optimise New Quantum Circuits**

   The goal is to explore and implement various quantum circuit designs for use in quantum transformers. This includes investigating different data encoding methods such as amplitude encoding, angle encoding, and block encoding to effectively represent input data within quantum circuits. We will employ Pennylane's basic and strong entangling layers for constructing VQCs to enhance the expressibility and entangling capabilities of the quantum circuits. Additionally, we will use an autoencoder to adjust the dimension of the word embedding to fit into the next layer, ensuring efficient data flow through the quantum and classical components.

2. **Compare Quantum and Classical Models and Attempt to Reproduce Existing Results**

   To conduct a thorough comparison between quantum and classical transformer models, focusing on accuracy and computational resources. This includes an effort to reproduce results from recent studies to assess the performance and potential benefits of quantum circuits in these models.

3. **Reduce Quantum Model Training Time on Simulated Quantum Computers**

   To explore different quantum computing frameworks, such as Pennylane and TensorCircuit, as well as classical deep learning frameworks like PyTorch and TensorFlow. The goal is to optimise the training process, making quantum transformer models more manageable to be trained on simulated quantum computers, potentially reducing the overall training time and making them more feasible for practical applications.

By pursuing these objectives, our research aims to contribute to the foundational understanding and practical advancement of quantum transformers. This work seeks to explore and demonstrate the capabilities and limitations of integrating quantum computing with deep learning, ultimately contributing to the evolution of artificial intelligence in the quantum era.

1.3 Our Contributions

This research makes several key contributions to the integration and optimisation of quantum transformer models. First, we designed and implemented VQCs within transformer architectures, exploring both basic and strong VQC layers. By using these quantum circuits to process classical word embeddings into quantum states, we experimented with different encoding strategies, such as amplitude encoding and angle encoding. This work provides valuable insights into how quantum circuits can be effectively integrated with classical transformer models, showcasing the potential of quantum-enhanced natural language processing.

Another important contribution of this work is the extensive comparison between quantum and classical transformer models. Through detailed experimentation, we demonstrated that quantum models, particularly those employing amplitude encoding, can outperform classical models in specific tasks. This finding not only aligns with existing research but also provides compelling evidence that, when properly optimised, quantum architectures can exceed the performance of traditional classical transformers, highlighting their potential in machine learning applications.

Finally, a significant focus of this study was on reducing the training time for quantum models, which is often a challenge in practical applications. By adopting the TensorCircuit framework with TensorFlow CUDA, we were able to drastically reduce training times from

several hours per epoch to just minutes. These optimisations make quantum models more feasible for large-scale tasks, furthering the practical application of quantum computing in machine learning.

1.3.1 Open Source Code Contributions

As part of this research, we have contributed to the open-source community by making our code publicly available. The code for the quantum transformers, including the design and implementation of quantum circuits, is hosted on GitHub. This repository provides a comprehensive set of tools and resources for researcherss interested in quantum machine learning and transformer-based models. Our code repository can be found at https://github.com/NicholasChoong/QuantumTransformer.

1.4 Outline

This dissertation is structured as follows:

1. **Chapter 2: Literature Review**

   This chapter presents a comprehensive review of the existing literature on transformer models, quantum machine learning, quantum neural networks, and variational quantum circuits. It also discusses various data embedding techniques and the limitations of current models, providing a foundation for the development of quantum transformers.

2. **Chapter 3: Methodology**

   The methodology outlines the research design, data collection methods, and experimental setup. Furthermore, it describes the implementation of both classical and quantum transformer models, including the specific architectures, encoding techniques, and optimisations employed. The chapter concludes with a description of the evaluation metrics used to compare the performance of the model.

3. **Chapter 4: Experiments and Results**

   In this chapter, the experiments conducted on both classical and quantum transformers are presented. The results of these experiments, including the performance metrics, are thoroughly analysed. Comparisons between the classical and quantum models are made, with a focus on the effectiveness of different quantum circuits, encoding methods, and training optimisations. The chapter also addresses the primary research questions posed in this study.

4. **Chapter 5: Conclusion and Future Work**

The conclusion summarises the key findings of the research, reflecting on the contributions made towards the development and optimisation of quantum transformer models. The chapter also outlines potential directions for future research.

5. **Appendices**

   The appendices include supplementary materials such as the research proposal and user manual.

Chapter 2

Literature Review

2.1 Background

2.1.1 Transformer Models

Transformer architectures have emerged as a dominant paradigm in natural language processing. Unlike traditional recurrent neural networks and convolutional neural networks, transformers avoid sequential processing in favour of self-attention mechanisms that allow for parallel processing of input sequences.

The key innovation of transformers lies in their ability to capture long-range dependencies within input sequences, making them particularly effective for tasks such as language translation, text generation, and sentiment analysis. Models such as Bidirectional Encoder Representations from Transformers (BERT) and Generative Pre-trained Transformer (GPT) have achieved state-of-the-art performance on various natural language processing benchmarks, demonstrating the power of transformer architectures.

2.1.2 Quantum-Inspired Machine Learning

Quantum-Inspired Machine Learning (QiML) refers to a class of machine learning algorithms that draw inspiration from quantum mechanics but are implemented on classical hardware. Unlike Quantum Machine Learning (QML), which relies on quantum computation to process data, QiML applies quantum principles to classical algorithms without the need for quantum devices. QiML has gained attention due to its potential to harness computational advantages by simulating quantum effects through classical frameworks. Techniques such as tensor networks, dequantized algorithms, and quantum-inspired optimisation algorithms are prominent examples of QiML approaches. These methods have been shown to improve efficiency and performance on certain tasks traditionally solved using classical machine learning models.

The relationship between QiML and QML can be understood as a spectrum of quantum

mechanics integration, as illustrated in Figure 2.1. On the left side of the spectrum, we have classical approaches like dequantized algorithms that employ quantum-inspired techniques but operate entirely within classical systems. As we move to the right, more quantum mechanics are integrated, including tensor networks and density matrices used as features, which bridge classical and quantum methods. Finally, on the far right, we encounter true QML approaches, such as VQCs and quantum kernels, which require quantum hardware for computation.

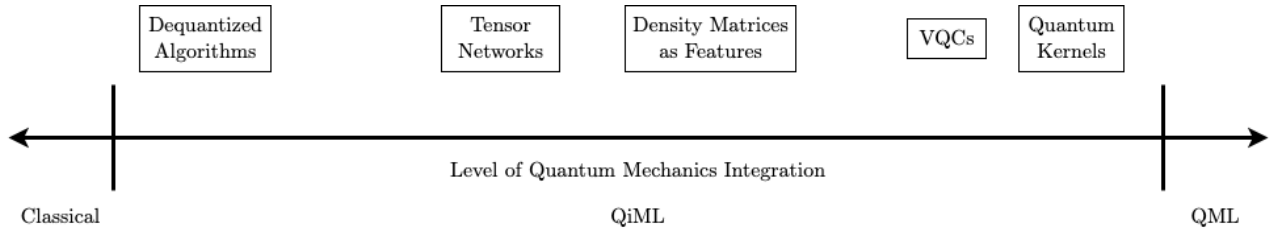


Figure 2.1: A spectrum of machine learning approaches from classical to quantum-inspired to quantum-based methods [6].

The primary difference between QML and QiML lies in their reliance on quantum hardware. While QML explores the use of quantum computers to solve machine learning problems, QiML remains entirely within the classical computational realm, applying quantum concepts without leveraging quantum hardware. This distinction makes QiML more accessible given current technological limitations in quantum computing.

It is important to note that our research focuses on QML, not QiML. In this project, we will directly employ simulated quantum hardware and algorithms to explore how quantum computing can enhance machine learning, specifically for sentiment analysis tasks using transformer models.

2.1.3 Quantum Neural Network

A Quantum Neural Network (QNN) is a machine learning architecture that employs quantum computing principles. It has a set of trainable parameters (θ) that can be realised based on an initial probability distribution. The goal of machine learning is to train these parameters and achieve a probability distribution that closely resembles the underlying problem. A model with high generalisation ability can identify existing patterns in the testing data without overfitting to the training data.

The specific approach involves providing the data to the quantum model through a process called a data embedding strategy, which can significantly impact the functional representation of the model. The objective function is the expectation value of a variational quantum circuit. These circuits make use of continuous variable group rotations, allowing for the manipulation of quantum states.

Since there is no established framework for designing variational quantum circuits, it is crucial to fine-tune the circuit architecture, as it can directly affect the performance of the

quantum neural network model. Fortunately, many quantum computing libraries, such as Pennylane and Qiskit, provide templates for VQCs.

2.1.4 Data Embedding

There are many encoding strategies for loading classical data into a quantum computer but for this literature, we will only be discussing about three encoding strategies. They are Angle embedding, Amplitude embedding, Quantum Approximate Optimization Algorithm (QAOA) embedding and Block Encoding.

- Angle embedding [11] is a straightforward strategy where classical data features are encoded into the angles of quantum gates. Each feature is mapped to the rotation angle of a qubit, typically using single-qubit rotation gates such as RX, RY, or RZ. To encode n features, Angle embedding requires n qubits, as each feature is directly assigned to a qubit's rotation. While this encoding method is straightforward, it can be resource-intensive for large datasets, as the number of qubits required grows rapidly with the number of features.

- Amplitude embedding [11] is a more complex strategy that encodes classical data into the amplitudes of a quantum state. This method involves preparing a quantum state where the amplitudes of the state vector represent the classical data values. Given a classical data vector $x = [x_1, x_2, \ldots, x_n]$, the first step is to normalise the data. The normalisation is done by dividing each element by the Euclidean norm $\|x\|$, where:

$$\|x\| = \sqrt{x_1^2 + x_2^2 + \cdots + x_n^2} \tag{2.1}$$

  This ensures the total probability of the quantum state is 1. The normalised vector is:

$$\tilde{x} = \frac{1}{\|x\|}[x_1, x_2, \ldots, x_n] \tag{2.2}$$

  Next, the normalised vector $\tilde{x}$ is encoded into the amplitudes of a quantum state $|\psi\rangle$, which can be represented as:

$$|\psi\rangle = \sum_{i=1}^{n} \tilde{x}_i |i\rangle \tag{2.3}$$

  where $|i\rangle$ are the computational basis states. To encode n features, $\log_2(n)$ qubits are required, assuming n is a power of two, because the quantum state must accommodate all n amplitudes, and $\log_2(n)$ qubits are needed to create a state with n possible amplitudes. For example, if $n = 4$, the resulting quantum state would be:

$$|\psi\rangle = \tilde{x}_1|00\rangle + \tilde{x}_2|01\rangle + \tilde{x}_3|10\rangle + \tilde{x}_4|11\rangle \tag{2.4}$$

This encoding strategy is particularly useful for applications that involve processing high-dimensional data, as it can significantly reduce the number of qubits required compared to other encoding methods.

