Quantum Neural Networks: Architecture Design and Quantum Training

Yidong Liao

MSc

Thesis submitted in fulfilment of the requirements for the degree of

Doctor of Philosophy

under the supervision of Chris Ferrie

Mar 2025

University of Technology Sydney

Faculty of Engineering and Information Technology

Certificate of Original Authorship

I, Yidong Liao, declare that this thesis is submitted in fulfilment of the requirements for the award of Doctor of Philosophy, in the Faculty of Engineering and Information Technology at the University of Technology Sydney.

This thesis is wholly my own work unless otherwise referenced or acknowledged. In addition, I certify that all information sources and literature used are indicated in the thesis.

This document has not been submitted for qualifications at any other academic institution.

This research is supported by the Australian Government Research Training Program.

Signature: Production Note:
Signature removed prior to publication.

5 Mar 2025

Abstract

Quantum Neural Networks (QNNs) are promising machine learning models with potential quantum advantages over classical neural networks. This thesis focuses on their architecture design, training methodologies, and certain applications, addressing three challenges in QNN research: overcoming barren plateaus in training, designing problem-specific QNN models, and tackling state-of-the-art classical machine learning models. The thesis is divided into three main parts, each targeting a specific challenge. The first part proposes quantum-optimization-powered training methods that exploit hidden structures in the QNN optimization problem to mitigate the barren plateau issue. The second part designs problem-tailored QNNs for graph-structured data, incorporating inductive biases into their architectures to enhance trainability and generalization. The third part explores the quantum implementation of Generative Pre-trained Transformers (GPT) — the original version of ChatGPT. By addressing these challenges, this thesis contributes to advancing the field of Quantum Machine Learning, offering new insights and methodologies for designing and training QNNs.

Contributions to the thesis

Yidong Liao conducted the research and wrote the thesis, Chris Ferrie supervised the research and thesis writing.

Acknowledgments

First and foremost, I would like to thank my supervisor, Chris Ferrie, for his invaluable intellectual guidance and financial support throughout my studies. His willingness to grant me the freedom to explore my interests, while ensuring I remained aligned with the conventional path, has been immensely beneficial. This balance has greatly contributed to my personal and academic development, and I am deeply grateful for his mentorship.

I would also like to express my gratitude to several individuals who have enriched my research journey. I wish to acknowledge Min-Hsiu Hsieh's advice during the first year of my Ph.D. My appreciation also goes to Oscar Dahsten, who supervised me as a visiting student before my Ph.D. and introduced me to the field of quantum machine learning, and to my former colleagues Daniel Ebler and Feiyang Liu, whose collaboration led to my first publication in this exciting field. My sincere thanks go to Xiaoming Zhang for helping to advance my research on quantum GNN after I submitted my Ph.D. thesis. My gratitude furtger extends to Yao-zhong Zhang and Ian Marquette, who supervised me during my Master's degree at the University of Queensland, where I achieved my debut publication in mathematical physics.

Above all, I thank my mom and dad for their years of love and support.

Order for the Remainder of the Thesis

- Dedications
- Table of Contents
- Lists of Figures and Tables
- Main text of the thesis
- Appendices
- Bibliography

To my parents

Contents

List of Figures

List of Tables

Chapter 1

Introduction

The emergence of Machine Learning (ML) marks one of the most significant scientific breakthroughs of the 20th century. Unlike traditional computing methods where tasks are accomplished through explicit programming by users, resulting in code that processes input to yield specific outputs, ML adopts a distinctly different methodology. In this paradigm, a computer system is trained to interpret and learn from data, aiming to effectively address problems when faced with new and unfamiliar scenarios [22]. Presently, ML finds extensive application across various scientific fields, including but not limited to the advancement of autonomous vehicles, drug discovery, and exploration of new materials [23].

The advancement of quantum computing has sparked significant interest in its application to ML, leading to the development of a new field called Quantum Machine Learning (QML) [24]. The primary objective of QML is to utilize quantum phenomena such as entanglement and superposition to solve ML problems more efficiently than classical algorithms executed on classical computers, gaining a quantum advantage. A QML model typically involves a data embedding process followed by a parameterized quantum circuit, frequently referred to as a Quantum Neural Network (QNN). Training this model involves the optimization of the parameters within the QNN [25].

Despite some promising results (e.g.[26, 25, 27]), QML is still in its infancy and facing many challenges [28]. In this thesis, we aim to address the following challenges of QML:

- **Challenge 1: Barren plateaus in training** Training quantum neural networks (QNNs) using gradient-based or gradient-free classical optimisation approaches is severely impacted by the presence of barren plateaus in the cost landscapes.[29]

- **Challenge 2: Designing problem-specific QNN model** Recent research [30] has indicated that employing problem-agnostic QNN architectures can result in significant

issues that negatively affect the performance of the QML models, Therefore, there is a need for designing problem-tailored QNN model.

- **Challenge 3: Addressing state-of-the-art classical ML models** Current QML are still far from state-of-the-art classical ML for real world applications[31], efforts need to be made towards tackling state-of-the-art classical ML models.

The thesis consists of three parts, each addressing one of the above challenges:

- **Thesis Part 1: Quantum Training of QNNs** consists of Chapter 3 4 5 In this part of the thesis, we devise a framework for utilising quantum-optimisation powered training methods to find optimal parameters of QNNs for certain tasks. Our quantum training methods of QNNs exploits hidden structure in the QNN optimisation problem and is expected to mitigate the barren plateau issue.
- **Thesis Part 2: QNN Architecture for graph-structured data** consists of Chapter 6 7 8 9 In this part of the thesis, we design problem-tailored QNNs for graph-structured data by incorporating the inductive biases into their architectures.
- **Thesis Part 3: GPT on Quantum Computer** consists of Chapter 10 11 In this part of the thesis, we explore the quantum implementation of GPT — the original version of the state-of-the-art language models such as ChatGPT.

We outline each part of the thesis in the rest of this chapter.

1.1 Quantum Training of QNNs

A QNN consists of a set of parameterized quantum gates within a predefined circuit ansatz. The design of the ansatz together with the value of the gate parameters determine the outcome of the QNN. In order to successfully perform certain tasks, QNNs must be trained to find optimal parameters for generating desired outcomes.

In the majority of QNN research, the training is carried out by employing variational hybrid quantum-classical algorithms [32], in which the parameters are optimized by a classical optimizer using gradient-based or gradient-free approaches. Though an ever increasing amount of effort is being put into QNN research, there is evidence that they will be difficult to train due to flat optimisation landscapes called barren plateaus [29].

In the first part of the thesis, we devise a framework for leveraging quantum optimisation algorithms to find optimal parameters of QNNs for certain tasks. To cast the optimisation problem of training QNN into the context of quantum optimisation, the parameters in QNN are quantised — moved from being classical to being stored in quantum registers which are

in addition to those upon which the QNN is performing its computation. We then coherently encode the cost function of QNNs onto relative phases of a superposition state in the Hilbert space of the QNN parameters. The parameters are tuned with an iterative quantum optimisation structure using adaptively selected Hamiltonians. The quantum mechanism of this framework could exploit hidden structure in QNN optimisation problem, hence are expected to provide beyond-Grover speed up and mitigate the Barren Plateau issues for training QNN.

The applications of our framework include the training of Variational Quantum Eigensolvers (VQE) and data-driven quantum machine learning models such as Variational Quantum Classifier. As an example, we present the circuit construction of our quantum training for VQE in Figure 1.1. An animation of the circuit construction from scratch is available at https://youtu.be/c4JRVza0AAw

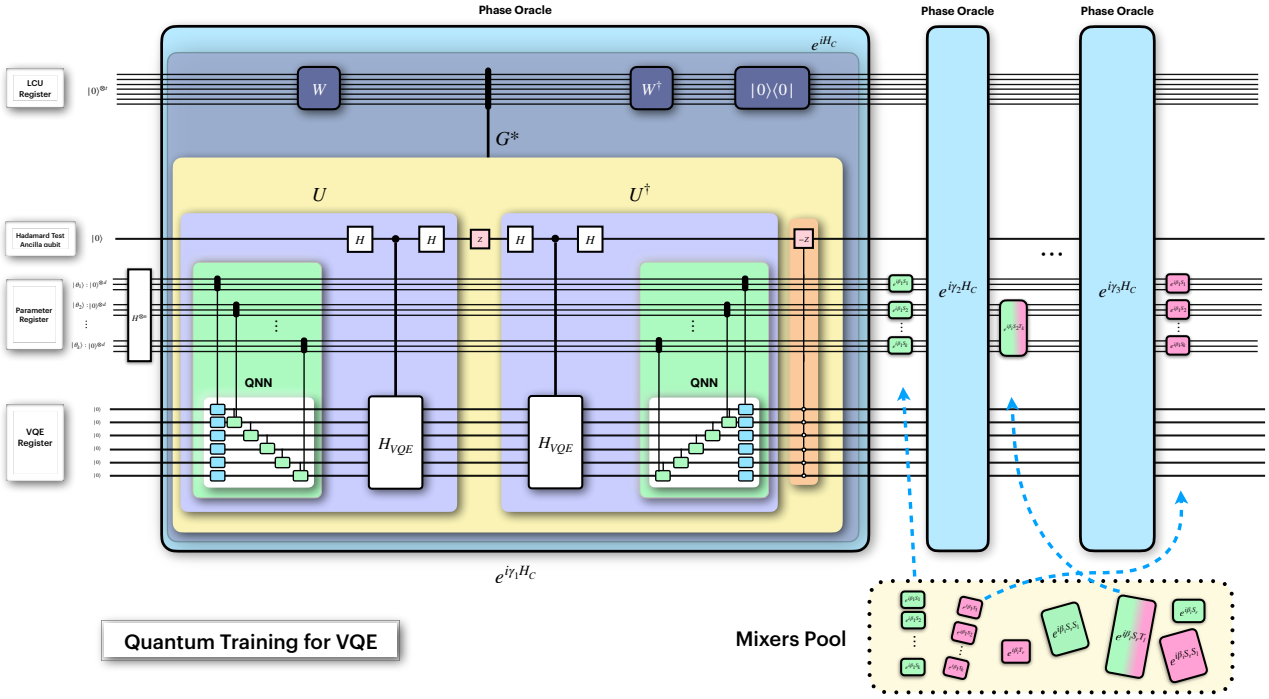

Figure 1.1: *Schematic of Our quantum training algorithm for VQE.* Here we use the training of VQE as an example, to present the schematic circuit construction of our quantum training algorithm for QNN. A video animation of the circuit construction is available at https://youtu.be/RVWkJZY6GNY. (This is vector image and best view with the zoom feature in standard PDF viewers.) Note: *1.* In all figures of this Paper, we omit the minus signs in all time-evolution-like terms (i.e. exponential of a Hamiltonian e^{-iHt}) for sake of brevity and space. *2.* Some quantum registers are not depicted in this figure due to the limitation of space.

1.2 QNN Architectures for graph-structured data

Most conventional problem-agnostic QNN architectures do not exploit the underlying structure of data for specific problems, which leads to issues on trainability and generalization [30]. In

the second part of the thesis, we design problem-tailored QNNs for graph-structured data by incorporating the inductive biases into their architectures. Specifically, we devise QNN architectures in accordance with three major types of classical Graph Neural Networks(GNNs): Graph Convolutional Neural Networks, Graph Attention Neural Networks, and Message-Passing GNNs. A brief introduction of the three classical GNNs is given in section 6.1. Fig.1.2 shows the illustration of the overall circuit construction for the three Quantum GNN architectures along with the three fundamental types of classical GNNs [1].

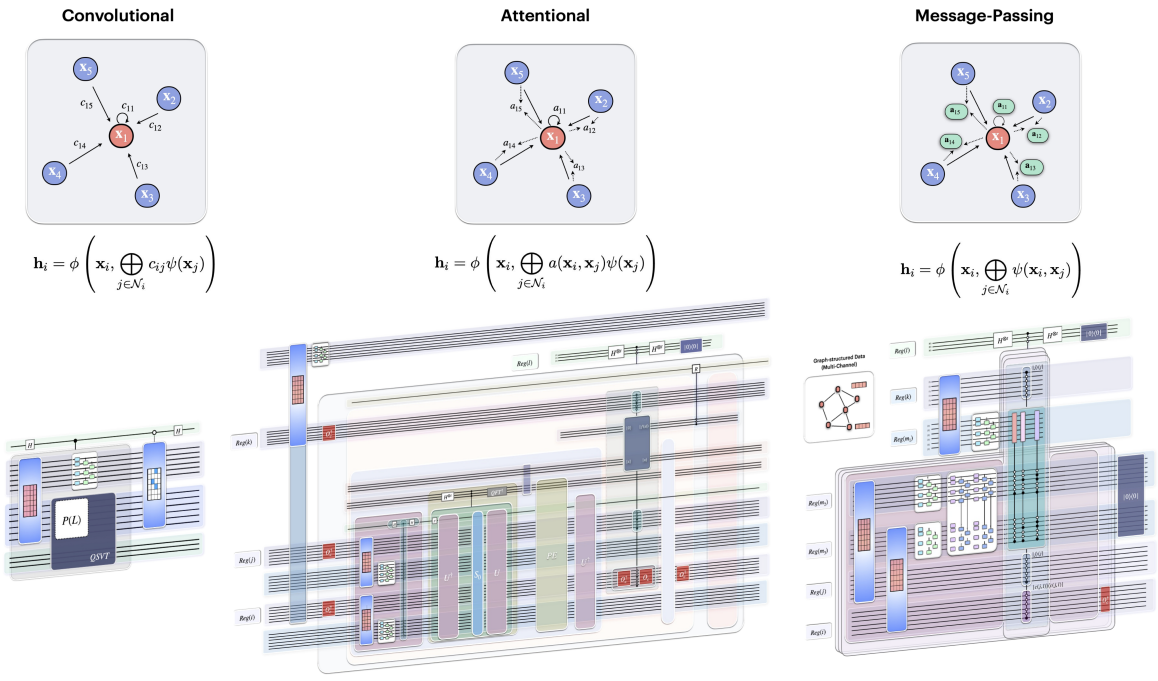

Figure 1.2: *Overall circuit construction for the three Quantum GNN architectures along with the three fundamental types of classical GNNs [1].*

Potential advantages of our QNNs can be viewed from two perspectives: (1) Compared to conventional QNNs, the number of parameters in our QNNs could be significantly reduced, improving trainability of the model. (2) Compared to the classical GNNs, our QNNs for graph-structured data could have better scalability and expressivity.