- The Quantum Approximate Optimization Algorithm (QAOA) embedding [8] is a hybrid approach that encodes classical data into the parameters of a QAOA circuit. QAOA is originally designed for solving combinatorial optimisation problems but can be adapted for data encoding by associating the problem parameters with data features. The number of qubits required for QAOA embedding depends on the specific problem and the depth of the QAOA circuit but generally requires at least as many qubits as there are features to encode the data adequately. For n features, a minimum of n qubits is typically needed, with additional qubits possibly required depending on the circuit's complexity and the problem structure. In general, QAOA embedding requires several qubits that scale with the number of variables and constraints in the optimisation problem. This encoding strategy is particularly useful for solving combinatorial optimisation problems on quantum computers.

- Block encoding embeds a non-unitary operator as a sub-block of a larger unitary matrix, allowing non-unitary matrices to be processed within quantum circuits. In general, a matrix $A \in \mathbb{C}^{N\times N}$, where $N = 2^n$, can be block-encoded into a unitary matrix by extending the dimension of the matrix and adding ancilla qubits.

  The block-encoded matrix U can be represented as:

$$U = \begin{pmatrix} A & * \\ * & * \end{pmatrix} \tag{2.5}$$

  The block-encoded form allows us to work with non-unitary operators in a quantum framework while maintaining the overall unitary evolution required by quantum mechanics. To perform block encoding, we use a combination of quantum oracles U_A and U_B. The oracle U_A encodes the matrix elements $A_{i,j}$ into the amplitude of an ancillary qubit, while U_B ensures proper indexing over the matrix entries. The combination of these operations allows the matrix A to be embedded as a block within a larger unitary matrix. This technique is particularly useful for quantum algorithms that involve manipulating large, structured, or sparse matrices, as it efficiently encodes the matrix into a quantum state without directly applying non-unitary operations. Block encoding is crucial for several quantum machine learning and linear algebra algorithms, enabling the practical implementation of otherwise challenging quantum computations.

2.1.5 Variational Quantum Circuit

In classical neural networks, increasing the number of parameters enhances the expressivity of the models. However, in quantum circuits, having too many parameters can lead to redundancy because over-parameterised circuits may not improve performance and can make optimisation more challenging. Excess parameters can result in entangled states that do not contribute meaningfully to the solution, potentially causing the circuit to explore a larger, less relevant portion of the parameter space. As mentioned earlier, there are no standard frameworks for designing these architectures, resulting in varying designs from one author to another. Nonetheless, quantum computing libraries offer templates for VQCs to facilitate their use.

$$U(\theta) = \prod_{i=1}^{L} U_i(\theta_i) \tag{2.6}$$

where L is the number of VQC iterations, U is the VQC unitary, $\theta = (\theta_1, ..., \theta_L)$ is a set of trainable parameters.

The variational quantum circuit [7] consists of repeated rotation gates applied to each qubit, along with CNOT gates that entangle the qubits to form a highly entangled system. These rotation gates, such as RX, RY, or RZ, adjust the qubit states based on the trainable parameters. The CNOT gates, on the other hand, create entanglement between pairs of qubits, enabling complex interactions across the entire quantum system.

This combination of rotation and entangling gates allows the circuit to explore a wide range of quantum states, enhancing its ability to model complex functions and relationships within the data. By fine-tuning the parameters of these gates, the quantum circuit can be optimised to solve specific problems or recognise patterns in data. However, designing these circuits requires careful consideration, as the arrangement and number of gates can significantly impact the circuit's performance and computational efficiency.

2.1.6 Gradient Calculation

The most common gradient calculation method in classical machine learning is the back-propagation algorithm. This technique computes the gradient of each function that the trainable parameters pass through, using the chain rule to create an automatically differentiable machine learning routine. In the context of quantum machine learning, back-propagation can also be employed. However, there's a key difference: it requires access to the quantum state vector. As a result, its application is limited to quantum simulators rather than real quantum processing units (QPUs).

As a result, it is crucial to find alternative methods that can operate on actual QPUs. One such alternative is the finite difference method [12], a form of numerical differentiation used to approximate derivatives. The finite difference method estimates the derivative of a function

by evaluating the function at slightly shifted parameter values and calculating the ratio of the change in the function value to the change in the parameter.

However, while finite difference provides a useful approximation, it can be quite unstable when used iteratively in processes such as gradient descent. This instability arises because the method is sensitive to numerical errors, which can accumulate over multiple iterations, leading to inaccurate gradient estimates. Additionally, near-term quantum hardware is inherently noisy, which further exacerbates the inaccuracy of finite differences, making it unreliable for precise optimisation tasks.

Another approach is the parameter shift rule [12], which provides an exact derivative by evaluating the circuit twice for each trainable parameter. Unlike finite difference methods, it does not require a small perturbation but instead can use a macroscopic shift. By optimising the shift to maximise the distance in parameter space between the two circuit evaluations, the parameter shift rule offers a robust and precise gradient estimation.

The parameter shift rule involves shifting the parameter by a fixed amount in two directions and using the difference in the circuit's outputs to calculate the gradient. Specifically, for a parameter θ, the gradient $\frac{\partial f(\theta)}{\partial \theta}$ is obtained by evaluating the function $f(\theta + s)$ and $f(\theta - s)$ where s is the shift value.

The exact derivative is then computed as:

$$\frac{\partial f(\theta)}{\partial \theta} = \frac{f(\theta + s) - f(\theta - s))}{2sin(s)} \tag{2.7}$$

This method is advantageous because it provides an unbiased estimator of the gradient, ensuring convergence even when the gradient is estimated with a single shot. In contrast, finite difference methods approximate the derivative by evaluating the function at slightly perturbed values of the parameter, which can be unstable and sensitive to noise, especially in iterative processes like gradient descent. Finite differences rely on a small perturbation to estimate the gradient, which can accumulate numerical errors over multiple iterations, leading to less accurate results.

However, the parameter shift rule does pose challenges. The number of circuit evaluations required increases linearly with the number of trainable parameters, which can limit the complexity of quantum models. As the number of parameters grows, the computational resources needed for gradient estimation also increase, potentially making the process resource-intensive.

2.1.7 Hybrid Classical-Quantum Autoencoder

The variational quantum circuit uses angle embedding, where the input dimension is restricted to a certain size due to the limitations of the available qubits. This restriction limits the quantum circuit's ability to process larger input dimensions directly.

To address this limitation, a classical autoencoder is integrated into the model, comprising an encoder and a decoder:

- The encoder acts as a compression mechanism, performing the squeezing of high-dimensional input data into a smaller latent space that fits the fixed input dimension of the quantum circuit. This is crucial for ensuring that the data can be processed within the qubit limitations of the VQC.
- The decoder performs the reverse operation, acting as an unsqueezing mechanism. After the quantum circuit processes the compressed data, the decoder expands the quantum-processed data back to its original size or another appropriate dimension for subsequent layers in the network.

By using this autoencoder structure, the model effectively bridges the dimensional gap between high-dimensional classical data and the qubit-limited quantum circuit. The encoder enables dimensionality reduction while maintaining key features, and the decoder restores the data to a useful size for further processing. This combination allows the quantum circuit to be utilised efficiently within the overall model without losing the ability to work with larger datasets.

2.2 Quantum Transformers

2.2.1 Basic Quantum Transformer

In 2021, Sipio et al. [14] proposed a Quantum Transformer that replaces 5 linear layers with 5 Quantum layers which are ansatzes, 4 of which are in the multi-head attention, and 1 of which is in the feed-forward network. The quantum layers use the basic entangler layer template provided by PennyLane.

A variational quantum circuit cannot change the dimensionality of the input but only rotate the state of the qubits, so the VQC is sandwiched between 2 linear layers. A linear layer is used to compress the dimension of the input to match the number of qubits and another linear layer is used to uncompress the dimension of the output to match the original dimension of the input so it can be passed to the next layer.

This paper presents a conflicting scenario where the embedding dimension and the number of qubits are mismatched. This incongruity poses a significant challenge because the published code mandates that the embedding dimension and the number of qubits align for the multi-head attention mechanism to function correctly.

The parameters for training are as follows: 1 epoch, a batch size of 32, a vocabulary size of 50,000, a maximum sequence length of 64, an embedding dimension of 8, 1 transformer block, 2

transformer heads (the number of attention mechanisms in the multi-head attention component), 1 quantum layer, and 2 qubits. The dataset used for training and testing is the IMDB dataset, which consists of movie reviews labelled as either positive or negative. Both the training and testing sets contain 25,000 reviews each.

The author did not publish the results of the quantum transformer, only mentioning that it took 100 hours to train the classifier for a single epoch. I replicated the transformer and tested it on a subset of the dataset, consisting of just 3,200 reviews. My results showed that after 40 epochs, the training accuracy was 73% and the testing accuracy was 63%. However, the model appeared to be overfitted, as indicated by the train and test loss. More fine-tuning is needed to increase the accuracy and prevent overfitting.

As this is one of the first papers on Quantum Transformers, I assume that the author employed fundamental quantum machine learning techniques to replace one or two components from classical to quantum. By focusing on basic modifications, this entry-level code provides a clear and accessible starting point for understanding the process of converting Transformer components. This incremental approach allows for a step-by-step conversion, making it easier to grasp how quantum machine learning can be applied iteratively to transition from classical to quantum models. This understanding is crucial for developing more advanced quantum models in the future, as it lays the groundwork for integrating quantum computing into existing machine learning frameworks.

2.2.2 Basic Quantum Vision Transformer

In 2023, Comajoan Cara et al. [4] adapted Sipio et al. [14]'s basic quantum transformer into a basic quantum vision transformer. Comajoan Cara et al. [4] used 3 datasets, MNIST dataset, Quark-Gluon dataset, and Electron-Photon dataset.

- MNIST dataset contains 70,000 1-channel 28x28 images of handwritten digits, which are labelled with the corresponding digit. The dataset is split into 60,000 images for training and 10,000 images for testing.

- Quark-Gluon consists of 933,206 3-channel 125x125 images, with half representing quarks and the other half gluons. Each of the three channels in the images corresponds to a specific component of the Compact Muon Solenoid (CMS) detector of the LHC: the inner tracking system (Tracks) that identifies charged particle tracks, the electromagnetic calorimeter (ECAL) that captures energy deposits from electromagnetic particles, and the hadronic calorimeter (HCAL) which detects energy deposits from hadrons.

- Electron-Photon contains 498,000 2-channel 32x32 images, with half representing electrons and the other half representing photons. Here, only information from the CMS electromagnetic calorimeter (ECAL) is used. In particular, the first channel contains

energy information (as in the Quark-Gluon dataset), and the second one contains timing information.

To convert the quantum transformer for text classification into a quantum vision transformer for image classification, several modifications are necessary. One major change involves splitting the image into n patches and passing these patches through a linear layer to flatten them before feeding them into the transformer. Another significant change is appending the class tag to the very front of the patches so that the multi-layer perceptron can use it for classification.

The parameters for training the quantum vision transformer are as follows: 50 epochs, a patch size of 32, an embedding size of 8, 8 transformer blocks, 2 transformer heads, and 8 qubits. The training results are as follows:

- For the MNIST dataset, the classical transformer achieved 99.71% accuracy, while the quantum transformer achieved 98.94%.
- For the Quark-Gluon dataset, the classical transformer achieved 79.76% accuracy, while the quantum transformer achieved 77.62%.
- For the Electron-Photon dataset, the classical transformer achieved 76.50% accuracy, while the quantum transformer achieved 77.93%.

From these results, we can see that the classical transformer outperformed the quantum transformer on two of the datasets. For the third dataset, the difference in performance between the classical and quantum transformers is negligible.

Comajoan Cara et al. [4] was aware that Sipio et al. [14]'s Quantum Transformer requires a long training time, so they experimented with different libraries to compare training durations. After testing various options, they found that the library utilising tensor networks was the fastest. This observation highlights that the choice of quantum machine learning library can significantly influence not only the training time but also the performance of the models. Different libraries may offer optimisations, algorithms, and implementations that impact efficiency and accuracy. Therefore, selecting the appropriate library is crucial for achieving optimal results in quantum machine learning experiments and applications. This insight suggests that the performance of the Quantum Transformer could be further improved by using tensor networks.

2.2.3 Quantum Self Attention Neural Network

In 2023, Li et al. [7] proposed a Quantum Self Attention Neural Network for classification. This implementation has 2 parts. The first part is in the quantum realm and the second part is in the classical realm. The input is x, but this paper did not use any positional encoding. The Quantum Self-Attention mechanisms are connected sequentially and they are called the Quantum Self-Attention Layer. Each layer has a set of four ansatzes. The outputs of the final

layer are averaged and fed into the second part which is a classical fully connected layer for classification.

In the Quantum Self-Attention Layer, the embedding dimension of each of the inputs has the same size as the number of qubits as each element in the embedding is mapped to its corresponding qubits in the ansatzes to encode the embeddings into their corresponding quantum states. Then, a set of three ansatzes representing query, key, and value is applied to each state. Note that it is the same set of ansatzes applied to all the input states.

The measurement outputs of the query z-rotation part and the key z-rotation part are computed through a Gaussian function to obtain the quantum self-attention coefficients. We then calculate classically weighted sums of the measurement outputs of the value and add the inputs to get the outputs of the current layer where the weights are the normalised coefficient.

All the ansatz parameters and weight are initialised from a Gaussian distribution with zero mean and 0.01 standard deviation, and the bias is initialised to zero. Here, the ansatz parameters are not initialised uniformly from $[0, 2\pi)$ is mainly due to the residual scheme applied before the output of the current layer. They are using the parameter shift rule for the gradient calculation.

The QSANN is not a transformer as it lacks two key components: multi-head attention and position encoding. In their study, two simple synthetic datasets were used, named MC and RP. The MC dataset contains 17 words and 130 sentences (70 for training, 30 for development, and 30 for testing), with each sentence consisting of 3 or 4 words. The RP dataset contains 115 words and 105 sentences (74 for training and 31 for testing), with each sentence containing 4 words.

The results are somewhat unusual. The model achieved 100% accuracy on both training and testing for the MC dataset. For the RP dataset, the training accuracy was 95.35% and the testing accuracy was 67.74%. The second dataset's results seem more reasonable than the first. The anomalous result for the MC dataset might be due to the extremely small size of the dataset, which contains only 17 unique words.

Unlike previous authors, these authors provide a comprehensive and detailed explanation of their implementation, starting from the basics of quantum computing and progressing through to the classical gradient chain rule and the quantum parameter shift rule. They also include clear and informative diagrams to aid in their explanations. This thorough approach ensures that readers can follow the concepts and methodologies step by step. The depth of detail and clarity in this paper have been incredibly helpful in enhancing my understanding of how Quantum Transformers are implemented on quantum computers. Their meticulous explanation and visual aids make complex concepts more accessible and comprehensible, bridging the gap between classical and quantum machine learning techniques. This paper stands out as a valuable resource for anyone looking to grasp the intricacies of Quantum Transformers.

2.2.4 Quantum Vision Transformer

In 2024, Cherrat et al. [2] proposed two Quantum Vision Transformers. They designed a custom data loader with two registers. The top register loads the norms of each row of the input patch vector, while the lower register sequentially loads each row by applying the vector loader and its adjoint for each row, using CNOTs controlled by the corresponding qubit of the top register. In essence, they use amplitude encoding to transfer classical data into the quantum realm.

Cherrat et al. [2] also designed three circuits which they called the orthogonal layer: Pyramid, X, and Butterfly. Instead of using CNOTs to entangle the qubits, they employed a different entangling gate known as the RBS gate. The process begins with a Hadamard gate applied to each of the two qubits, followed by a two-qubit CZ gate. This is then followed by a Ry rotation gate on each qubit, where the top qubit is rotated by $\frac{\theta}{2}$ and the bottom qubit by $-\frac{\theta}{2}$. Another two-qubit CZ gate is applied, and finally, a Hadamard gate is applied to each qubit.

The first transformer they proposed is called the Quantum Orthogonal Transformer. This circuit begins by loading the input patch vector, followed by a trainable quantum orthogonal layer. Next, an inverse data loader for the input patch vector creates a state where the probability of measuring 1 on the first qubit is exactly the square of the attention coefficient. Another circuit, the Orthogonal Patch-wise Neural Network, is similar but lacks the inverse data loader. Its output, the feature vector, combined with the attention coefficient, can then be classically processed to obtain the attention vectors.

Additionally, they designed a circuit called Direct Quantum Attention to calculate the attention output within the quantum computer. This circuit consists of two registers: the top register loads the attention coefficients, while the bottom register contains a data loader for the input patch vector. The loader is controlled by the top register using a CNOT gate, so the input patch vector is loaded with all rows rescaled according to the attention coefficients. The data is then passed through a quantum orthogonal layer, which performs matrix multiplication between the feature vector and the newly encoded vector. The output of the second register is the attention vector. With this tool, the Quantum Orthogonal Transformer with Direct Quantum Attention can perform attention calculations entirely within the quantum realm.

They also proposed another Quantum Vision Transformer implementation, claiming it utilises quantum-native linear algebraic operations that cannot be efficiently performed classically. However, they did not publish their code, and Cara's attempts to replicate their work were unsuccessful, noting that some explanations in their work were unclear. Therefore, this implementation will only be briefly mentioned.

They trained and tested their models on the MedMNIST dataset, a collection of 12 pre-processed, two-dimensional, open-source medical image datasets. Their results demonstrated that the orthogonal transformer and the compound transformer, both of which implement non-trivial quantum attention mechanisms, delivered very competitive performances compared

to the classical vision transformer. Notably, these quantum transformers outperformed the classical vision transformer on 7 out of the 12 MedMNIST datasets.

The authors tested their models on actual quantum computers, providing a practical demonstration of their implementation. Furthermore, they developed a Quantum Transformer capable of calculating the attention output directly on the quantum computer. This approach contrasts with previous quantum transformers, which rely on classical computation for attention calculation. By integrating the attention mechanism fully within the quantum framework, their Quantum Transformer potentially offers improved performance and efficiency, showcasing a significant advancement in the application of quantum computing to machine learning tasks. This novel approach demonstrates the feasibility and benefits of performing more complex computations directly on quantum hardware.

2.3 Synthesis

All the mentioned models use variational quantum circuits to replace classical neural networks with quantum neural networks. Most of these models calculate the attention output classically, maintaining some dependence on classical computation. The only model that calculates the attention output directly on the quantum computer is the direct quantum attention add-on for Cherrat et al. [2]'s model, setting it apart as a more integrated quantum solution.

Despite these differences, a common feature among all models is the use of a classical fully connected layer for classification. This reliance on classical components means that all these models are hybrid quantum transformers rather than purely quantum-native transformers. The hybrid approach leverages the strengths of both quantum and classical computing but also indicates that fully quantum-native transformers are still under development.

Moreover, while these models share the goal of enhancing computational efficiency and performance through quantum computing, their varying approaches highlight the experimental nature of the field. For instance, the choice of entangling gates, data encoding strategies, and specific quantum operations differ across models, reflecting ongoing exploration of optimal techniques.

Interestingly, none of the papers discuss the issue of barren plateaus, a problem in quantum neural networks where gradients vanish during training. This phenomenon can severely affect training efficiency or even bring it to a halt, as the gradient of the model tends to zero in all directions. According to McClean et al. [10], quantum neural networks must be shallow and use a limited number of qubits to remain trainable, which contradicts the vision of high-dimensional, million-qubit quantum machine learning models. This issue underscores a significant challenge in the field that future research must address to realise the full potential of quantum neural networks.

Chapter 3

Methodology

This research follows an experimental framework aimed at comparing the performance of classical and quantum transformer models in natural language processing tasks. The study focuses on sentiment classification, evaluating the accuracy of predictions made by both classical transformers and hybrid classical-quantum transformers. In the quantum model, classical linear layers were replaced with variational quantum circuit (VQC), integrating quantum computation into the standard transformer architecture. Two types of transformer models were developed in this study: one with embedding and positional encoding layers embedded within the transformer, and another without these layers, requiring external encoding through the pre-trained BERT model.

Both models were trained and evaluated using identical datasets, allowing for a fair performance comparison based on a metric called accuracy. This approach was designed to explore the potential advantages of quantum transformers over classical models in terms of performance and computational efficiency, while also considering different encoding methods.

3.1 Data Collection

Publicly available datasets from the Hugging Face library were utilised to train and evaluate our transformer models. The datasets used were carefully selected for their diversity in sentiment analysis tasks and their scale, allowing robust model performance testing. Below is a detailed overview of the datasets:

- **IMDb Dataset**: This dataset [9] consists of 50,000 movie reviews and is designed for binary sentiment classification, with 25,000 reviews allocated for training and 25,000 for testing. The goal is to predict the polarity of reviews—either positive or negative—based on the text, making this dataset ideal for natural language processing and text analytics.
- **Amazon Polarity Dataset**: The Amazon Polarity dataset [15] contains approximately 35 million product reviews sourced from Amazon. Reviews are labelled as negative (ratings

of 1 or 2) or positive (ratings of 4 or 5), with ratings of 3 excluded. The dataset provides 1,800,000 training samples and 200,000 test samples for each class. Due to the large size of the dataset, it was sampled down to 600,000 reviews to make it manageable for storage.