1.3 GPT on Quantum Computer

Large Language Models (LLMs) have rapidly gained prominence and made a profound impact worldwide, transforming how we interact with and understand the capabilities of artificial intelligence. The success of models like GPT-4 has showcased the immense potential of LLMs in various applications across different domains. In the domain of natural language processing (NLP), LLMs are adept at various tasks including machine translation, sentiment analysis,

question answering, and text summarization, among others. They have demonstrated a high level of effectiveness in identifying complex patterns in language, comprehending the context, and producing text that is both coherent and contextually appropriate.

Despite rapid evolution and notable progress in the field of Quantum Machine Learning, research connecting QML advancements to these state-of-the-art language models is still in its infancy. Recent studies (e.g., Ref. [33, 34, 35, 36, 37]) highlight a growing interest in leveraging quantum computing to elevate the capabilities of language models. In the third part of the thesis, we explore the quantum implementation of GPT — the original version of ChatGPT. Fig.1.3 provides a preview/summary of this part of the thesis: the overall quantum implementation of GPT.

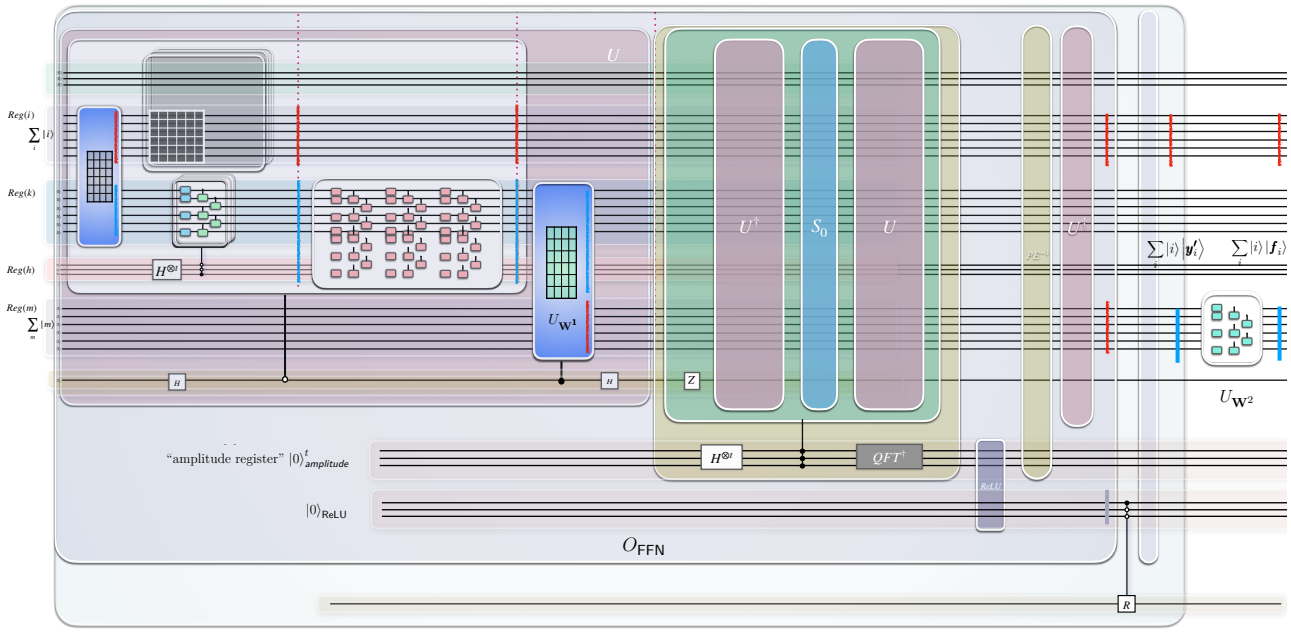

Figure 1.3: *Preview/summary of the third part of the thesis: overall quantum implementation of GPT*

Chapter 2

Preliminary

2.1 Quantum Neural Networks

Classical neural networks are fundamentally built on the structure of multi-layer perceptrons which involve layers of trainable linear transformations and element-wise non-linear transformations (activation functions such as ReLU, sigmoid, or tanh).[1] On the other hand, Quantum Neural Networks (QNNs), which are often defined as parametrized quantum circuits with a predefined circuit ansatz, do not naturally exhibit this kind of structure. In QML literature, a QNN, denoted as $U(\vec{\theta})$, often have a has an L-layered structure of the form [38]

$$U(\vec{\theta}) = \prod_{l=1}^{L} U_l(\vec{\theta}_l), \quad U_l(\vec{\theta}_l) = \prod_{k=1}^{K} e^{-i\theta_{lk}H_k}, \tag{2.1}$$

where the index l represents the layer, and the index k covers the Hermitian operators H_k that generates the unitaries in the circuit ansatz, $\vec{\theta}_l = (\theta_{l1}, \ldots \theta_{lK})$ represents the parameters in a single layer, and $\vec{\theta} = \{\vec{\theta}_1, \ldots, \vec{\theta}_L\}$ represents the collection of adjustable parameters in the QNN. Examples of circuit ansatz represented by Eq. 2.1 include: the hardware-efficient ansatz [39], quantum alternating operator ansatz [2], and quantum optimal control Ansatz [40], among others.

The emulation of classical perceptrons with non-linearities in quantum circuits is an area of active research. There are several proposals for how this might be achieved [41, 42, 43], but they often involve intricate methods of encoding information into quantum states and performing measurements. The difficulty arises from the need to replicate the non-linear characteristics of classical neural networks within the linear framework of quantum mechanics. Nevertheless, it is still not clear what benefits these somewhat cumbersome quantum implementations could provide compared to the straightforward classical perception.

[1] we assume the readers of this thesis are familiar with classical neural networks. For reference of classical neural networks, see e.g. [22]

In the second and third parts of this thesis, we utilize the conventional QNN as in Eq. 2.1 for implementing some trainable linear transformations in classical neural networks(often referred as "linear layer", or "fully connected layer"), and we extend the conventional notion of a QNN to "a broad structured quantum circuit contains parameterized quantum circuits (represented by Eq. 2.1) as its components", that is, parameterized quantum circuits (which contain only parameterized gates) sit within a broader non-parameterized quantum circuit.

Parametrized quantum circuit for implementing trainable linear layer

In machine learning, a linear layer[22] is a fundamental component of neural networks architectures that maps input vectors to output vectors through affine transformation: Given an input vector $\mathbf{x} \in \mathbb{R}^n$, the linear layer transforms it to an output vector $\mathbf{y} \in \mathbb{R}^m$ using a weight matrix $W \in \mathbb{R}^{m\times n}$ and an optional bias vector $\mathbf{b} \in \mathbb{R}^m$ as $\mathbf{y} = W\mathbf{x} + \mathbf{b}$, and W, b contain the trainable parameters.

In quantum case, the input vector $\mathbf{x}$ can be encoded in the amplitudes of a quantum state $|\psi_x\rangle$ as

$$|\psi_x\rangle = \sum_{i=1}^{n} x_i |i\rangle$$

where x_i is the i-th element of $\mathbf{x}$, and $|i\rangle$ is the i-th computational basis state. Applying a parameterized quantum circuit $U(\vec{\theta}) \in \mathbb{C}^{n\times n}$ on $|\psi_x\rangle$ be interpreted as a special linear transformation (represented as a square matrix) on $\mathbf{x}$. By omitting the optional bias vector b and setting $m = n$, we can utilize parameterized quantum circuit as in Eq. 2.1 to implement a special type of the trainable linear layers $\mathbf{y} = W\mathbf{x} + \mathbf{b}$. For the case where the weight matrix $W \in \mathbb{R}^{m\times n}$ is rectangular, e.g. $m < n$[2], we can adjust the dimension of the output vector to be the same as the input vector in the neural network architecture, or we can consider the output of the parameterized quantum circuit $U(\vec{\theta})|\psi_x\rangle$ as the concatenation of $\mathbf{y} \in \mathbb{R}^m$ and another vector and $W \in \mathbb{R}^{m\times n}$ being a part of the unitary $U(\vec{\theta}) \in \mathbb{C}^{m\times m}$.

In the second and third parts of this thesis, we utilize parameterized quantum circuits as in Eq. 2.1 for implementing some of the trainable linear layers. Note that in case where the weight matrix $W \in \mathbb{R}^{m\times n}$ is rectangular ($m \neq n$), by default we adjust the dimension of the output vector to be the same as the input vector in our QNN architecture, without specifying the adjustment in the prior description of the classical architecture.

It should be emphasized that the trainable circuit parameters $\vec{\theta}$ are not equivalent to the weights in the weight matrix, but rather they are parameterized by them, as in $W(\vec{\theta})$. Ref [38] contains a discussion of the parametrization.

[2] for case where $m > n$, see an example in section 11.6

2.2 Quantum Optimisation Algorithms

Zoo of Quantum Optimisation Algorithms

For completeness, we list some typical quantum optimisation algorithms in Table. 2.1, including the primitive ones (adiabatic, quantum walks, QAOA, Grover adaptive search), their hybridizations [19] [44, 45, 46, 47], and their variants [48, 49] [50] [5, 20] [21, 3] [51]. In this paper, for the training of QNNs, we focus on utilising QAOA and its variants as well as Grover adaptive search, which we will review in the following subsections.

Primitives	Adiabatic	Quantum Walk	QAOA	Grover adaptive search
Hybridization of Primitives	Hybrid adiabatic–quantum-walk algorithms, others			
Variants of Primitives	Shortcut to adiabaticity	Quantum stochastic walk	Adaptive QAOAs, Others	Quantum basin hopping

Table 2.1: *Zoo of Quantum Optimisation Algorithms.* The first row contains the four primitive quantum optimisation algorithms by adiabatic quantum evolution, quantum walks, QAOA and Grover adaptive search. The second row contains the hybridization among these four primitives, e.g. hybrid adiabatic–quantum-walk algorithms [19]. The third row contains the variants of the primitives, e.g. variants of QAOA include Adaptive QAOAs [5, 20], and others [21, 3].

QAOA and its variants

In this section, we review the original quantum approximation optimization algorithm (QAOA) proposed in Ref. [52] and its variants. Consider an unconstrained optimization problem on n-bit strings $\mathbf{z} = (z_1, z_2, z_3, ...z_n)$ where $z_i \in \{0, 1\}$ We seek the optimal bit string $\mathbf{z}$ that maximizes (or minimizes) a cost function $C(\mathbf{z})$. Given the cost function $C(\mathbf{z})$ of a problem instance, the algorithm is characterized by two Hamiltonians: the *cost Hamiltonian* H_C and the *Mixing Hamiltonian* H_{M}. The cost Hamiltonian H_C encodes the cost function $C(\mathbf{z})$ to be optimized, and acts on n-qubit computational basis states as

$$H_C \left|\mathbf{z}\right\rangle = C(\mathbf{z}) \left|\mathbf{z}\right\rangle .$$

The mixing Hamiltonian H_{M} is chosen as to be

$$H_{\mathrm{M}} = \sum_{j=1}^{n} X_j,$$

where X_j is the Pauli X operator acting on the jth qubit. The initial state is the even superposition state of all possible solutions: $\left|s\right\rangle = \frac{1}{\sqrt{2^n}} \sum_{\mathbf{z}} \left|\mathbf{z}\right\rangle$. The QAOA algorithm consists of