- **Yelp Reviews Polarity Dataset**: Derived from the 2015 Yelp Dataset Challenge, this dataset [15] consists of 1,569,264 review samples, where star ratings of 1 and 2 are labelled as negative, and reviews rated 4 and 5 as positive. It includes 280,000 training samples and 19,000 test samples per class. Similar to the Amazon Polarity dataset, Yelp reviews were sampled down to 600,000 reviews for storage reasons.

3.1.1 Data Preprocessing

We utilised two distinct transformer models: one with an internal embedding and positional encoding layer, and another without these layers, requiring external encoding.

For the transformer model that includes an embedding layer, the IMDb dataset was pre-processed using the torchtext library. The vocabulary was built by tokenising the text with a basic English tokenizer and creating a vocabulary index list from the dataset. The vocabulary size was capped at 50,000 to ensure efficient memory usage during training. Special tags such as $\langle$unk$\rangle$ for unknown tokens and $\langle$pad$\rangle$ for padding were added to the vocabulary. Once the vocabulary was established, all words in the document were converted to their respective indices from the vocabulary, preparing the dataset for input into the transformer model.

For the transformer model without an embedding layer, the pre-trained BERT tokenizer from the Hugging Face transformer library was used to tokenise the Amazon Polarity and Yelp datasets. Given the large size of these datasets, they were sampled down to 600,000 entries to manage memory usage. After tokenisation, the pre-trained BERT model was used to encode the words into 768-dimensional word embeddings. The preprocessed datasets were then encoded and saved in tensor format, with approximately 800GB of memory required to store these tensors in the UWA Institutional Research Data Store (IRDS). Since the embeddings are fixed at 768 dimensions, further dimensionality adjustments are handled later in the pipeline of the model. Further sampling reduced the datasets to 300,000 entries for experiments.

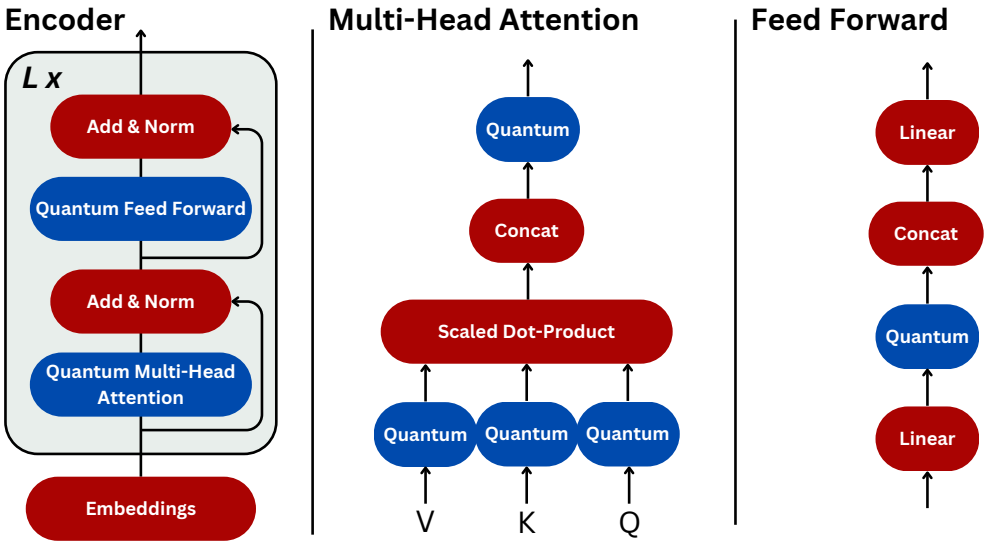
This preprocessing ensured that the datasets were ready for input into the respective transformer models, maintaining consistency in text encoding across experiments.

3.2 Implementation Details

The implementation of the transformer models involved several technical steps to ensure compatibility between the classical and quantum components and to handle the different preprocessing requirements of each model.

For the transformer model with internal encoding layers, the torchtext library was employed to manage tokenisation and vocabulary creation. Text was first tokenised into word indices, which were subsequently embedded using the built-in embedding layer of the model. The integral positional encoding was automatically applied to the embeddings, ensuring that sequence information was preserved. This model adhered to a more traditional deep learning framework, where data preprocessing and model operations were seamlessly integrated. The inclusion of these encoding layers enabled the model to process raw text data directly, without requiring any external encoding steps.

In contrast, the second transformer model relied on external preprocessing. The text data was tokenised using the pre-trained BERT tokenizer, and the resulting tokens were passed through the BERT model to generate 768-dimensional embeddings. These embeddings were then fed into the transformer model, ensuring that the input was in a structured format suitable for further processing, even in the absence of internal embedding layers. This architecture facilitated experimentation with external encoding methods and allowed for analysis of their impact on model performance.



(a) Quantum Transformer Architecture

Figure 3.1: An overview of the quantum transformer. The red components represent classical elements, including the embeddings, linear layers, and parts of the feedforward and attention mechanisms. The blue components denote quantum circuits, where quantum multi-head attention and quantum feedforward layers are integrated. Specifically, the classical linear layers are replaced by VQC, which process the word embeddings within the quantum sections of the model. The illustration of the Transformer Encoder was inspired by Sipio et al. [14].

For the quantum transformer, the classical linear layers were replaced by VQCs, as illustrated

in Figure 3.1. These circuits were constructed using the Pennylane framework. The quantum circuits processed the word embeddings, with an encoder used to adjust the dimensionality of the embeddings to match the input size constraints of the quantum circuits. Various encoding methods, including amplitude encoding and angle encoding, were employed to translate classical data into quantum states. For models using amplitude encoding, which compresses 2^n features down to n features, a decoder was needed to decompress the features for the next layer. The basic VQC consisted of a single layer of parameterised gates followed by a ring of controlled-NOT (CNOT) gates, introducing entanglement between qubits. The strong VQC increased the complexity, incorporating three layers of parameterised gates followed by a ring of CNOT gates. This layering enhanced the expressibility of the model, allowing it to capture more complex relationships in the input data.

There are also configurations that can be set within the quantum circuits. For angle encoding, the rotation gate can be chosen from R_x, R_y, and R_z, depending on the desired rotation axis. The basic and strong VQCs are templates provided by Pennylane. In the case of the basic VQC, the parameterised rotation gates can be selected as R_x, R_y, or R_z, giving flexibility in the choice of rotational operations. However, for the strong VQC, the parameterised gates are fixed to a sequence of R_z, R_y, and R_z gates, which cannot be altered. The only adjustable component in the strong VQC is the choice of the two-qubit control gate, where options such as CNOT, CZ, or other controlled gates can be applied.

Finally, the expectation values of the Pauli-Z for all qubits are measured. Measuring in Pauli-Z corresponds to measuring in the computational basis (i.e., $|0\rangle$ and $|1\rangle$), which allows us to interpret the results classically. Although other measurements, such as Pauli-X, are possible, which measure in the $|+\rangle$ and $|-\rangle$ basis, we focus on Pauli-Z measurements as they directly reflect the classical binary outcomes needed for further processing in the model.

Both models were trained and evaluated on identical datasets, with the same training configurations, including learning rates and batch sizes. The evaluation focused on comparing the performance of the models using accuracy metrics. By using the same datasets and training conditions, the study provided a fair comparison between the classical and quantum transformer models. Simulated quantum environments were employed for the quantum model experiments, using Pennylane and TensorCircuit to run simulations on classical hardware.

3.3 Setup

The experiments were conducted in a High-Performance Computing (HPC) environment using two types of GPU clusters, depending on the availability of the clusters, to train and evaluate both classical and quantum transformer models. The following describes the computational environment:

- **Hardware**: Two different types of GPU nodes were utilised from the University of

Western Australia (UWA) cluster:

  - **Large GPU Nodes**: Dual Intel Xeon CPUs (36 cores total @ 3.1GHz), 768GB RAM, and Dual NVIDIA 32GB V100 GPU cards.
  - **Medium GPU Nodes**: Dual Xeon CPUs (28 cores @ 2.0GHz), 256GB RAM, and 4 NVIDIA 16GB P100 GPU cards.

- Only one GPU was used per experiment. This limitation was necessary due to Pennylane's own parallelisation mechanisms, which are unstable and prone to conflicts with PyTorch's parallelisation. Both GPU nodes were configured to use CUDA 12.4.

- **Software Stack**: The following software stack was used for model development and experimentation:

  - **Python**: Python 3.11.8 was used as the core language for model development and experimentation.
  - **Jupyter Notebook**: All experiments were conducted and documented using Jupyter Notebooks for interactive model training and evaluation.
  - **Conda**: Conda was used as the package manager for creating isolated environments with the necessary dependencies for both classical (PyTorch, TensorFlow) and quantum (Pennylane, TensorCircuit) frameworks.

To ensure that all experiments were reproducible, several measures were implemented. First, all models were seeded, ensuring that the same random initialisations and conditions were applied across different experiments. This enabled reproducibility when using the same hardware and software configurations, ensuring consistency in the behaviour of the model.

Additionally, intermediate results, such as model checkpoints and partial outputs, were saved at various stages throughout the training process. This approach allowed for consistent tracking of progress and made it easier to reproduce experiments if needed. Saving intermediate outputs also helped resume training from checkpoints without having to restart from the beginning.

Lastly, Conda was used to manage the computational environment, ensuring that dependencies remained consistent across experiments. By isolating environments, potential issues related to version conflicts or software updates were avoided, maintaining a stable environment for the duration of the project.

Chapter 4

Experiments and Results

4.1 Experiments

A series of experiments were conducted to evaluate the performance of classical and quantum transformer models using different encoding techniques and variations of quantum circuit-based layers. The goal was to compare these models not only in terms of their accuracy but also in their training speed across different computational setups. To avoid inconsistencies in the implementation, we also refactored and aligned the model frameworks for fair comparison.

The experiments were performed on three datasets: IMDb, Amazon, and Yelp, each preprocessed and tokenised with a maximum sequence length of 128 tokens. The IMDb dataset was used for models both with and without embedding layers, while the Amazon and Yelp datasets were only used for models without embedding layers. A sampling strategy was applied to ensure the appropriate dataset size for training.

Two broad categories of models were tested. The first category involved models with embedding layers, where the vocabulary size was set to 50,000, and the architecture included two transformer layers, each with two multi-head attention heads. Quantum models in this category employed angle encoding using either R_x or R_z gates, integrated with either basic or strong VQCs, where the number of quantum layers varied between 3 and 8. For certain models, amplitude encoding was also explored using TensorFlow-CPU as the backend for TensorCircuit.

For models without embedding layers, we used external BERT embeddings with fixed embedding sizes of 8 and 256 dimensions. These models were tested in both classical and quantum variants, applying either angle or amplitude encoding. The quantum circuits used in these experiments involved both basic VQC architectures, with a single layer of parameterised gates followed by CNOT gates, and strong VQC architectures, with three layers of parameterised gates. Across all models, the training process was handled with the same set of hyperparameters, ensuring consistent learning conditions for each experiment.