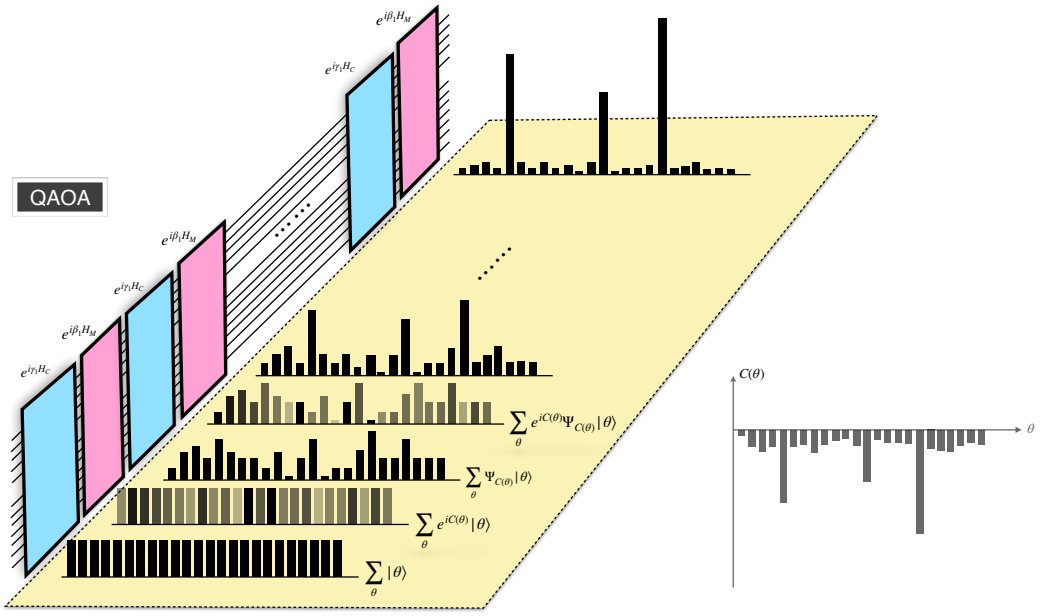

Figure 2.1: *Interference process of QAOA.* QAOA is an interference-based algorithm such that non-target states interfere destructively while the target states interfere constructively. Here we illustrate this interference process by presenting the evolution of the quantum state of the parameters(black bar graphs on the yellow plane) alongside with the QAOA operations(blue and pink boxes on circuit lines, representing the Phase encoding and Mixers respectively). The starting state $\sum_\theta |\theta\rangle$ (omitting the normalization factor) is the even superposition state of all possible parameter configurations. After the first Phase encoding operation, the state becomes $\sum_\theta e^{-iC(\boldsymbol{\theta})}|\theta\rangle$ for which we use the opacity of the bars to indicate the value of the phase, the magnitudes of the amplitudes in the state remains unchanged. After the first Mixer, the state becomes $\sum_\theta \Psi_{C(\boldsymbol{\theta})}|\theta\rangle$ in which the magnitudes of the amplitudes in the state has changed. A similar process happens to the following operations, until the amplitudes of the optimal parameter configurations are amplified significantly(the furthest bar graph). The grey bar graph in the right corner is the cost function being optimized by QAOA.

alternating time evolution under the two Hamiltonians H_C and H_{M} for p rounds, where the duration in round j is specified by the parameters γ_j and β_j, respectively. After all p rounds, the state becomes

$$|\boldsymbol{\beta},\boldsymbol{\gamma}\rangle = e^{-i\beta_p H_{\mathrm{M}}} e^{-i\gamma_p H_C} \ldots e^{-i\beta_2 H_{\mathrm{M}}} e^{-i\gamma_2 H_C} e^{-i\beta_1 H_{\mathrm{M}}} e^{-i\gamma_1 H_C} |s\rangle .$$

The alternating operations can be illustrated as in Fig. 2.2. Finally, a measurement in the computational basis is performed on the state. Repeating the above state preparation and measurement, the expected value of the cost function,

$$\langle C\rangle = \langle \boldsymbol{\beta},\boldsymbol{\gamma}| H_C |\boldsymbol{\beta},\boldsymbol{\gamma}\rangle ,$$

can be estimated from the samples produced from the measurements.

The above steps are then repeated altogether, with updated sets of time parameters $\gamma_1,\ldots,\gamma_p,\beta_1,\ldots,\beta_p$. Typically a classical optimization loop (such as gradient descent) is used

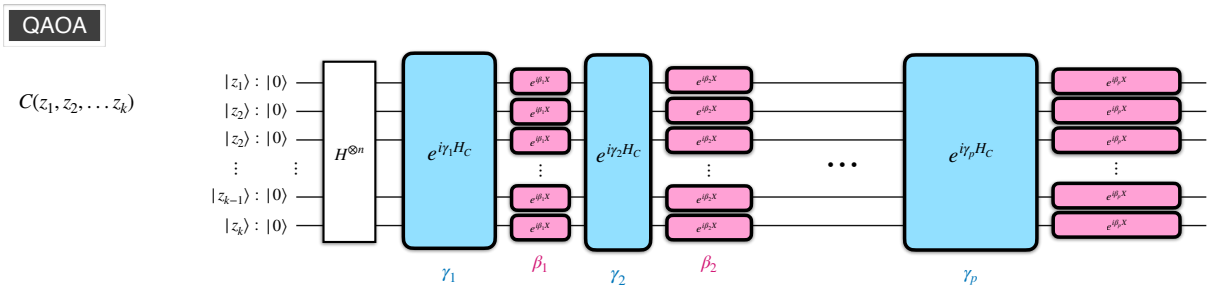

Figure 2.2: *Quantum circuit schematic of the operations in the original QAOA.* The state is initialized by applying Hadamard gates on each qubit, represented as $H^{\otimes n}$. This results in the equal superposition state of all possible solutions. QAOA consists of alternating time evolution under the two Hamiltonians H_C and H_{M} for p rounds, where the duration in round j is specified by the parameters γ_j and β_j, respectively. In the original QAOA, the mixing Hamiltonian H_{M} is chosen as to be $H_{\mathrm{M}} = \sum_{j=1}^{n} X_j$, After all p rounds, the state becomes $|\boldsymbol{\beta}, \boldsymbol{\gamma}\rangle = e^{-i\beta_p H_{\mathrm{M}}} e^{-i\gamma_p H_C} \ldots e^{-i\beta_2 H_{\mathrm{M}}} e^{-i\gamma_2 H_C} e^{-i\beta_1 H_{\mathrm{M}}} e^{-i\gamma_1 H_C} |s\rangle$.

to find the optimal parameters that maximize(or minimize) the the expected value of the cost function $\langle C \rangle$. Then measuring the resulting state of the optimal parameters provide an approximate solution to the optimization problem.

Here we present some remarks on the fundamental differences of the adiabatic and QAOA protocols. QAOA can be seen as a "trotterized" version of adiabatic evolution: the mixer Hamiltonians being the initial Hamiltonian in the analogous adiabatic algorithm, and the cost Hamiltonians being the final Hamiltonian. However short-depth QAOA is not really the digitized version of the adiabatic problem, but rather an ad hoc ansatz. In Ref. [53] it is shown that QAOA is able to deterministically find the solution of specially constructed optimization problems in cases where quantum annealing fail. We emphasise that QAOA is an interference-based algorithm such that non-target states interfere destructively while the target states interfere constructively. In Fig.2.1 we depict this interference process of QAOA.

There has been a lot of progress on QAOA recently on both the experimental and theoretical fronts. There is evidence suggesting that QAOA may provide a significant quantum advantage over classical algorithms [54, 55], and that it is computationally universal [56, 57]. Despite these advances, there are limitations of QAOA. The performance improves with circuit depth, but circuit depth is still limited in near-term quantum processors. Moreover, deeper circuits translate into more variational parameters, which introduces challenges for the classical optimizer in minimizing the objective function. Ref. [58] show that the locality and symmetry of QAOA can limit its performance. These issues can be attributed to the form of the QAOA ansatz. A short-depth ansatz that is further tailored to a given combinatorial problem could therefore address the issues with the standard QAOA ansatz. However, identifying such an alternative is

a highly non-trivial problem given the vast space of possible ansatzes. Farhi et al. [59] allowed the mixer to rotate each qubit by a different angle about the x-axis and modified the cost Hamiltonian based on hardware connectivity. This modification was made primarily out of hardware capability concerns with the hope that superior-than-classical performance can be experimentally verified.

LH-QAOA. In Ref. [2] Hadfield et al. considered alternative mixers including entangling ones on two qubits. The selection of mixers is based on the criteria of preserving the relevant subspace for the given combinatorial problem, for which they entitled it Local Hamiltonian-QAOA (LH-QAOA). Here we depict the quantum circuit schematic of LH-QAOA in Fig. 2.3.

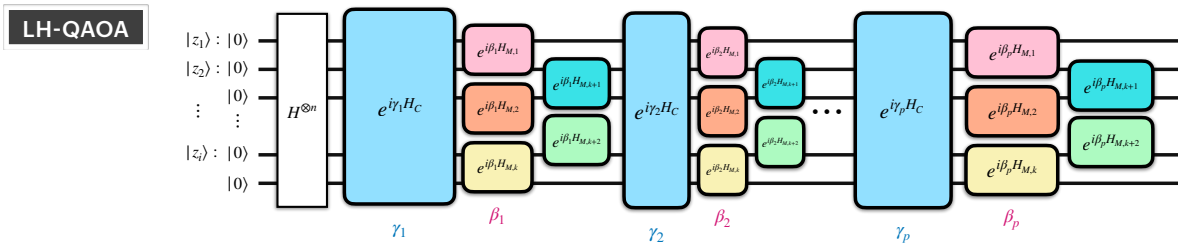

Figure 2.3: *Quantum circuit schematic of the operations in LH-QAOA*. The overall process of LH-QAOA is similar to that of the original QAOA in Fig. 2.2, where the difference is that the mixer of LH-QAOA contains entangling an mixer Hamiltonian on two qubits. These are represented by the $H_{M,i}$ blocks with various colors in the figure. Note that in order to avoid an excessive amount of hyper-parameters, Hadfield et al. [2] choose the β_j for each $H_{M,i}$ to be the same in every layer.

QDD. In Refs. [60, 3] Verdon et al. adjusted the mixers for continuous optimization problem in which the parameters to be optimized are continuous variables. In the original QAOA ansatz, the mixer is chosen to be single-qubit X rotations applied on all qubits. These constitute an uncoupled sum of generators of shifts in the computational basis. Similarly, the appropriate mixers in the continuous case should shift the value for each digitized continuous variables stored in independent registers. They entitled it Quantum Dynamical Descent (QDD). Here we depict the quantum circuit schematic of QDD in Fig. 2.4.

ADAPT-QAOA. LH-QAOA and QDD showcase the potential of problem-tailored mixers, but do not provide a general strategy for choosing mixers for different optimization problems. In Ref. [5] Zhu et al. replaced the fixed mixer H_M by a set of different mixers A_k that change from layer to layer. They entitled this variation of QAOA as ADAPT-QAOA. This adaptive approach would dramatically shorten the depth of QAOA layers while significantly improving the quality of the solution. Here we depict the quantum circuit schematic of ADAPT-QAOA in Fig. 2.5.

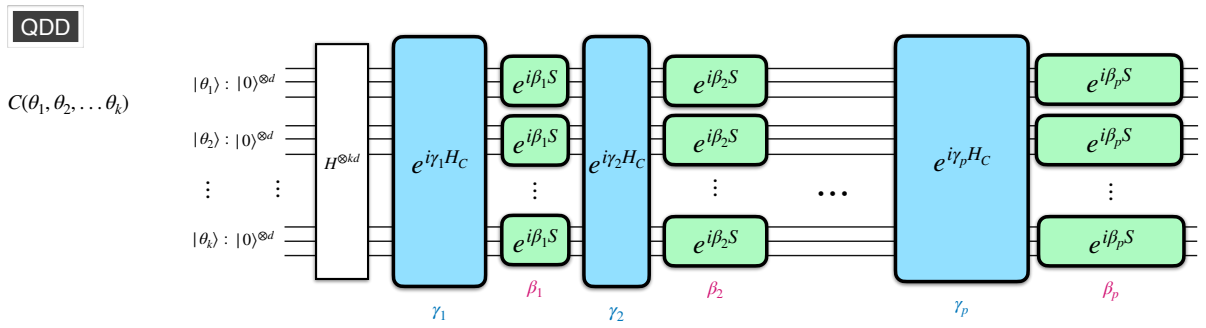

Figure 2.4: *Quantum circuit schematic of QDD.* QDD solves optimization problems of continuous variable. In this figure, θ_i are the continuous variables to be optimized in the training, where each θ_i is digitized into binary form and stored in an independent register. The overall process of QDD is similar to that of the original QAOA, where the difference is that the mixer of QDD with Hamiltonian S is acting on the registers of θ_i (rather than single qubits as in the original QAOA). The effect of the mixer in QDD is to shift the value for each θ_i.

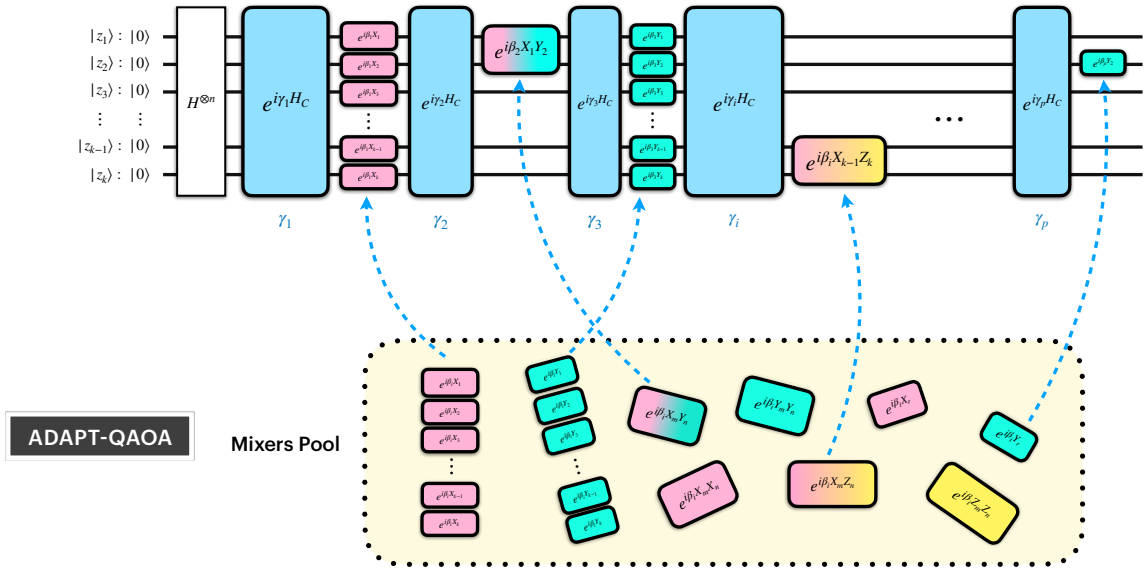

Figure 2.5: *Quantum circuit schematic of ADAPT-QAOA.* The overall process of LH-QAOA is similar to that of the original QAOA in Fig. 2.2, where the difference is that the mixer of LH-QAOA contains variable mixers taken from a *mixers pool*. Define Q to be the set of qubits. The mixer pool of ADAPT-QAOA is $P_{\text{ADAPT-QAOA}} = \cup_{i\in Q}\left\{X_i, Y_i, \sum_{i\in Q} X_i, \sum_{i\in Q} Y_i\right\} \cup_{i,j\in Q\times Q} \{B_i C_j | B_i, C_j \in \{X, Y, Z\}\}$.

Compared to the original QAOA, allowing Y mixers and entangling mixers enables ADAPT-QAOA to dramatically improve algorithmic performance while achieving rapid convergence for problems with complex structures. This effect of the adaptive mechanism is illustrated in Fig. 2.6.

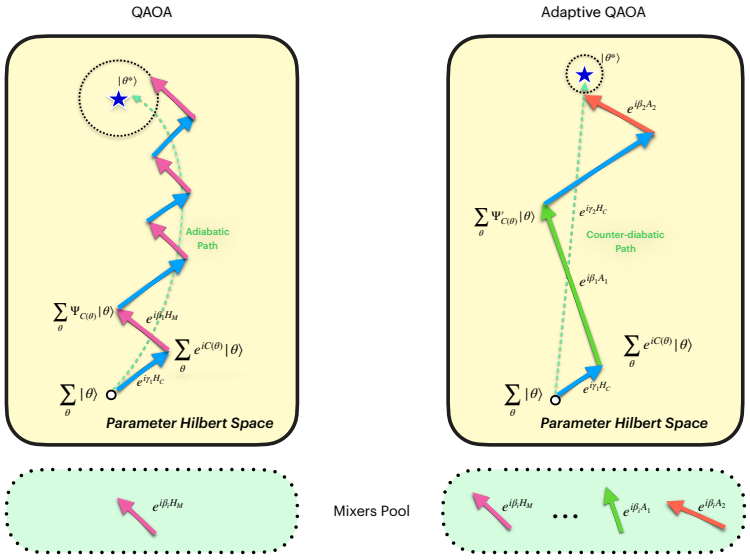

Figure 2.6: *Comparison of original QAOA and ADAPT-QAOA*. In the left and right panels of this figure, we depict the state change in the Hilbert space of the parameters to be optimized, for the original QAOA and ADAPT-QAOA respectively. The starting state $\sum_\theta |\theta\rangle$ (omitting the normalization factor), represented by the rounded dot at the bottom of each space, is the even superposition state of all possible solutions. The arrows represent the state evolution generated by the cost Hamiltonian and mixer Hamiltonian, and the color and direction of the arrows indicate the nature of the evolution. Blue arrows represent the evolution by the cost Hamiltonian. Arrows of other colors represent the evolution by different mixer Hamiltonians. In the original QAOA, there is only one mixer (shown in pink) available. Whereas, in ADAPT-QAOA there are more alternative mixers to chose from the mixers pool. The two algorithm try to reach the target state $|\theta^*\rangle$ (represented by the blue star) by stacking these arrows, which represent the alternating operations of two QAOAs. For reference we sketched the relevant paths — adiabatic path for the original QAOA and counter-diabatic path for ADAPT-QAOA — along the state evolution of the two QAOAs. As can be seen, the ADAPT-QAOA takes much fewer iterations to reach a closer point to the target state. This illustrates that compared to the original QAOA, allowing alternative mixers enables ADAPT-QAOA to dramatically improve algorithmic performance while achieving rapid convergence.

The advantage of the adaptive ansatz may come from the counter-diabatic (CD) driving mechanism. Numerical evidence shows that the adaptive mixer sequence chosen by the algorithm coincides with that of "shortcut to adiabaticity" by CD driving [5]. Inspired by the CD driving procedure, another variant of QAOA, CD-QAOA [20], also uses an adaptive ansatz to achieve similar advantages. CD-QAOA is designed for preparing the ground state of quantum-chaotic many-body spin chains. By using terms occurring in the adiabatic gauge potential as additional control unitaries, CD-QAOA can achieve fast high-fidelity many-body control.

AC-QAOA Inspired by above variants of QAOA, we design a new variant of QAOA tailored for our QNN training problem. In our case, for QNN training, the parameters we are optimizing (the angles of rotation gates) are continuous (real) values. Therefore, the choice of

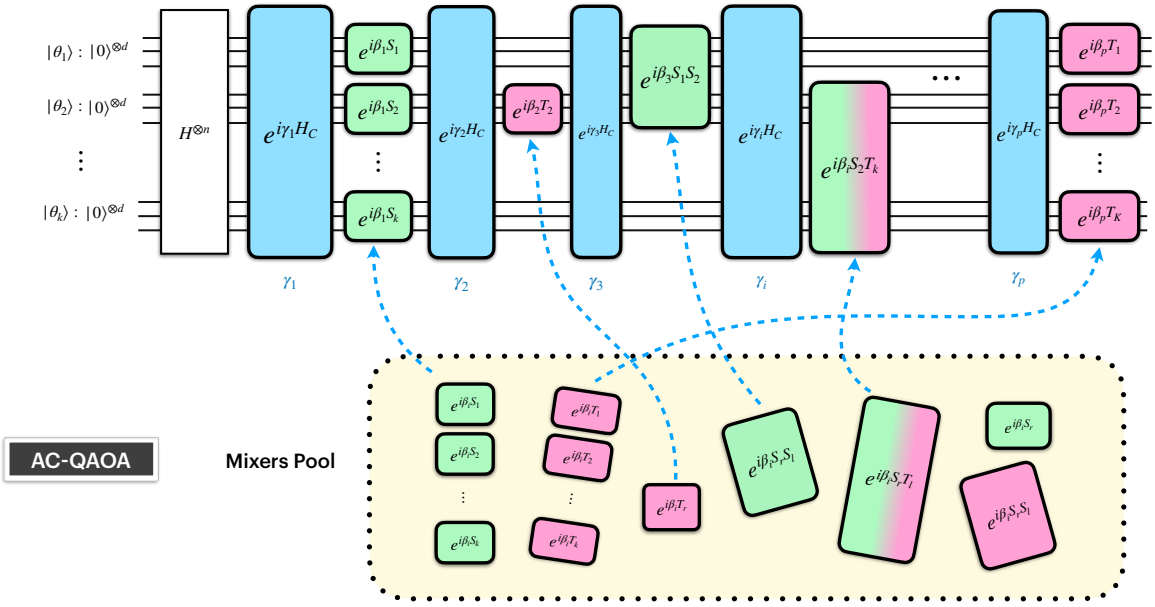

Figure 2.7: *Quantum circuit schematic of AC-QAOA.* AC-QAOA is a variant of QAOA we designed for solving optimisation of continuous variables with the short-depth advantage of QAOA layers. In this figure, θ_i are the continuous variables to be optimized in the training. Each θ_i is digitized into binary form and stored in an independent register. The overall process of AC-QAOA is similar to that of the original QAOA, with the difference being as follows. *1.* The mixers of AC-QAOA with Hamiltonians S_i and T_i are acting on the registers of θ_i (rather than single qubits as in the original QAOA). *2.* The mixers of AC-QAOA contain alternative mixers taken from a *mixers pool* and can vary from layer to layer.

mixer Hamiltonian has to be adapted (as in QDD). We also want take advantage of including alternative mixers and allowing adaptive mixers for different layers (as in ADAPT-QAOA). Thus, the proper QAOA ansatz for our QNN training problem should be an adaptive continuous version of QAOA, which we call *AC-QAOA*. Fig. 2.7 depicts the the quantum circuit schematic of AC-QAOA.

Grover Adaptive Search

Grover's algorithm is generally used as a search method to find a set of desired solutions from a set of possible solutions. Dürr and Høyer presented an algorithm based on Grover's method that finds an element of minimum value inside an array of N elements using on the order of $O(\sqrt{N})$ queries to the oracle [61]. Baritompa et al. [62] applied Grover's algorithm for global optimization, which they call Grover Adaptative Search (GAS). GAS has been applied in training classical neural networks [12] and polynomial binary optimization [63]. In the following we outline GAS.

Consider a function $f : X \to \mathbb{R}$, where for ease of presentation assume $X = \{0,1\}^n$. We are interested in solving $\min_{x \in X} f(x)$. The main idea of GAS is to construct an "adaptive" oracle for a given threshold y such that it flags all states $x \in X$ satisfying $f(x) < y$, namely the oracle marks a solution x if and only if another boolean function g_y satisfies $g_y(x) = 1$, where

$$g_y(x) = \begin{cases} 1 & \text{if} \quad f(x) < y \\ 0 & \text{otherwise} \end{cases}, \tag{2.2}$$

The oracle $\mathcal{O}_{Grover}$ then act as

$$\mathcal{O}_{Grover} |x\rangle = (-1)^{g_y(x)} |x\rangle . \tag{2.3}$$

We use Grover search to find a solution $\tilde{x}$ with a function value better than y. Then we set $y = f(\tilde{x})$ and repeat until some formal termination criteria is met — for example, based on the number of iterations, time, or progress in y.

2.3 Swap test, Hadamard test, and the Grover operator

Swap Test and its Grover operator

Let $|p_j\rangle, |t\rangle$ be the resulting quantum states of unitary operators P_j and T, respectively — that is, $|p_j\rangle = P_j|0\rangle^{\otimes n}$ and $|t\rangle = T|0\rangle^{\otimes n}$. The swap test is a technique that can be used to estimate $|\langle p_j|t\rangle|^2$ [64]. The circuit of Swap test is shown in Fig. 2.8.

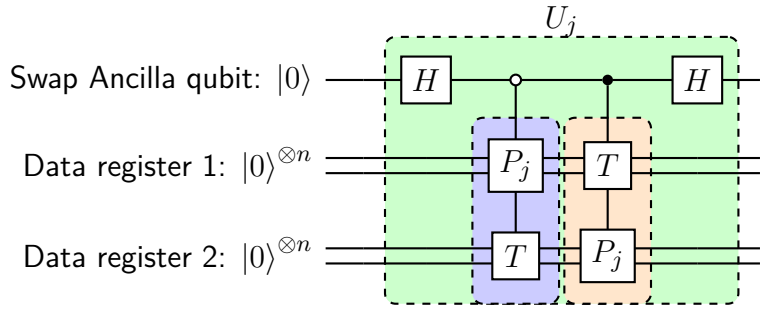

Figure 2.8: *Circuit diagram of Swap test.* Here we present an alternative form of swap test: instead of applying the swap operation on two quantum states, the circuit in this figure simulate the "swap" effect by applying two unitaries P_j, T on two registers in different order controlled by an ancilla qubit. The "anti-control" symbol is defined as: when the control qubit is in state $|0\rangle$, the unitary being controlled is executed; when the control qubit is in state $|1\rangle$, the unitary being controlled is not executed.

We denote the unitary of the Swap test circuit (dotted green box in Fig. 2.8) as U_j, which can be written as

$$U_j \coloneqq [H \otimes I \otimes I] \cdot [|0\rangle\langle 0| \otimes P_j \otimes T + |1\rangle\langle 1| \otimes T \otimes P_j] \cdot [H \otimes I \otimes I]. \tag{2.4}$$

The output state from U_j is denoted as $|\phi_j\rangle$:

$$|\phi_j\rangle = \frac{1}{\sqrt{2}}(|+\rangle|p_j\rangle|t\rangle + |-\rangle|t\rangle|p_j\rangle). \tag{2.5}$$

Rearranging the terms we have

$$|\phi_j\rangle = \frac{1}{2}\Big(|0\rangle \otimes (|p_j\rangle|t\rangle + |t\rangle|p_j\rangle) + |1\rangle \otimes (|p_j\rangle|t\rangle - |t\rangle|p_j\rangle)\Big). \tag{2.6}$$

We then define $|u_j\rangle$ and $|v_j\rangle$ as the normalized states of $|p_j\rangle|t\rangle + |t\rangle|p_j\rangle$ and $|p_j\rangle|t\rangle - |t\rangle|p_j\rangle$ respectively. Then there is a real number $\theta_j \in [\pi/4, \pi/2]$ such that

$$|\phi_j\rangle = \sin\theta_j|0\rangle|u_j\rangle + \cos\theta_j|1\rangle|v_j\rangle, \tag{2.7}$$

where θ_j satisfies $\cos\theta_j = \sqrt{1-|\langle p_j|t\rangle|^2}\,/\sqrt{2}$, $\sin\theta_j = \sqrt{1+|\langle p_j|t\rangle|^2}\,/\sqrt{2}$, therefore we have:

$$|\langle p_j|t\rangle|^2 = -\cos 2\theta_j. \tag{2.8}$$

From Eq. 2.8 and Eq. 2.7 we see that the value of $|\langle p_j|t\rangle|^2$ is encoded in the amplitude of the output state $|\phi_j\rangle$ of swap test. This will be used in the amplitude encoding of QNN cost function which is a crucial component of the quantum training.