Due to the fact that TensorCircuit only supports Just-In-Time (JIT) compilation and vectorised parallelism on TensorFlow and not PyTorch, we initially created what might be

termed a "Frankenstein" model—combining a classical model implemented in PyTorch with a quantum model in TensorCircuit (which uses TensorFlow). To address this inconsistency across frameworks, we developed a new version of the transformer model in TensorFlow. This unified approach allows both classical and quantum models to share the same backend, ensuring compatibility in CUDA versions. Due to CUDA version conflicts, the TensorFlow setup was initially limited to CPU. However, the new model in TensorFlow was designed specifically to utilise TensorFlow CUDA for TensorCircuit. This enabled a comparison of training speed between CPU and CUDA-enabled systems. The newly created models were refactored and tested against their older counterparts to measure improvements in training speed.

The following models were tested for training time:

1. The original PyTorch-based quantum model using GPU with PennyLane.
2. The original PyTorch-based quantum model using CPU with PennyLane.
3. The refactored PyTorch-based quantum model using GPU with PennyLane.
4. The refactored PyTorch-based quantum model using CPU with PennyLane.
5. The original PyTorch-based quantum model using TensorCircuit with TensorFlow-CPU as the backend.
6. The refactored PyTorch-based quantum model using TensorCircuit with TensorFlow-CPU as the backend.
7. The refactored TensorFlow-based quantum model using TensorCircuit with TensorFlow-CUDA as the backend.

Each of these models was trained for 4 epochs, and the average training time across these epochs was recorded for comparison. This allowed us to evaluate any potential speed increases when switching to a TensorFlow-CUDA backend for TensorCircuit, and whether the refactored models offered performance improvements over the original implementations.

The hyperparameters across all experiments were consistent. The learning rate was set at 0.001, with the Adam optimiser applied across both classical and quantum models. For most experiments, a batch size of 64 was used. A maximum sequence length of 128 was enforced for all input data, and each model utilised two multi-head attention layers and two transformer layers. The StepLR learning rate scheduler, with a step size of 5 and gamma of 0.1, was used for dynamic learning rate adjustment. For all models, the loss function was Binary Cross-Entropy with Logits Loss (BCEWithLogitsLoss), which combines the sigmoid activation and binary cross-entropy into a single step. The model outputs logits (raw scores), which are passed through a sigmoid function to transform them into probabilities between 0 and 1. These probabilities are then used to compute the binary cross-entropy loss, measuring how well the predicted probabilities match the true binary labels.

4.2 Results and Analysis

Our experimental results comparing classical and quantum transformer models with different encoding strategies are presented and analysed below. The experiments were conducted over various datasets, both with and without embedding layers, and across multiple quantum circuit designs.

4.2.1 Transformers with Embedding Layers

The experiments on models with embedding layers focused on comparing classical transformer models with quantum models that utilised angle encoding encoding integrated with basic and strong VQCs parameterised by the R_x gate. The dataset sizes ranged from 3,200 to 5,000, and the models were trained for 30 epochs.

(a) Classical vs. Quantum Model with Angle Encoding (Basic VQC)

Figure 4.1: A loss plot and an accuracy plot for the classical and quantum models on the IMDb dataset (3,200 samples), with 50,000 vocab size and 64 batch size, trained for 30 epochs. The plot compares the classical transformer model with a quantum transformer using angle encoding and 8 layers of basic VQCs.

The loss plot in Figure 4.1 shows the performance of two models: a classical transformer (blue) and a quantum model (red) with 8 layers of VQCs. For the classical model, overfitting is evident, as the training loss continues to decrease while the validation loss starts to diverge after the 17th epoch. At the lowest point, the validation loss reaches approximately 0.6 at the 17th epoch, while the training loss is around 0.42. These relatively high loss values indicate that the model is not performing optimally. In contrast, the quantum model shows little improvement in loss reduction until the 15th epoch, after which it starts to decrease slowly. By the 20th epoch, the quantum model's training loss is 0.66, and its validation loss is 0.69, indicating that the

quantum model struggled to learn effectively as both models began with a loss value of 0.7.

The accuracy plot in Figure 4.1 further highlights the classical model's superior performance. The classical model achieves a peak validation accuracy of 0.75, while the quantum model's validation accuracy only reaches 0.61. Additionally, there is a noticeable gap between the training and validation accuracy for the classical model, further confirming the presence of overfitting.

While the quantum model begins to learn around the 15th epoch, we stopped training at the 30th epoch. This suggests that extending the training could potentially allow the quantum model to learn further. However, this experiment was preliminary, designed to evaluate whether the quantum model could be trained within a feasible amount of time. Moving forward, further optimisations are necessary to enhance its learning efficiency and accuracy.

The poor performance of both models in terms of loss and accuracy can largely be attributed to the limited size of the dataset used and the challenges faced by the quantum model due to the depth of its circuit. For future experiments, increasing the dataset size and reducing the circuit depth could potentially yield better results and allow the quantum model to learn faster and more effectively.

(a) Classical vs. Quantum Models with Different Quantum Configurations

Figure 4.2: A loss plot and an accuracy plot for classical and quantum models on the IMDb dataset (5,000 samples), with a vocab size of 50,000 and a batch size of 32, trained over 30 epochs. The figure compares the classical model with quantum models using various configurations of angle encoding (R_x and R_z gates) and backends (QML and TensorCircuit) with 3 layers of VQCs.

In this experiment, we increased the dataset size to 5,000 samples and tested four quantum models with different configurations. To explore potential performance improvements, we reduced the batch size, hoping it would enhance the ability of the models to generalise and achieve higher accuracy.

The loss plot in Figure 4.2 reveals that all models, including the classical and quantum models, exhibit overfitting. While the training losses for all quantum models closely follow the training loss of the classical model (blue), the validation losses remain problematic. None of the models achieve a validation loss below 0.6, with all models starting at approximately 0.7. The fluctuation observed in the validation loss, particularly for the quantum models, indicates instability during training, suggesting that the batch size may be too small for stable generalisation.

When examining the accuracy plot in Figure 4.2, it is clear that the classical model still outperforms all quantum models. Among the quantum models, the configuration with two layers of angle encoding (purple) shows marginally better performance during training but results in similar validation accuracies to the other quantum models. This suggests that varying the rotation gate or adding additional layers of angle encoding does not significantly improve performance. The substantial fluctuations in the validation accuracy of the quantum models further point to the small batch size as a limiting factor in achieving stable training.

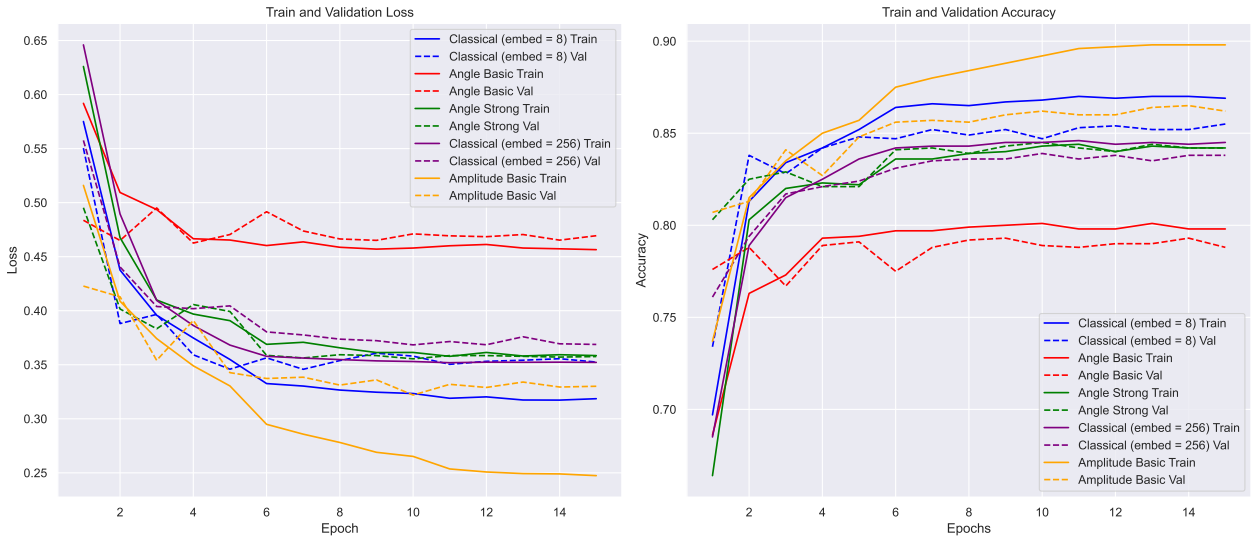
Given these results, future experiments should focus on extracting the embedding layer and using a pre-trained model to encode the documents. This approach may provide a stronger feature representation and lead to improved accuracy. Additionally, increasing the batch size back to 64 should help stabilise training and improve the overall model performance, especially for the quantum models. Finally, leveraging the full dataset could further refine the learning capacity of the quantum model learning capacity and boost validation accuracy.

4.2.2 Transformers without Embedding Layers

By eliminating the internal embedding layers, we aimed to assess how well the models could handle these externally encoded inputs without relying on the internal embedding layers. The models were evaluated on the IMDb, Amazon, and Yelp datasets, using embedding sizes of 8 for angle encoding and 256 for amplitude encoding, with a batch size of 64 over 20 epochs. The quantum models employed three layers of basic and strong VQCs parameterized by the R_z gate using 8 qubits.

The loss plot in Figure 4.3 demonstrates that all models are learning effectively, with no signs of overfitting or instability. The absence of fluctuations in the validation loss indicates that the batch size of 64 is well-suited for this experiment. The loss values are generally low, signalling that the models are optimising as expected. However, the quantum model using angle encoding with a basic VQC (red) is underperforming relative to the other models, trailing by at least 0.1 in loss reduction. In contrast, the quantum model with amplitude encoding and a basic VQC (yellow) is outperforming all other models, including the classical models.

Interestingly, the classical model with an embedding size of 8 also performs better than the classical model with an embedding size of 256. This could suggest that smaller embedding sizes



(a) IMDb Dataset - Classical vs. Quantum Models (Embedding Size 8 and 256)

Figure 4.3: A loss plot and an accuracy plot for classical and quantum models on the IMDb dataset, using embedding sizes of 8 and 256, batch size of 64, trained over 20 epochs. The figures compare the classical transformer models to quantum models with angle encoding (3 layers of basic and strong VQCs) and amplitude encoding (3 layers of basic VQCs).

encourage the model to learn more compact and efficient feature representations, leading to improved performance. The quantum models with amplitude encoding further highlight the effectiveness of this approach, demonstrating that advanced quantum circuits may offer superior learning capabilities in certain scenarios.