Applying the Schmidt decomposition method to state $|\phi_j\rangle$ we find

$$|\phi_j\rangle = \frac{-\boldsymbol{i}}{\sqrt{2}}(e^{\boldsymbol{i}\theta_j}|w_+\rangle_j - e^{-\boldsymbol{i}\theta_j}|w_-\rangle_j), \tag{2.9}$$

where $|w_\pm\rangle_j = \frac{1}{\sqrt{2}}(|0\rangle|u_j\rangle \pm \boldsymbol{i}|1\rangle|v_j\rangle)$.

One can construct a Grover operator using U_j as follows:

$$G_j := (I^{\otimes(2n+1)} - 2|\phi_j\rangle\langle\phi_j|)(Z \otimes I^{\otimes 2n}), \tag{2.10}$$

$$= U_j C U_j^\dagger (Z \otimes I^{\otimes 2n}), \tag{2.11}$$

where $Z = |0\rangle\langle 0| - |1\rangle\langle 1|$ is the Pauli-Z operator, $C = I^{\otimes(2n+1)} - 2|0\rangle^{\otimes(2n+1)}\langle 0|^{\otimes(2n+1)}$ can be implemented as the circuit shown in Fig. 2.9. The circuit representation of G_j is shown in Fig. 2.10.

It is easy to check that $|w_\pm\rangle_j$ are the eigenstates of G_j. — that is,

$$G_j|w_\pm\rangle = e^{\pm\boldsymbol{i}2\theta_j}|w_\pm\rangle_j. \tag{2.12}$$

Recall from Eq. 2.8 the value of $|\langle p_j|t\rangle|^2$ is encoded in the phase of the eigenvalue of G_j. This will be used in the phase encoding of QNN cost function which is a crucial component of the quantum training.

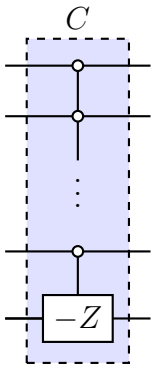

Figure 2.9: Quantum circuit to implement unitary $C = I^{\otimes(2n+1)} - 2|0\rangle^{\otimes(2n+1)}\langle 0|^{\otimes(2n+1)}$.

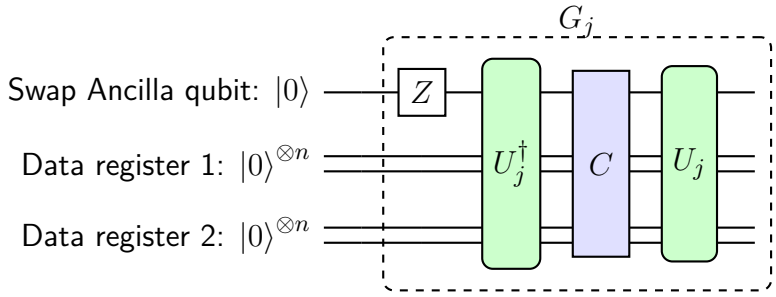

Figure 2.10: *Quantum circuit to implement G_j.* G_j is defined as $G_j := U_j C U_j^\dagger (Z \otimes I^{\otimes 2n})$.

Hadamard Test and its Grover operator

Similar to the swap test, the Hadamard test is a technique that is used to estimate $\langle 0| P_j^\dagger T P_j |0\rangle$, for two unitary operators P_j and T (assuming T is Hermitian). The circuit of Hadamard test is shown in Fig. 2.11.

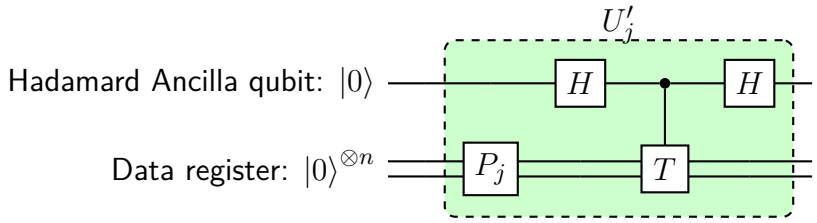

Figure 2.11: *Circuit diagram of Hadamard Test.* The circuit is used to estimate $\langle 0| P_j^\dagger T P_j |0\rangle$, for two unitary P_j and T. The Hadamard test will be used the phase encoding of QNN cost function which is a crucial component of the quantum training.

We denote the unitary of the Hadamard test circuit (the dotted green box in Fig. 2.11) as U_j' and the output state from U_j' as

$$|\phi_j'\rangle = \frac{1}{\sqrt{2}}(|+\rangle P_j |0\rangle + |-\rangle T P_j |0\rangle). \tag{2.13}$$

Rearranging the terms we have

$$|\phi_j'\rangle = \frac{1}{2}\Big(|0\rangle \otimes (P_j |0\rangle + T P_j |0\rangle) + |1\rangle \otimes (P_j |0\rangle - T P_j |0\rangle\Big). \tag{2.14}$$

Denote $|u_j'\rangle$ and $|v_j'\rangle$ as the normalized states of $P_j|0\rangle + TP_j|0\rangle$ and $P_j|0\rangle - TP_j|0\rangle$ respectively. Then there is a real number $\theta_j' \in [0, \pi/2]$ such that

$$|\phi_j'\rangle = \sin\theta_j'|0\rangle|u_j'\rangle + \cos\theta_j'|1\rangle|v_j'\rangle, \tag{2.15}$$

where θ_j' satisfies $\cos\theta_j' = \sqrt{1 - \langle 0| P_j^\dagger T P_j |0\rangle}\,/\sqrt{2}$, $\sin\theta_j' = \sqrt{1 + \langle 0| P_j^\dagger T P_j |0\rangle}\,/\sqrt{2}$. Therefore we have

$$\langle 0| P_j^\dagger T P_j |0\rangle = -\cos 2\theta_j'. \tag{2.16}$$

We can define the Grover operator G_j' from U_j' in the same way as in last subsection for the swap test and obtain similar eigen-relation. The value of $\langle 0| P_j^\dagger T P_j |0\rangle$ is encoded in the phase of the eigenvalue of G_j'. This will be used in the phase encoding of QNN cost function which is a crucial component of the quantum training.

2.4 Block-encoding

Block encoding is a powerful modern quantum algorithmic technique that is employed in a variety of quantum algorithms for solving linear algebra problems on a quantum computer[65]. A unitary U is a block encoding of a not-necessarily-unitary square matrix A (A is scaled to satisfy $\|A\|_2 \leq 1$[65]) if A is encoded in the top-left block of the unitary U_A as:

$$U_A = \begin{bmatrix} A & \cdot \\ \cdot & \cdot \end{bmatrix},$$

where the $\cdot$ symbol stands for a matrix block. Equivalently, we write

$$A = \left(\langle 0|^{\otimes a} \otimes I\right) U_A \left(|0\rangle^{\otimes a} \otimes I\right), \tag{2.17}$$

where a is the number of ancilla qubits used for the block encoding of A. The unitary U_A is considered as a probabilistic implementation of A, i.e., by applying the unitary U to an input state $|0\rangle^{\otimes a}|b\rangle$, measuring the first a-qubit register and post-selecting on the outcome $|0\rangle^{\otimes a}$, we obtain a state that is proportional to $A|b\rangle$ in the second register. This is illustrated in Fig.2.12.

The circuit implementation of Block-encoding in general is constructed using Linear Combination of Unitaries(LCU)[8] technique[66].

2.5 Quantum Singular Value Transformation (QSVT)

Quantum Singular Value Transformation (QSVT)[7] is a recently developed quantum algorithmic framework that can apply polynomial transformations to the singular values of a matrix encoded

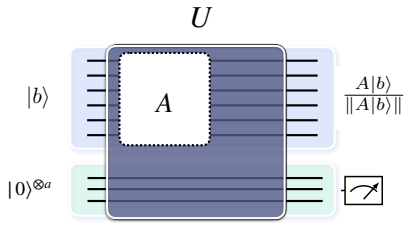

Figure 2.12: *Block-encoding* U, the Block-encoding of a matrix A, can be considered as a probabilistic implementation of A: applying the unitary U to a given input state $|0\rangle^{\otimes a}|b\rangle$, measuring the first a-qubit register and post-selecting on the outcome $|0\rangle^{\otimes a}$, we get state proportional to $A|b\rangle$ in the second register.

as a block of a unitary. The quantum circuits of QSVT have a simple structure, which often results in optimal algorithmic performance, and only require a constant number of ancilla qubits[7].

In this paper, we utilize a special instance of QSVT – Quantum Eigen Value Transformation of a Hermitian matrix which can be specified as follows:

For a Hermitian matrix A, let U be a unitary diagonalizing A and $\lambda_1, \lambda_2, ...\lambda_n$ be the set of eigenvalues of A, i.e.

$$A = U \begin{bmatrix} \lambda_1 & & \\ & \ddots & \\ & & \lambda_n \end{bmatrix} U^\dagger$$

Applying QSVT to A aims to achieve applying a certain polynomial function to the eigenvalues of A such that:

$$P(A) = U \begin{bmatrix} P(\lambda_1) & & \\ & \ddots & \\ & & P(\lambda_n) \end{bmatrix} U^\dagger$$

where $P(x)$ is a polynomial function satisfying certain conditions. This can carried out if we are given access to a unitary U_A as a block-encoding of A:

$$U_A := \begin{bmatrix} A & . \\ . & . \end{bmatrix}$$

Using unitary U_A and its inverse, plus some phase gates extended onto an additional ancillary qubit, QSVT realizes a new unitary that is the block-encoding of $P(A)$:

$$\begin{bmatrix} P(A) & . \\ . & . \end{bmatrix}$$

The quantum circuit for QSVT can be depicted as in Fig.2.13:

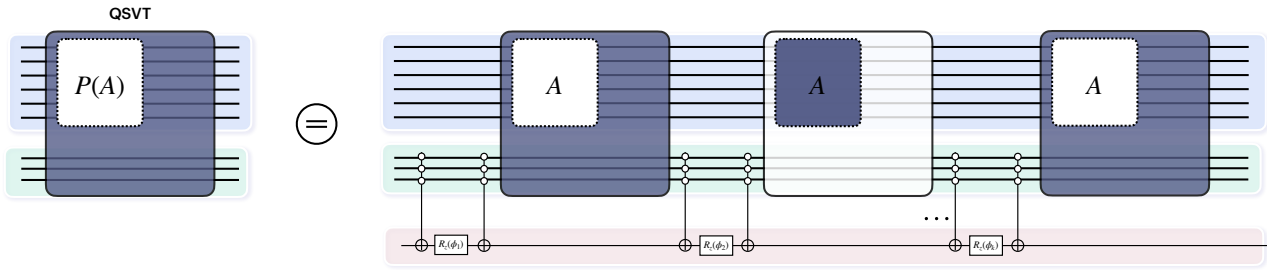

Figure 2.13: *Quantum circuit for Quantum Singular Value Transformation (QSVT)* Given access to a multi-qubit unitary $U(A)$ as a block-encoding of A, using unitary $U(A)$ and it's inverse, plus some phase gates extended onto an additional ancillary qubit, QSVT realizes a new unitary that is the block-encoding of $P(A)$.

The phases $\Phi = \{\phi_0, \phi_1, \ldots, \phi_k\}$ in the QSVT circuit determine what polynomial function is applied to the singular values of the matrix and vice-versa. For certain polynomials, the phases have simple form: e.g., for d-th Chebyshev polynomial of the first kind, the phases take the following values: $\Phi = \left\{(1-d)\frac{\pi}{2}, \frac{\pi}{2}, \ldots, \frac{\pi}{2}\right\}$.

Part I

Quantum Training of QNNs

Chapter 3

Quantum Training of QNNs: Overview

A QNN consists of a set of parameterized quantum gates within a predefined circuit ansatz. The design of the ansatz together with the value of the gate parameters determine the outcome of the QNN. In order to successfully perform certain tasks, QNNs must be trained to find optimal parameters for generating desired outcomes. In the majority of QNN research, the training is carried out by employing variational hybrid quantum-classical algorithms [32], in which the parameters are optimized by a classical optimizer using gradient-based or gradient-free approaches. There are two main avenues for the application of QNNs. The first uses QNNs to generate quantum states that minimize the expectation value of a given Hamiltonian, such as the case in Variational Quantum Eigensolvers (VQE) [67] for chemistry problems or Quantum Approximate Optimization Algorithms (QAOA) [52] for combinatorial optimization problems. The second path uses QNNs as data-driven machine learning models to perform discriminative [68, 69, 70] and generative [71, 72, 73, 74, 75] tasks for which QNNs could have more expressive power than their classical counterparts [76]. Though an ever-increasing amount of effort is being put into QNN research, there is evidence that they will be difficult to train due to flat optimisation landscapes called barren plateaus [29].