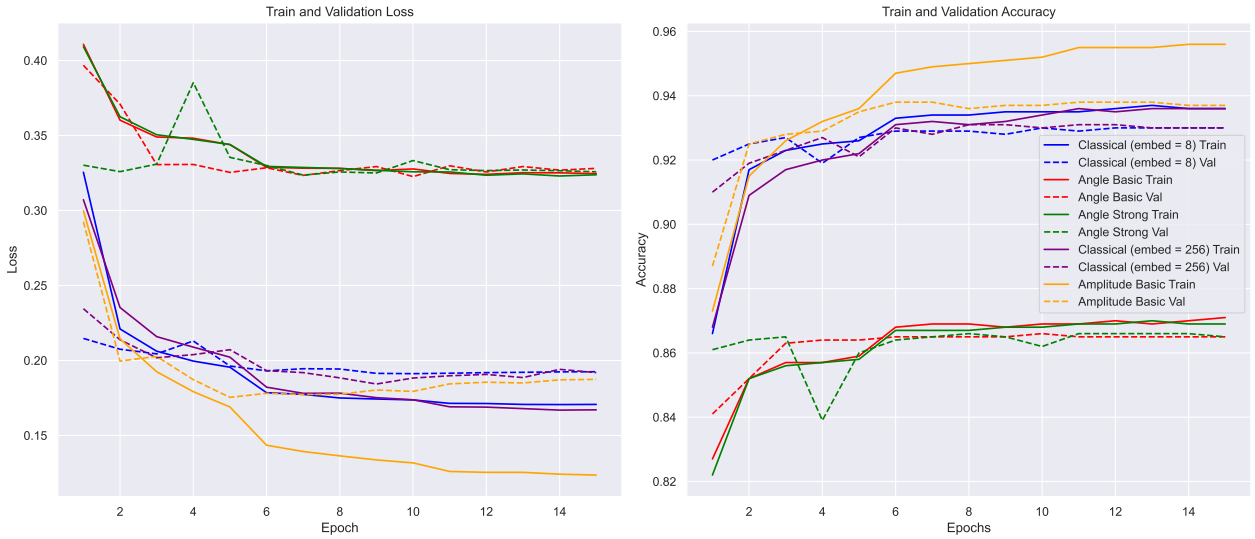
In the accuracy plot of Figure 4.3, we observe similar trends as seen in the loss plot. Even the worst-performing model—again, the quantum model with angle encoding and a basic VQC (red)—achieves a solid accuracy of at least 0.78. Despite lagging behind by approximately 0.08 compared to the other models, this still represents a competitive performance. Moreover, the quantum model with the strong VQC (green) outperforms its counterpart with the basic VQC, indicating that increasing circuit complexity improves accuracy, though not as drastically as one might expect.

The top-performing model is the quantum model using amplitude encoding (yellow), which achieves an impressive accuracy of 0.86, surpassing the classical models. This reinforces the notion that amplitude encoding, when combined with quantum circuits, can provide significant advantages in learning performance, especially when compared to classical architectures and simpler quantum models.

In this study, one of our key objectives was to reproduce results from prior research [2], which indicated that quantum models have the potential to outperform classical models. Based on these results, we have successfully achieved this goal, with the quantum model using amplitude encoding consistently outperforming the classical models in both loss and accuracy. This confirms the findings from previous studies and demonstrates that well-optimised quantum

models can indeed surpass classical models in certain learning tasks.

Overall, these models demonstrate substantial improvements over previous configurations. The results indicate that quantum models, particularly those using amplitude encoding, are capable of outperforming classical models when tuned appropriately. Given the promising results, the next logical step would be to test these models on additional datasets to evaluate their generalisability and robustness across different tasks. This would provide a more comprehensive understanding of their strengths and limitations in diverse scenarios.



(a) Amazon Dataset - Classical vs. Quantum Models (Embedding Size 8 and 256)

Figure 4.4: A loss plot and an accuracy plot for classical and quantum models on the Amazon dataset, with embedding sizes of 8 and 256, batch size of 64, trained for 20 epochs. The figures compare the classical transformer models to quantum models with angle encoding (3 layers of basic and strong VQCs) and amplitude encoding (3 layers of basic VQCs).

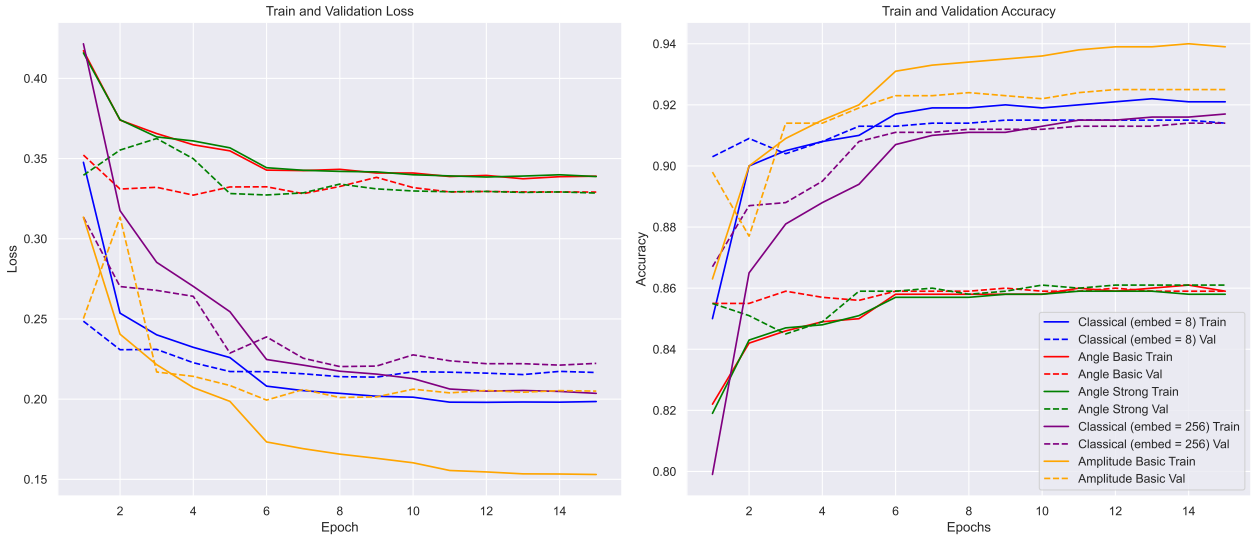
In the loss plot of Figure 4.4, we can see that the loss values across all models appear to be optimally decreasing as training progresses. The training losses consistently drop and eventually plateau, while the validation losses show minimal decline, suggesting that the models are generalising effectively. Interestingly, the quantum model using a strong VQC (green), which previously performed better, now lags behind both the classical models and the quantum model with amplitude encoding. The strong VQC exhibits a notable spike around the 4th epoch, which may indicate that the weights of the model fell into a poor local minimum. This behaviour suggests that the current optimiser might not be ideal for this configuration, and future experiments could benefit from exploring different optimisers, changing the weight initiliasation method or lowering the learning rate to help avoid such performance dips.

Additionally, the quantum model with amplitude encoding (yellow) continues to outperform the other models in terms of training loss. However, its validation loss converges to similar levels as the other models, meaning its generalisation to unseen data is comparable to the classical models.

In the accuracy plot of Figure 4.4, the overall performance of the models has improved compared to previous experiments. All models except the strong VQC exhibit stable and steadily increasing accuracy. The quantum model with amplitude encoding (yellow) maintains its lead, achieving the highest training and validation accuracy across all models. However, the strong VQC model again shows a significant dip in accuracy at the 4th epoch, further reinforcing the idea that it might have encountered issues with its optimisation process.

The behaviour of the strong VQC model, which falls behind in both loss and accuracy, indicates that increasing circuit complexity does not always translate into better performance. The large spike in both the loss and accuracy plots suggests that more careful tuning of hyperparameters, such as the learning rate or optimiser, could improve the stability and performance of this model.

Overall, these results are promising, as the models are learning well and generalising effectively. The fact that amplitude encoding outperforms the other models in training suggests that it is an efficient strategy for quantum models. Moving forward, it would be valuable to test these models on additional datasets and experiment with different optimisers or regularisation techniques to further refine their performance.



(a) Yelp Dataset - Classical vs. Quantum Models (Embedding Size 8 and 256)

Figure 4.5: A loss plot and an accuracy plot for classical and quantum models on the Yelp dataset, using embedding sizes of 8 and 256, batch size of 64, trained for 20 epochs. The figures compare the classical transformer models to quantum models with angle encoding (3 layers of basic and strong VQCs) and amplitude encoding (3 layers of basic VQCs).

These plots in Figure 4.5 are largely consistent with the previous results, with similar patterns emerging. Once again, we observe early spikes in validation loss for both the strong VQC (green) and amplitude encoding (yellow) models. This indicates that these models are still struggling with initial instability during training. Correspondingly, their accuracies also drop early on before stabilising, reflecting the same training difficulties.

Despite this early volatility, both models recover and stabilise as training progresses. The amplitude encoding model continues to perform well, achieving the highest accuracy, while the strong VQC remains comparable to other models, though it does not show any significant advantage over the classical models or basic VQCs.

The superior performance of the amplitude encoding model suggests potential improvements for future architectures. One idea is to replace the angle encoding in the first transformer block layer with amplitude encoding. By doing this, we can eliminate the encoding layer that squeezes the input dimension from 768 to 2^8 (256) and instead feed the entire input dimension of 768 directly into the quantum circuit using amplitude encoding. With 10 qubits, this could handle 1024 values, padding the input with zeros if necessary to match the required size of 1024. In subsequent layers, where the input dimension would be reduced to 10, angle encoding could still be applied effectively.

Additionally, adding a Hadamard gate before the angle encoding in these subsequent layers could increase the expressibility of the quantum circuits [13]. Moreover, replacing the CNOT gates in the VQC with parameterised controlled rotation gates would provide even greater expressibility, potentially enhancing the ability of the model to capture more complex relationships in the data [3]. These changes could help address some of the initial instability observed in the current quantum models.

In summary, while the early fluctuations remain an issue, the overall performance of the models aligns with previous findings. The amplitude encoding model outperforms the others, and with further optimisation, especially through architectural adjustments like those mentioned above, the performance of quantum models could be improved even further.

4.2.3 Training Speed Comparison

Table 4.1: Training Time Comparison Across Deep Learning and Quantum Computing Frameworks.

Deep Learning Framework	**Quantum Computing Framework**	**per Epoch**
PyTorch	PennyLane (CPU)	4.5 hour
PyTorch	PennyLane (CUDA)	3.5 hour
PyTorch	TensorCircuit (Tensorflow-CPU)	2.5 hour
PyTorch + Refactored Code	PennyLane (CPU)	10 min
PyTorch + Refactored Code	PennyLane (CUDA)	7 min
PyTorch + Refactored Code	TensorCircuit (Tensorflow-CPU)	3 min
Tensorflow + Refactored Code	TensorCircuit (Tensorflow-CUDA)	2 min 45 sec

In Table 4.1, we present a comparison of training times across various deep learning and quantum computing frameworks, as shown in the table. One of the key objectives of this work was to significantly reduce training time, and we achieved this goal by a considerable margin.

Initially, using PyTorch combined with PennyLane on the CPU took 4.5 hours per epoch, with CUDA providing a slight improvement, bringing the time down to 3.5 hours. Switching to TensorCircuit with TensorFlow on the CPU resulted in further reductions, lowering the training time to 2.5 hours per epoch.

However, after refactoring the code to improve efficiency, the training time saw dramatic improvements. With the refactored PyTorch code running on PennyLane CPU backend, the training time was reduced to just 10 minutes per epoch. Similarly, the CUDA version of PennyLane with refactored code reduced the time to 7 minutes. The most significant reduction came with TensorCircuit and TensorFlow on both CPU and CUDA. The refactored code brought the training time down to 3 minutes on TensorFlow-CPU, and using TensorFlow-CUDA, we achieved the fastest training time of 2 minutes and 45 seconds per epoch.

It is important to note that the training time for the first epoch is generally longer than for subsequent epochs due to JIT compilation in TensorCircuit. Once the compilation is complete, the remaining epochs train much faster, further optimising the training process.

This dramatic reduction in training time highlights the effectiveness of both code optimisation and leveraging more efficient quantum computing frameworks like TensorCircuit. These improvements not only make training quantum models more feasible but also align with our objective of reducing the overall time required to train these models significantly.

Chapter 5

Conclusion and Future Work

5.1 Summary of Contributions

In this work, we explored the performance of quantum transformer models and compared them to classical transformers across several experiments. By designing and implementing different quantum circuits, including basic and strong VQCs, and employing both amplitude and angle encoding techniques, we investigated how quantum models can be optimised for natural language processing tasks. Our findings revealed that, under certain configurations, quantum models, particularly those using amplitude encoding, were able to outperform their classical counterparts, aligning with the results of prior studies.

A key focus of this work was the optimisation of training time, an important consideration in the practical use of quantum models. By refactoring our code and adopting more efficient quantum computing frameworks like TensorCircuit with TensorFlow-CUDA, we achieved a significant reduction in training time, bringing it down from hours per epoch to just minutes. This optimisation was critical in demonstrating that quantum models can be trained within a reasonable time frame, making them more viable for large-scale tasks.

While the results are promising, there are still challenges to address, particularly in stabilising the training process for some quantum models. The early spikes in validation loss, especially for models using strong VQC and amplitude encoding, highlight areas where further optimisation is needed. Additionally, our exploration of encoding methods showed that quantum models can benefit from different approaches depending on the layer, which opens avenues for future work.

In summary, this work contributes to the growing field of quantum machine learning by demonstrating that with careful design and optimisation, quantum models can match or even exceed the performance of classical models in specific tasks, all while reducing the required training time. Future efforts will focus on further refining these models, exploring more advanced quantum circuits, and testing their generalisability across a wider range of datasets and tasks.

5.2 Future Work

Building on the results of this study, there are several avenues for further research and optimisation. One promising direction is the superior performance observed in models using amplitude encoding, which suggests potential improvements for future quantum architectures. Specifically, replacing the angle encoding in the first transformer block layer with amplitude encoding could offer significant benefits. This modification would allow the removal of the encoding layer that reduces the input dimension from 768 to 2^8 (256), and instead, the entire input dimension of 768 could be directly passed into the quantum circuit using amplitude encoding. To achieve this, we would require 10 qubits to handle 1024 values, with zero-padding applied where necessary to match this size. For subsequent layers, where the input dimension is reduced to 10, angle encoding would still be a viable approach.

Further optimisation could also involve incorporating a Hadamard gate before the angle encoding in these subsequent layers. This adjustment could improve the expressibility of the quantum circuits, enabling them to capture more complex relationships within the data, as demonstrated by Sim et al. [13]. Additionally, replacing the CNOT gates in the VQC with parameterised controlled rotation gates could further enhance expressibility and allow the circuits to better model intricate patterns in the data [3]. These changes could potentially address the initial instability observed in the quantum models during training, particularly for those using strong VQC and amplitude encoding.

In addition to these architectural improvements, there is potential to explore replacing the classical multi-layer perceptron (MLP) components with a quantum MLP. One approach is to replace the ReLU activation function with a quantum equivalent or to apply input re-uploading technique [3], where the original input or the ReLU output is angle-encoded between two VQC layers. This would introduce non-linearity into the quantum model, enhancing its capacity to learn more complex representations. Such changes could further refine the hybrid quantum-classical models and lead to more robust performance improvements.

These proposed modifications will not only build upon the gains seen in the current study but also push the boundaries of quantum machine learning by leveraging advanced quantum circuits and non-linear transformations. The next steps will involve implementing these ideas and conducting experiments to assess their impact on model performance and training stability.

Appendix A

Research Proposal

A.1 Background

The advent of deep learning has been transformative, marking a paradigm shift in our approach to artificial intelligence. Prior to 2011-2012, deep learning was largely deemed impractical. This perception changed dramatically when a team participating in the ImageNet Large Scale Visual Recognition Challenge (ILSVRC) in 2012 leveraged deep learning to secure a dominating lead, outperforming the second-place team by 9.8 percentage points—a margin that highlighted the difference between second and third place of 0.8.

The roots of deep learning extend back to 1943, yet it remained largely theoretical until recent advancements in hardware made its practical application feasible. Today, we stand on the shoulders of giants, benefiting from the perseverance of researchers who continued to explore this field despite its challenges.

Quantum computing holds similar transformative potential. The transformer model represents the next evolutionary step in deep learning, and by laying the groundwork for a quantum transformer, we can pave the way for future advancements.

Sipio et al. [14] proposed a quantum-enhanced transformer for sentiment analysis by adapting the multi-head self-attention and feed-forward layers to the quantum realm while Li et al. [7] introduced Gaussian Projected Quantum Self-Attention, which they used to create a Quantum Self-Attention Neural Network for text classification. They argued that this method is more suitable for handling quantum data than the straightforward inner-product self-attention used in the work by Sipio et al. [14]. Notably, in both models, the computation of attention coefficients and outputs remains within the classical domain.

In terms of performance, Cherrat et al. [2] provided results supporting the notion that quantum transformers may match the performance of their classical counterparts while requiring fewer resources in terms of runtime and parameter count. However, their approach has faced criticism, particularly regarding the exponential cost associated with encoding matrices into quantum states.

The field of quantum computing is poised to revolutionise deep learning, yet the integration of quantum mechanisms within neural network architectures remains under-explored. This study seeks to address the problem of how quantum components, specifically variational quantum circuits, can be effectively incorporated into transformer models to enhance their learning efficiency and reduce generalisation errors. Specifically, we seek to enhance the quantum-enhanced transformer initially proposed by Sipio et al. [14]. We hypothesise that a quantum transformer will require less training time and exhibit improved performance metrics compared to its classical counterparts.

A.2 Aim

To develop and validate a quantum transformer model that integrates variational quantum circuits into its architecture, thereby enhancing learning efficiency and reducing generalisation errors. This will involve:

1. Analysing the current limitations of classical transformer models in terms of learning efficiency and generalisation.
2. Designing a quantum transformer architecture that incorporates variational quantum circuits as a core component.
3. Comparing the performance of the proposed quantum transformer with classical models, focusing on training time and performance metrics.
4. Assessing the feasibility of the quantum transformer in practical applications, considering both its computational efficiency and the quality of its outputs.

A.3 Methodology

A.3.1 Data Collection

Before diving into our models, it is essential to outline the datasets employed in this study. We start with a lightweight dataset, which lays the groundwork for our initial analysis then transition to a more sophisticated dataset, presenting a richer set of challenges and opportunities for our models to tackle.

In this project, we will use three well-known text classification datasets: the IMDb, Amazon Polarity, and Yelp Reviews Polarity datasets, to explore the performance of our models.

The IMDb dataset consists of 50,000 movie reviews, making it ideal for natural language processing and text analytics. This dataset is designed for binary sentiment classification, where each review is classified as either positive or negative. With 25,000 movie reviews allocated for training and another 25,000 for testing, this dataset offers a robust amount of data for sentiment

analysis tasks. Our model will predict the polarity of reviews, aiming to differentiate between positive and negative sentiments using deep learning or classification algorithms.

The Amazon Polarity dataset is a collection of product reviews sourced from Amazon over an 18-year period. The dataset includes approximately 35 million reviews up to March 2013. Reviews with ratings of 1 and 2 are labelled as negative (class 1), while reviews rated 4 and 5 are labelled as positive (class 2), with reviews rated 3 being excluded. The dataset provides 1,800,000 training samples and 200,000 testing samples for each class, making it a large-scale dataset well-suited for binary classification tasks. The inclusion of product and user information, ratings, and plaintext reviews enables us to further explore consumer sentiment through our text transformer.

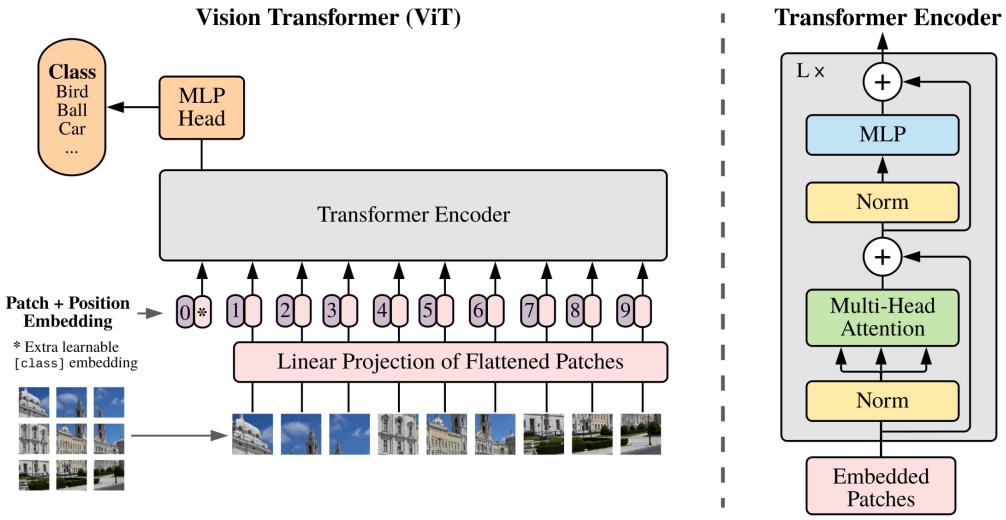
The Yelp Reviews Polarity dataset is derived from the 2015 Yelp Dataset Challenge and consists of 1,569,264 samples of review text. For this polarity classification task, reviews with star ratings of 1 and 2 are labelled as negative, while reviews rated 3 and 4 are labelled as positive. The dataset provides 280,000 training samples and 19,000 test samples per polarity class. This dataset provides another large and diverse set of review data that can be utilised to refine our text transformer model, further strengthening its ability to classify sentiment in various domains.