The barren plateau issue has spawned several studies on the strategies to avoid them, including layerwise training [77], using local cost functions [78], correlating parameters [79], and pre-training [80], among others [81, 82, 83]. Such strategies give hope that the variational quantum-classical algorithms may avoid the exponential scaling due to the barren plateau issue. However, it has been shown that these strategies do not avoid another type of Barren Plateaus induced by hardware noise [84], and some strategies may lack theoretical grounding [85]. In addition to noise, there is also other sources of barren plateaus due to entanglement growth [86]. Moreover, it has been shown that gradient-free approaches are also adversely affected by barren plateaus [87].

The above-noted results indicate that training QNNs using classical optimisation methods

has unprecedented challenges as the system scales up. Therefore, one seeks to leverage alternative optimisation methods for training QNNs. In the first part of the thesis including Chapter3 4 5, we design a scalable, maximally quantum pipeline of the applications of QNNs by replacing the classical optimizer by quantum optimizers. In short, we employ quantum optimisation methods for training QNNs. Our quantum training methods of QNNs exploit hidden structures in the QNN optimisation problem and are expected to mitigate the barren plateau issue.

3.1 Previous work on Quantum Training of QNNs

Preliminary attempts have been made in utilizing quantum methods for training QNNs. Verdon et al. proposed a QAOA-like training protocol for QNNs [3] and Gilyén et al. developed a quantum algorithm for calculating gradients faster than classical methods [4]. In these two works, to cast the optimisation problem of training QNNs into the context of quantum optimisation, the network parameters in the QNN are quantized — moved from being classical to being stored in quantum registers, which are in addition to those upon which the QNN is performing its computation. The quantized parameters are used as control registers of the parameterized gates on the QNN registers. The parameters can now be in superposition, which one hopes would allow for a *quantum parallelism*-type computation of the QNN with multiple parameter configurations.

In Ref. [3], the quantum training process can be described as the state evolution in the joint Hilbert space of the parameter register and the QNN register. Their quantum training protocol consists of two alternating operations in a QAOA fashion — the first operation acts on both the parameter register and QNN register to encode the cost function of QNN onto a relative phase of the parameter state. The second operation acts only on the parameter register and it is a variant of the original QAOA Mixers, tailored for the case that the parameters in the QNN are continuous variables. These two operation can be mathematically expressed as $e^{-i\gamma_i C(\boldsymbol{\theta})}$ and $e^{-i\beta_i H_M}$, where $\boldsymbol{\theta}$ are the parameters of QNN, $C(\boldsymbol{\theta})$ is the cost function of the QNN, and γ_i and β_i are tunable hyperparameters, H_M is the Mixer Hamiltonian. By heuristically tuning the hyperparameters, the quantum training is expected to home in on the optimal parameters of the QNN after several iterations of the QAOA alternating operations. We illustrate the alternating operations of their quantum training in Fig. 3.1.

Despite being the pioneering application of the QAOA method for training QNNs, the protocol in Ref. [3] has some limitations. In the phase encoding operation, the parameter register and the QNN register are generally always entangled. This will have the effect of causing phase decoherence in the parameter eigenbasis. To minimize the effect of this decoherence, the tuneable hyper-parameter γ_i must be sufficiently small — in other words, the phase encoding is

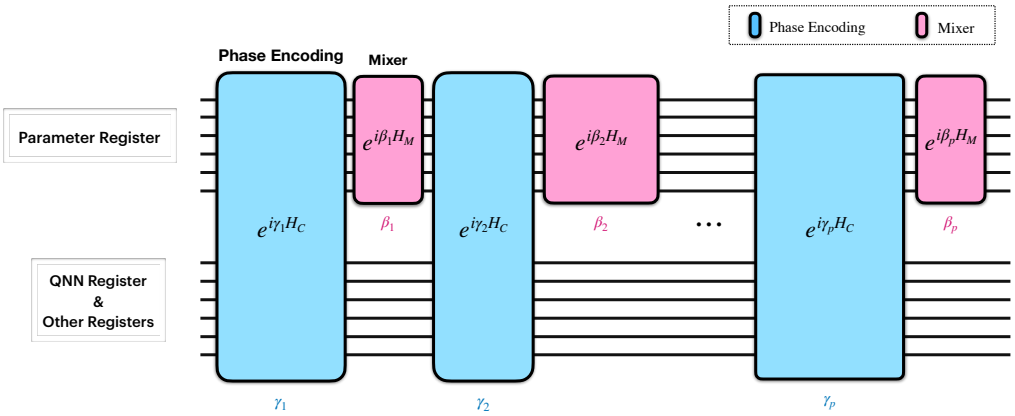

Figure 3.1: *QAOA-like training protocol for QNN, proposed in Ref. [3].* The quantum training protocol consists of two alternating operations in a QAOA fashion — the first operation acts on both the parameter register and QNN register to encode the cost function of QNN onto a relative phase of the parameter state. This operation is represented by the blue blocks in the figure. The second operation acts only on the parameter register and it is a variant of the original QAOA Mixers, tailored for the case that the parameters in the QNN are continuous variables. This operation is represented by the pink blocks in the figure. These two operation can be mathematically expressed as $e^{-i\gamma_i C(\boldsymbol{\theta})}$ and $e^{-i\beta_i H_M}$, where $\boldsymbol{\theta}$ are the parameters of QNN, $C(\boldsymbol{\theta})$ is the cost function of the QNN, and γ_i and β_i are tunable hyperparameters,H_M is the Mixer Hamiltonian. The width of each block represents the hyperparameters γ_i and β_i — the wider the block, the larger the value of the hyperparameters. The phase encoding operation $e^{-i\gamma_i H_C}$ act as $e^{-i\gamma_i C(\boldsymbol{\theta})}$.

coherent only in the first order of γ_i. To overcome this limitation — to enact phase encoding operation with arbitrary hyperparameters — the phase encoding operation with a small hyperparameter $\Delta\gamma$ should be repeated an excessive amount of times. This simulates the phase encoding operation with a large hyperparameter γ via $e^{-i\gamma C(\boldsymbol{\theta})} = e^{-i\Delta\gamma C(\boldsymbol{\theta})} e^{-i\Delta\gamma C(\boldsymbol{\theta})} e^{-i\Delta\gamma C(\boldsymbol{\theta})} \ldots$ These repetitions will yield large overhead in the complexity of the algorithm. In Ref. [4], a *phase oracle* is designed for the phase encoding and can achieve it coherently and efficiently. (Note that throughout this paper, the term *phase oracle* has a different meaning than the one in Ref. [4], our *phase oracle* stands for the term *fractional phase oracle* in Ref. [4].) Nevertheless, they did not utilise the phase encoding as a component of QAOA routine to accomplish a fully quantum training algorithm for QNNs. Instead, they use the phase oracle as a component of the quantum evaluation of the gradient of a QNN, which serves for gradient-based classical training of QNNs. However, this improvement will not be practically useful due to the barren plateau issue of QNNs.

3.2 Our Framework

In this part of the thesis, we devise an improved framework for training QNNs, taking advantage of the well-established parts in Refs. [3] and [4], while eliminating the shortcomings. A schematic of our quantum training framework for QNNs is depicted in Fig. 3.2. More specifically, we

achieve the following:

1. We replace the phase encoding operations in QAOA-like protocol of Ref. [3] by the *phase oracle* in Ref. [4]. This achieves coherent encoding of the cost function onto a relative phase of parameter state, while avoiding the limitations of the hyper-parameters in the phase encoding.

2. For the mixers in the QAOA-like routine we adopt a similar approach to Ref. [5] by making the mixers adaptive — that is, we allow different mixers for each layer (particularly, to allow entangling mixers that act across different parameters). This potentially leads to a dramatic shortening of the depth of QAOA layers while significantly improving the quality of the solution (the optimal QNN parameters found by the QAOA routine).

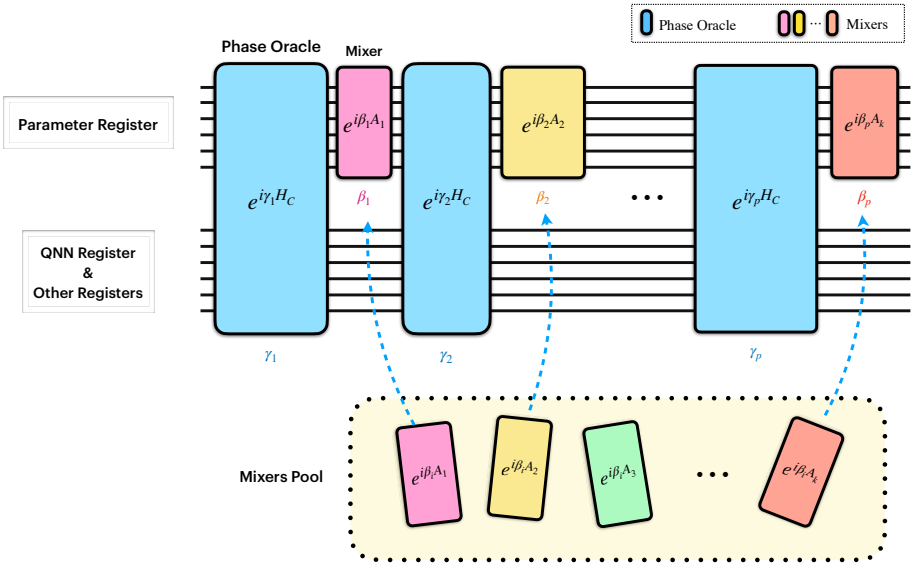

Figure 3.2: *Schematic of our framework for quantum training of QNN.* Our quantum training for QNN takes advantage of the well-established parts in Refs. [3] and [4], while eliminating their shortcomings. We replace the phase encoding operations in QAOA-like protocol of Ref. [3](as depicted in Fig 3.1) by the *phase oracle* in Ref. [4]. For the mixers in the QAOA-like routine, we allow different mixers for each layer, similar to Ref. [5]. In this figure, the colour of each block represents the nature of the corresponding Hamiltonian: different colour corresponds to different Hamiltonian (One can see that the Cost Hamiltonian is the same throughout the training whereas the mixer varies from layer to layer). The mixers pool contains the proper mixers tailored to our QNN training problem. These rules also apply to the other circuit schematic in this paper.

By making the mixers flexible and adaptive to specific optimisation problems, it is demanding to find an efficient way of determining the best sequence of mixers and the optimized hyperparameters. To address these we adopt machine learning approaches (in particular, Recurrent Neural Networks and Reinforcement Learning) as proposed in Refs. [88, 20, 89, 80]. The quantum mechanism of this framework is the best candidate to exploit hidden structures in the

QNN optimisation problem, which would provide beyond-Grover speed-up and mitigate the barren plateau issues for training QNNs.

3.3 Creating Superpositions of QNNs

As an essential building block for our quantum training protocol, we present a way to create superpositions of QNNs entangled with corresponding parameters. That is, we construct a controlled unitary P such that

$$P|\boldsymbol{\theta}\rangle|0\rangle \to |\boldsymbol{\theta}\rangle \otimes U(\boldsymbol{\theta})|0\rangle \quad \text{for every } \boldsymbol{\theta} \tag{3.1}$$

in which $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)$ is the set of trainable parameters in the QNN and $U(\boldsymbol{\theta})$ is the unitary of the QNN with corresponding parameters. When P acts on a superposition state of parameters $\sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle$, we have

$$P \sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle |0\rangle \to \sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle \otimes U(\boldsymbol{\theta})|0\rangle . \tag{3.2}$$

The action of the controlled unitary P is depicted in Fig. 3.3.

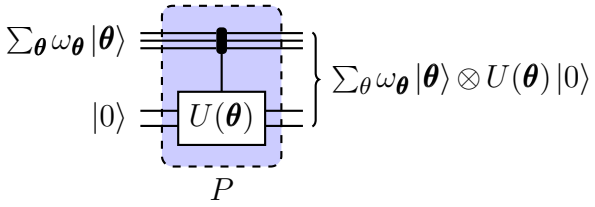

Figure 3.3: *Action of the controlled unitary P.* In this figure, the upper register is the parameter register and the lower register is the QNN register. $\boldsymbol{\theta} = (\theta_1, \ldots, \theta_M)$ is the set of trainable parameters in the QNN and $U(\boldsymbol{\theta})$ is the unitary of the QNN with corresponding parameters. The qubits in the parameter register act as control qubits on the rotation gates in the QNN. The controlled operations (in the dotted blue box) is denoted as P. When P acts on a superposition state of parameters $\sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle$, the output state is $\sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle \otimes U(\boldsymbol{\theta})|0\rangle$. in which the parameter register and QNN register are entangled.

This controlled unitary can be realised by dividing each rotation gate in QNN into a sequence of binary segments, followed by applying controlled operations on them. A simple example of one rotation gate, for example $U(\boldsymbol{\theta}) = R_z(\theta)$, is illustrated in Fig. 3.4.

Each bit string of the parameter register is a binary representation of the rotation angle. Furthermore the associated basis state of the register is entangled with the rotation gate of the corresponding angle. For instance, the bit string 111 corresponds to the angle $7\bar{\theta}/8$ and $|111\rangle$ is associated with $R_z(7\bar{\theta}/8)$, where $\bar{\theta}$ is the maximum value that angle θ can take. This relation can be fully illustrated in Fig. 3.5, in which we take $\bar{\theta} = \pi$.

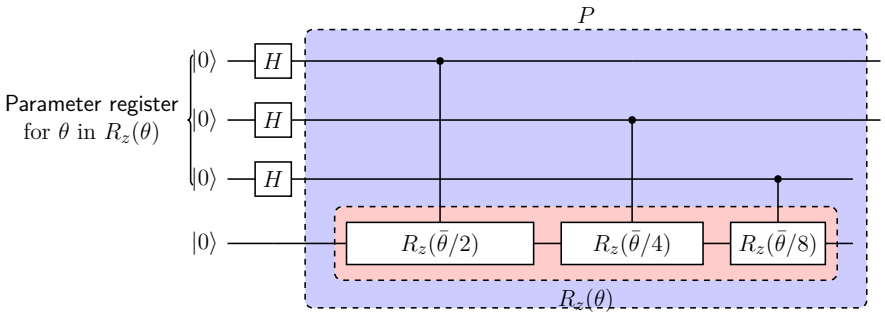

Figure 3.4: *An example of the construction of P for one rotation gate $R_z(\theta)$.* In this example, the parameter register consists of three qubits, each qubit controls a "partial" rotation on the fourth qubit. The "partial" rotation are the binary segments $R_z(\bar{\theta}/2)$,$R_z(\bar{\theta}/4)$,$R_z(\bar{\theta}/8)$ in which $\bar{\theta}$ is the maximum value that angle θ can take.

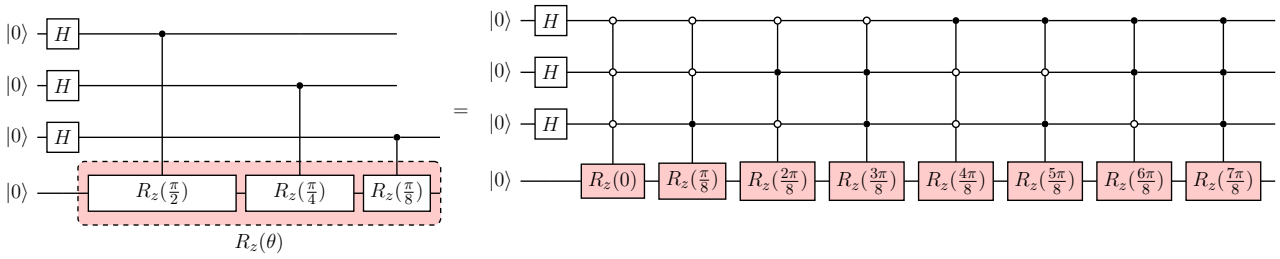

Figure 3.5: *An example of the effect of P defined in Fig. 3.4.* Each bit string of the parameter register can be seen as a binary representation of the rotation angle and the associated basis state of the register is entangled with the rotation gate of the corresponding angle. For instance, in the example above, the bit string 111 corresponds to the angle $7\bar{\theta}/8$ and $|111\rangle$ is associated with $R_z(7\bar{\theta}/8)$.

The unitary operator of P can be written as:

$$P = \sum_j |j\rangle\langle j| \otimes P_j, \tag{3.3}$$

in which P_j is a specific configuration of the QNN defined by its control bit string j. This representation does not only apply to a single rotation gate, but also to the case where there are multiple parameterised rotation gates in the QNN. An example of two rotation gates is depicted in Fig. 3.6.

In order to achieve precision ϵ_0 for each rotation angle, the number of control qubits needed is $d = \lceil \log_2(1/\epsilon_0) \rceil$. Let r be the number of rotation gates in a QNN, then the total number of control qubits needed is dr.

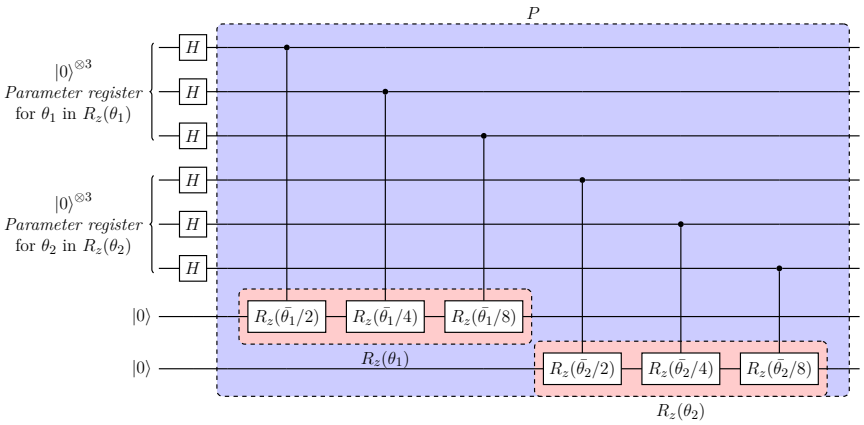

Figure 3.6: *Example of the construction of P for QNN consisting of two rotation gates.* In this example, the QNN consist of two rotation gates $R_z(\theta_1)$, $R_z(\theta_2)$ on the lower two qubits. The upper 6 qubits are divided into two parameter registers for the two rotation angles θ_1, θ_2 respectively. Each qubit controls a "partial" rotation. For instance, the "partial" rotations of $R_z(\theta_1)$ are the binary segments $R_z(\bar{\theta_1}/2)$,$R_z(\bar{\theta_1}/4)$,$R_z(\bar{\theta_1}/8)$ in which $\bar{\theta_1}$ is the maximum value that angle θ_1 can take.

Chapter 4

QNN training by Grover Adaptive Search

In this chapter, we discuss using Grover adaptive search to perform global optimisation of QNNs. As presented in Section 2.2, the core of the Grover adaptive search is the adaptive oracle defined in Eq. 2.3. In what follows, we detail how to construct such oracle for QNN training.

4.1 Construction of the Grover Oracle

The adaptive Grover Oracle $\mathcal{O}_{Grover}$ in the context of QNN training acts as

$$\mathcal{O}_{Grover} |\boldsymbol{\theta}\rangle \otimes |\mathbf{0}\rangle_{QNN+ancillas} = (-1)^{g(C(\boldsymbol{\theta})-C^*)} |\boldsymbol{\theta}\rangle \otimes |\mathbf{0}\rangle_{QNN+ancillas}, \quad \text{for every } \boldsymbol{\theta}, \tag{4.1}$$

in which C^* is the adaptive threshold for the cost function and the function g is defined as

$$g(x) = \begin{cases} 1 & x < 0 \\ 0 & \text{otherwise} \end{cases}. \tag{4.2}$$

When $\mathcal{O}_{Grover}$ acts on a superposition state of parameters $\sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle$, we found

$$\mathcal{O}_{Grover} \sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle \otimes |\mathbf{0}\rangle_{QNN+ancillas} = \sum_{\boldsymbol{\theta}} (-1)^{g(C(\boldsymbol{\theta})-C^*)} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle \otimes |\mathbf{0}\rangle_{QNN+ancillas}. \tag{4.3}$$

The QNN Grover oracle $\mathcal{O}_{Grover}$ can be constructed by the following steps.

4.1.1 Amplitude Encoding

The first step is to encode the cost function of QNN into amplitude. Depending on the form of the cost function of the QNN, the amplitude encoding is achieved by the swap test or Hadamard test. The correspondences are summarized in Table 4.1.

Task	**Cost function**		**Amplitude encoding Method**
Generating Ground state of T	Expectation value	$C(\boldsymbol{\theta}) = \langle 0\vert U^\dagger(\boldsymbol{\theta})TU(\boldsymbol{\theta})\vert 0\rangle$	***Hadamard test***
Generating a pure state $\vert\psi\rangle = T\vert 0\rangle$ (T is a given unitary)	Fidelity	$C(\boldsymbol{\theta}) = \vert\langle 0\vert U^\dagger(\boldsymbol{\theta})T\vert 0\rangle\vert^2$	***Swap test***

Table 4.1: *QNN Cost functions for two types of tasks.* Here we present the Cost functions for two tasks respectively: For the task of Generating Ground state of some given Hamiltonian T(we use T instead of H here, and assume T is Hermitian), the cost function is chosen to be the Expectation value of T. For the task of Generating a pure state $\vert\psi\rangle = T\vert 0\rangle$ (T is a given unitary), the cost function is chosen to be the Fidelity between the generated state from QNN and the state $\vert\psi\rangle = T\vert 0\rangle$.

Amplitude Encoding by Swap test. For the task of learning a pure state $\vert\psi\rangle = T\vert 0\rangle$ (T is a given unitary), the cost function is the fidelity between the generated state from the QNN and the state $\vert\psi\rangle = T\vert 0\rangle$. In this case, the amplitude encoding can be achieved by swap test, as shown in the circuit in Fig. 4.1.

We denote the unitary for the swap test circuit (in the dotted green box) as U, and the input and output state of U as $\vert\Psi_0\rangle$ and $\vert\Psi_1\rangle$, respectively. The input to U, $\vert\Psi_0\rangle$, can be written as (note here and throughout the paper, we omit the normalization factor):

$$\vert\Psi_0\rangle = \vert 0\rangle \otimes (\sum_j \vert j\rangle) \otimes \vert 0\rangle^n_{QNN1} \vert 0\rangle^n_{QNN2} \tag{4.4}$$

Then U can be written explicitly as

$$U \coloneqq [H\otimes I\otimes I\otimes I]\cdot[\vert 0\rangle\langle 0\vert \otimes (\sum_j \vert j\rangle\langle j\vert \otimes P_j \otimes T) + \vert 1\rangle\langle 1\vert \otimes (\sum_j \vert j\rangle\langle j\vert \otimes T \otimes P_j)]\cdot[H\otimes I\otimes I\otimes I], \tag{4.5}$$

Here, P_j represents QNN with specific parameter configuration defined by its control bit string j, as defined in Eq. 3.3. It can be proven (see Appendix A.2) that U can be rewritten as

$$U = \sum_j \vert j\rangle\langle j\vert \otimes U_j, \tag{4.6}$$

where U_j is the *individual swap test unitary* on unitary P_j and target unitary T, defined as in Eq. 2.4:

$$U_j \coloneqq [H\otimes I\otimes I]\cdot[\vert 0\rangle\langle 0\vert \otimes P_j \otimes T + \vert 1\rangle\langle 1\vert \otimes T \otimes P_j]\cdot[H\otimes I\otimes I]. \tag{4.7}$$

As in Eq. 2.7, the resulting state of U_j acting on $\vert\Psi_0\rangle$ is $\vert\phi_j\rangle \coloneqq U_j \vert 0\rangle \vert 0\rangle^n_{QNN1} \vert 0\rangle^n_{QNN2}$ and has the following form:

$$\vert\phi_j\rangle = \sin\theta_j \vert u_j\rangle \vert 0\rangle + \cos\theta_j \vert v_j\rangle \vert 1\rangle . \tag{4.8}$$

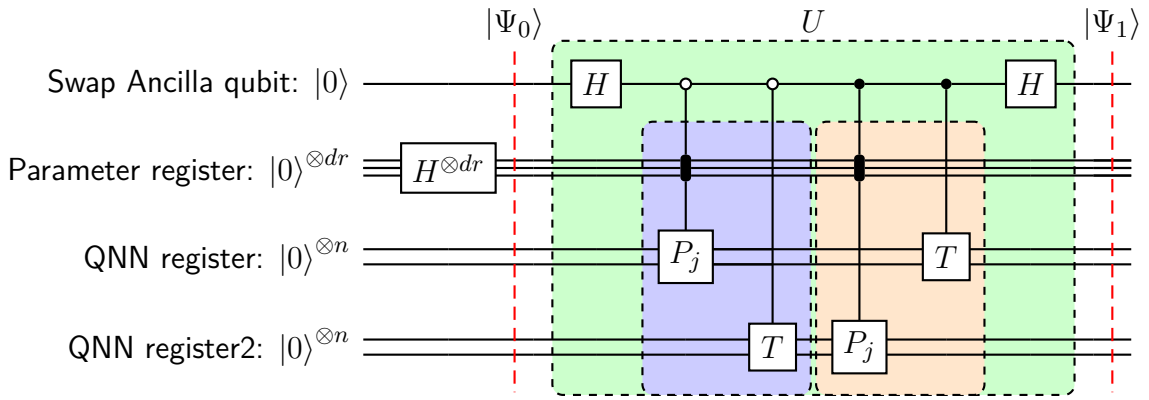

Figure 4.1: *Amplitude Encoding by Swap test.* This circuit can perform the swap test depicted in Fig. 2.8 in parallel for multiple P_j. Here, P_j represents QNN with specific (the "jth") parameter configuration. To achieve swap test in parallel, we add an extra register– the parameter register— as the control of P_j: each computational basis j of the parameter register corresponds to a specific parameter configuration in P_j. As illustrated in Fig. 3.3, once the parameter register is in superposition state(by the Hadamard gates $H^{\otimes dr}$), the corresponding P_j are in superposition. We refer to the control operation on QNN as "controlled-QNN". Compared with the normal swap test depicted in Fig. 2.8, the difference here is that the Swap ancilla qubit is anti-controlling /controlling the "controlled-QNN" together with the Unitary T(as gathered together in the dotted blue/orange box). It can be proven that the entire circuit in dotted the green box (denoted as U) can be expressed as $U = \sum_j |j\rangle\langle j| \otimes U_j$ where U_j is the swap test unitary for P_j defined in Fig. 2.8. This indicates that U effectively perform the swap test in parallel for multiple P_j. Recall the fact that the normal swap test U_j encode $|\langle p_j|t\rangle|^2$ in the amplitude of the output state (Eq. 2.8 and Eq. 2.7), here the "parallel swap test" U encodes the QNN cost function $|\langle p_j|t\rangle|^2$ in the amplitude of a superposition of P_j(QNN) with different parameters.