A.3.2 Models

Classical Transformer

We will implement a classical transformer based on the standard architecture as shown in Figure A.1. While the diagram represents the structure of a vision transformer, the overall structure of the text transformer will be similar, with key modifications to accommodate text-based input. We chose the standard transformer encoder model to serve as the baseline for our benchmark. The general steps of our text transformer are as follows:

1. The input text is tokenised into sequences of words or subwords using a pre-trained tokenizer.
2. Each token is then transformed into embeddings using a pre-trained word embedding model such as BERT.
3. Since the transformer processes the tokens in parallel, it does not inherently capture the order of the sequence. To address this, positional embeddings are added to the token embeddings to encode the sequential information.
4. The new embeddings, along with an additional special token embedding, are passed through the transformer encoder.
5. The output of the encoder is then processed by a multi-layer perceptron (MLP) head, which converts the token representations into a class prediction for sentiment analysis.



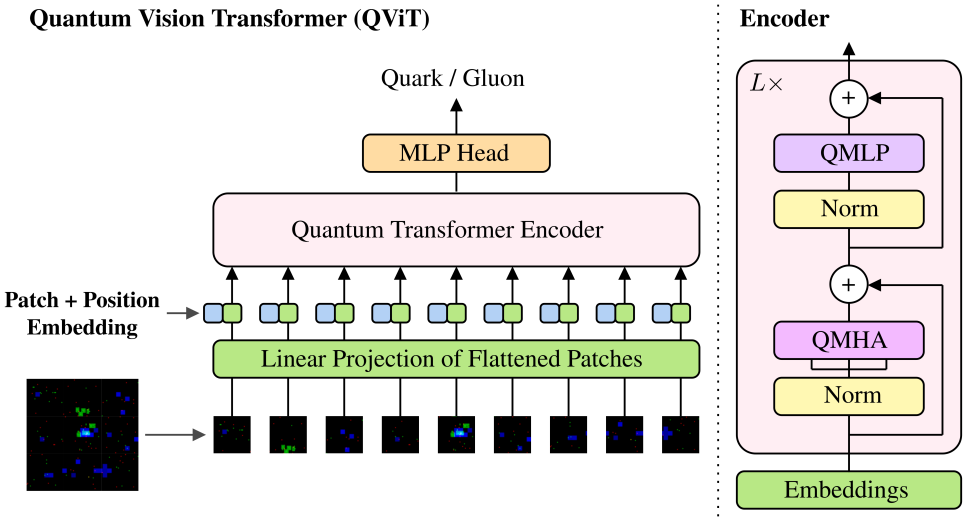
(a) Vision Transformer Architecture

Figure A.1: An overview of the classical model [5].

In the standard transformer encoder, we use multi-head attention, which employs several self-attention mechanisms to capture different types of relationships between tokens. The multi-layer perceptrons (MLPs) contain two layers with Gaussian Error Linear Unit (GELU) non-linearity to model complex interactions between tokens. Layer normalisation stabilises and accelerates training, while residual connections prevent the vanishing gradient problem.

This architecture will be adapted to text-based sentiment analysis tasks using datasets such as IMDb, Amazon Polarity, and Yelp Reviews Polarity.

Quantum Transformer



(a) Quantum Vision Transformer Architecture

Figure A.2: An overview of the quantum model [1]. The illustration of the Transformer Encoder was inspired by Sipio et al. [14].

Cara [1] introduced a model depicted in Figure A.2, which incorporates variational quantum circuits into the multi-head attention and multi-layer perceptron components of the original architecture. These circuits serve as the equivalent of fully-connected layers within both components, potentially offering improved performance in specific tasks. As our comprehension of the quantum transformer deepens, the model may be further augmented with additional quantum components.

A.3.3 Training and Evaluation

To maintain consistency with the previous work by Cara [1] and Cherrat et al. [2], we will adopt similar hyper-parameters: cross-entropy loss function, Adam optimizer, 100 training epochs, a batch size of 32 and a learning rate of 10^{-3}.

For model performance evaluation, we will employ two metrics: the area under the receiver operating characteristic (ROC) curve (AUC) and accuracy (ACC).

A.4 Software and Hardware Requirements

A.4.1 Software

- Python: An easy-to-learn and really popular programming language to write quantum code.
- PyTorch: A Python library for deep learning.
- TensorFlow: Another Python library for deep learning.
- PennyLane: A Python library for quantum machine learning.
- TensorCircuit: Another Python library for quantum machine learning.
- Overleaf: An online latex editor for documentation.
- GitHub: An online hub to version control the code for this project.
- Visual Studio Code: A code editor to write Python code and SSH into Kaya.
- Hugging Face: A machine learning platform for sharing machine learning models and datasets.

A.4.2 Hardware

- Personal Laptop: A device to write and edit code.
- Kaya: The UWA High-Performance Computational Research Platform.

A.5 Timeline

The following comprises a rough timeline for the outlined project.

- Research proposal: 18th of March
- Quantum Computing Study: February - 14 October
  - Textbook Readings:
    - Explorations in Quantum Computing (Recommended by Jingbo)
    - Quantum Computing for Computer Scientists (Recommended by Microsoft)
    - Quantum Computation and Quantum Information (Recommended in the Quantum Computing Subreddit)
  - Interactive Learning:
    - IBM Quantum
    - Microsoft Azure Quantum
    - PennyLane Xanadu Quantum Codebook
- Implement a classical transformer from scratch: 18 March - 31 March
- Train and evaluate the transformer: April - May
- Literature Review: March - 13 May
- Implement a quantum transformer: May - August
- Seminar Abstract: August - 16 September
- Seminar: August - (30 September - 4 October)
- Thesis: August - 14 October

Appendix B

User Manual

B.1 Installation

To set up the required dependencies for this project, follow these steps:

1. First, install all dependencies listed in the `requirements.txt` file by running the following command:

   ```
   pip install -r requirements.txt
   ```

   If the installation fails, you may need to install some dependencies manually.

2. For the PyTorch environment, manually install the required packages by running the following commands:

   ```
   # Install PyTorch with CUDA support
   pip install torch torchvision torchaudio --index-url \
     https://download.pytorch.org/whl/cu124

   # Install other essential packages
   pip install -U pennylane scikit-learn tqdm \
     seaborn ipykernel nbconvert papermill

   # Install TensorCircuit
   pip install tensorcircuit

   # Install Hugging Face tools
   pip install datasets transformers accelerate

   # Install additional quantum and LaTeX-related tools
   pip install -U qiskit cirq pylatexenc
   ```

3. To install TensorFlow, use the following commands:

   - **CPU version:**

     ```
     pip install tensorflow
     ```

   - **GPU version:**

     ```
     pip install tensorflow[and-cuda]
     ```

Note: TensorCircuit relies on Qiskit to render quantum circuit diagrams.

By following these steps, you should have all the required packages installed to run the project smoothly.

Bibliography

[1] M. C. Cara. Quantum transformers in google summer of code 2023 at ml4sci, 2023. URL https://salcc.github.io/blog/gsoc23/.

[2] E. A. Cherrat, I. Kerenidis, N. Mathur, J. Landman, M. Strahm, and Y. Y. Li. Quantum vision transformers. *Quantum*, 8:1265, Feb. 2024. ISSN 2521-327X. doi: 10.22331/q-2024-02-22-1265. URL http://dx.doi.org/10.22331/q-2024-02-22-1265.

[3] C. Chu, N.-H. Chia, L. Jiang, and F. Chen. Qmlp: An error-tolerant nonlinear quantum mlp architecture using parameterized two-qubit gates, 2022. URL https://arxiv.org/abs/2206.01345.

[4] M. Comajoan Cara, G. R. Dahale, Z. Dong, R. T. Forestano, S. Gleyzer, D. Justice, K. Kong, T. Magorsch, K. T. Matchev, K. Matcheva, and E. B. Unlu. Quantum vision transformers for quark–gluon classification. *Axioms*, 13(5), 2024. ISSN 2075-1680. doi: 10.3390/axioms13050323. URL https://www.mdpi.com/2075-1680/13/5/323.

[5] A. Dosovitskiy, L. Beyer, A. Kolesnikov, D. Weissenborn, X. Zhai, T. Unterthiner, M. Dehghani, M. Minderer, G. Heigold, S. Gelly, J. Uszkoreit, and N. Houlsby. An image is worth 16x16 words: Transformers for image recognition at scale, 2021.

[6] L. Huynh, J. Hong, A. Mian, H. Suzuki, Y. Wu, and S. Camtepe. Quantum-inspired machine learning: a survey, 2023. URL https://arxiv.org/abs/2308.11269.

[7] G. Li, X. Zhao, and X. Wang. Quantum self-attention neural networks for text classification, 2023.

[8] S. Lloyd, M. Schuld, A. Ijaz, J. Izaac, and N. Killoran. Quantum embeddings for machine learning, 2020.

[9] A. L. Maas, R. E. Daly, P. T. Pham, D. Huang, A. Y. Ng, and C. Potts. Learning word vectors for sentiment analysis, June 2011. URL http://www.aclweb.org/anthology/P11-1015.

[10] J. R. McClean, S. Boixo, V. N. Smelyanskiy, R. Babbush, and H. Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1), Nov. 2018. ISSN 2041-1723. doi: 10.1038/s41467-018-07090-4. URL http://dx.doi.org/10.1038/

s41467-018-07090-4.

[11] M. Schuld. Supervised quantum machine learning models are kernel methods, 2021.

[12] M. Schuld, V. Bergholm, C. Gogolin, J. Izaac, and N. Killoran. Evaluating analytic gradients on quantum hardware. *Physical Review A*, 99(3), Mar. 2019. ISSN 2469-9934. doi: 10.1103/physreva.99.032331. URL http://dx.doi.org/10.1103/PhysRevA.99.032331.

[13] S. Sim, P. D. Johnson, and A. Aspuru-Guzik. Expressibility and entangling capability of parameterized quantum circuits for hybrid quantum-classical algorithms. *Advanced Quantum Technologies*, 2(12), Oct. 2019. ISSN 2511-9044. doi: 10.1002/qute.201900070. URL http://dx.doi.org/10.1002/qute.201900070.

[14] R. D. Sipio, J.-H. Huang, S. Y.-C. Chen, S. Mangini, and M. Worring. The dawn of quantum natural language processing, 2021.

[15] X. Zhang, J. Zhao, and Y. LeCun. Character-level convolutional networks for text classification, 2016. URL https://arxiv.org/abs/1509.01626.