The final output state from U, $|\Psi_1\rangle = U|\Psi_0\rangle$, is therefore

$$|\Psi_1\rangle = \sum_j |j\rangle \underbrace{(\sin\theta_j |u_j\rangle|0\rangle + \cos\theta_j |v_j\rangle|1\rangle)}_{|\phi_j\rangle} = \sum_j |j\rangle|\phi_j\rangle \tag{4.9}$$

From Eqs. 4.9 and 2.8 we see that the cost function (fidelity $|\langle p_j|t\rangle|^2$) for different parameters has been encoded into the amplitudes of the state $|\Psi_1\rangle$.

Amplitude encoding by Hadamard Test. For the task of generating ground states of given Hamiltonian T, the cost function is the expectation value of T with respect to the generated state from the QNN. In this case, the amplitude encoding can be achieved by the Hadamard test, as shown in the circuit in Fig. 4.2.

Since the analysis for the case of the Hadamard test is very similar to that of the swap test, we omit the details here. For the same reason, we only present the case using the swap test also in the next section when discussing phase encoding.

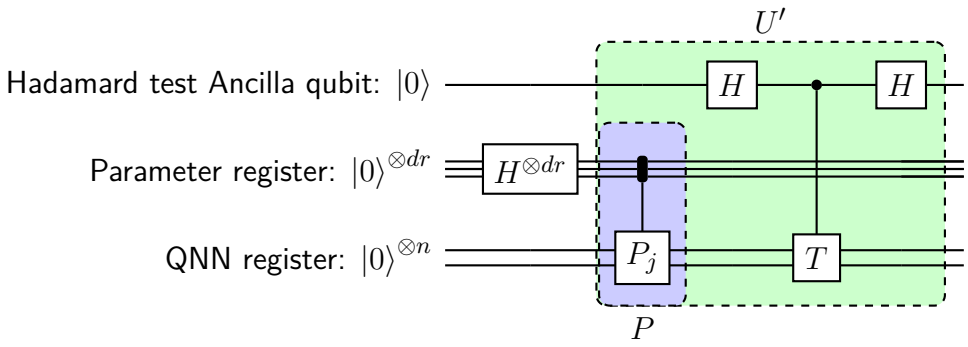

Figure 4.2: *Amplitude encoding by Hadamard Test* This circuit can perform the Hadamard test depicted in Fig. 2.11 in parallel for multiple P_j. Here, P_j represents QNN with specific (the "jth") parameter configuration. To achieve Hadamard test in parallel, we add an extra register– the parameter register— as the control of P_j: each computational basis j of the parameter register corresponds to a specific parameter configuration in P_j. As illustrated in Fig. 3.3, once the parameter register is in superposition state(by the Hadamard gates $H^{\otimes dr}$), the corresponding P_j are in superposition. It can be proven that the entire circuit in dotted the green box (denoted as U) can be expressed as $U' = \sum_j |j\rangle\langle j| \otimes U'_j$ where U'_j is the Hadamard test unitary for P_j defined in Fig. 2.11. This indicates that U' effectively perform the swap test in parallel for multiple P_j. Recall the fact that the normal Hadamard test U'_j encode $\langle 0| P_j^\dagger T P_j |0\rangle$ in the amplitude of the output state (Eq. 2.15 and Eq. 2.16), here the "parallel Hadamard test" U' encodes the QNN cost function $\langle 0| P_j^\dagger T P_j |0\rangle$ in the amplitude of a superposition of P_j(QNN) with different parameters.

4.1.2 Amplitude estimation

The second step following the amplitude encoding is to use amplitude estimation [90] to extract and store the cost function into an additional register which we call the "amplitude register". In the following, we present the details of amplitude estimation.

After the amplitude encoding by the swap test, we introduce an extra register $|0\rangle^t_{\textit{amplitude}}$ and the output state $\left|\overline{\Psi_1}\right\rangle = |\Psi_1\rangle \otimes |0\rangle^t_{\textit{amplitude}}$ becomes

$$\left|\overline{\Psi_1}\right\rangle = \sum_j |j\rangle |\phi_j\rangle |0\rangle^t_{\textit{amplitude}}, \tag{4.10}$$

where $|\phi_j\rangle$ can be decomposed as

$$|\phi_j\rangle = \frac{-i}{\sqrt{2}} \left(e^{i\theta_j} |\omega_+\rangle_j - e^{i(-\theta_j)} |\omega_-\rangle_j \right). \tag{4.11}$$

Hence, we have

$$|\Psi_1\rangle = \sum_j \frac{-i}{\sqrt{2}} \left(e^{i\theta_j} |j\rangle |\omega_+\rangle_j - e^{i(-\theta_j)} |j\rangle |\omega_-\rangle_j \right) |0\rangle^t_{\textit{amplitude}}. \tag{4.12}$$

The overall Grover operator G is defined as

$$G \coloneqq U C_2 U^{-1} C_1, \tag{4.13}$$

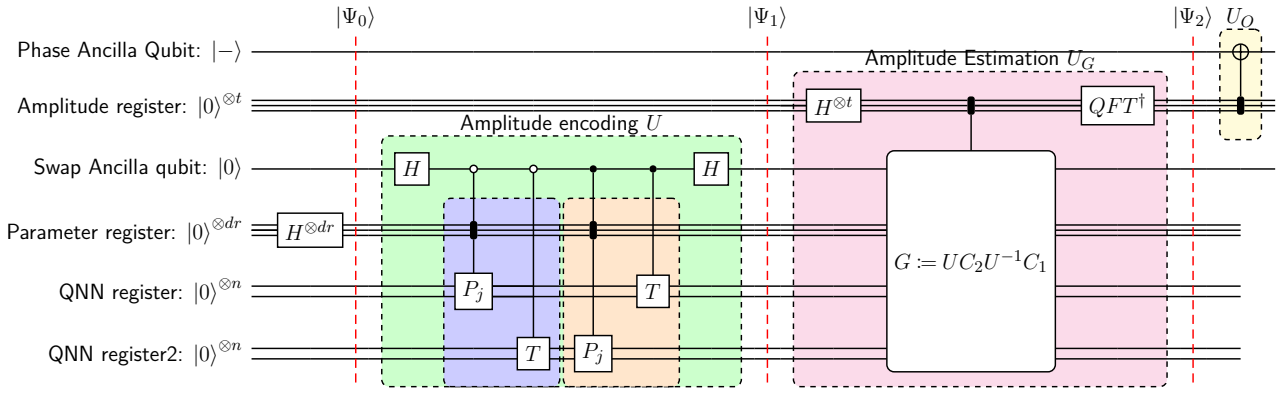

Figure 4.3: *Major steps in the Construction of the Grover Oracle.* Step 0: We initialize the system by applying Hadamard gates on the parameter register, leading to the state $|\Psi_0\rangle = |0\rangle_s \otimes (\sum_j |j\rangle) \otimes |0\rangle^n_{QNN1} |0\rangle^n_{QNN2}$. Step 1(dotted green box): Amplitude encoding of the cost function, as illustrated in Fig. 4.1 (refer the caption of Fig. 4.1 for the meaning of each symbol), resulting in the state $|\Psi_1\rangle = \sum_j |j\rangle (\sin\theta_j |u_j\rangle |0\rangle + \cos\theta_j |v_j\rangle |1\rangle)$, in which θ_i contains the cost function. Step 2(dotted pink box): Amplitude estimation to extract and store the cost function into an additional register which we call the "amplitude register", resulting in the state $|\Psi_2\rangle = \sum_j \frac{-i}{2} \left(e^{i\theta_j} |j\rangle |\omega_+\rangle_j |2\theta_j\rangle - e^{i(-\theta_j)} |j\rangle |\omega_-\rangle_j |-2\theta_j\rangle\right)$. Step 3(dotted yellow box): Threshold Oracle to encode the cost function into relative phase by using a Phase ancilla qubit, resulting in the state $|\Psi_3\rangle = \sum_j \frac{-i}{2} (-1)^{g(\theta_j - \theta^*)} \left(e^{i\theta_j} |j\rangle |\omega_+\rangle_j |2\theta_j\rangle - e^{i(-\theta_j)} |j\rangle |\omega_-\rangle_j |-2\theta_j\rangle\right)$.

where C_1 is the Z gate on the swap ancilla qubit, and C_2 is "flip zero state" unitary which is similar to that defined in Fig. 2.9. It can be shown (see Appendix A.2) that G can be expressed as

$$G = \sum_j |j\rangle\langle j| \otimes G_j, \tag{4.14}$$

where G_j is the individual Grover operator as defined in Eq. 2.11. The overall Grover operator G possesses the following eigen-relation:

$$G |j\rangle |\omega_\pm\rangle_j = e^{i(\pm 2\theta_j)} |j\rangle |\omega_\pm\rangle_j . \tag{4.15}$$

Next, we apply phase estimation of the overall Grover operator G on the input state $|\Psi_1\rangle$. The resulting state $|\Psi_2\rangle$ can be written as

$$|\Psi_2\rangle = \sum_j \frac{-i}{\sqrt{2}} \left(e^{i\theta_j} |j\rangle |\omega_+\rangle_j |2\theta_j\rangle - e^{i(-\theta_j)} |j\rangle |\omega_-\rangle_j |-2\theta_j\rangle\right). \tag{4.16}$$

Note here in Eq. 4.16, $|\pm 2\theta_j\rangle$ denotes the eigenvalues $\pm 2\theta_j$ being stored in the amplitude register with some finite precision.

4.1.3 Threshold Oracle and Uncomputations

Next, we apply a threshold oracle U_O on the amplitude register and an extra phase ancilla qubit, which acts as

$$U_O |\pm 2\theta_j\rangle = (-1)^{g(\theta_j - \theta^*)} |\pm 2\theta_j\rangle , \tag{4.17}$$

where θ^* is implicitly defined as

$$C^* = -\cos 2\theta^*, \quad \theta^* \in [\pi/4, \pi/2]. \tag{4.18}$$

Note that in Eq. 4.17 we omit the state of the phase ancilla qubit.

The state after the oracle $|\Psi_3\rangle$ can be written as

$$|\Psi_3\rangle = \sum_j \frac{-i}{\sqrt{2}}(-1)^{g(\theta_j-\theta^*)}\left(e^{i\theta_j}|j\rangle|\omega_+\rangle_j|2\theta_j\rangle - e^{i(-\theta_j)}|j\rangle|\omega_-\rangle_j|-2\theta_j\rangle\right), \tag{4.19}$$

The procedure thus far can be illustrated in a circuit as in Fig. 4.3.

After we perform the uncomputation of Phase estimation, the resulting state is

$$|\Psi_4\rangle = \sum_j \frac{-i}{\sqrt{2}}(-1)^{g(\theta_j-\theta^*)}\left(e^{i\theta_j}|j\rangle|\omega_+\rangle_j|0\rangle^t_{\mathit{amplitude}} - e^{i(-\theta_j)}|j\rangle|\omega_-\rangle_j|0\rangle^t_{\mathit{amplitude}}\right), \tag{4.20}$$

$$= \sum_j (-1)^{g(\theta_j-\theta^*)}|j\rangle|\phi_j\rangle|0\rangle^t_{\mathit{amplitude}}. \tag{4.21}$$

Finally, we perform the uncomputation of the swap test and the resulting state is

$$|\Psi_5\rangle = \sum_j (-1)^{g(\theta_j-\theta^*)}|j\rangle|0\rangle^n_{\mathit{QNN1}}|0\rangle^n_{\mathit{QNN2}}|0\rangle|0\rangle^t_{\mathit{amplitude}}. \tag{4.22}$$

As can be seen from Eqs. 4.22 and 2.8, the above steps implemented the Grover oracle $\mathcal{O}_{\mathit{Grover}}$ (defined in Eq. 4.1) After the above procedure a relative phase, which depends on the cost function of the QNN $|\langle p_j|t\rangle|^2$ and the threshold, have been coherently added to the parameter state. Importantly, uncomputation allows the parameter register to be decoupled from the QNN and other registers.

Performance of the Quantum training by Grover Adaptive Search

Taking training VQE as an example, in Table. 4.2 we present the result for the number of "controlled-QNN" runs, the number of QNN runs and the number of measurements needed in the quantum training by Grover Adaptive Search. The derivation is included in Appendix A.3.

4.2 Advantages and disadvantages of training by Grover Adaptive Search

In the presence of a noise-free barren plateau, the Grover Adaptive Search mechanism can find global optima without an exponential number of measurements. However, it has the following disadvantages:

- It can be seen from Table. 4.2 that in Quantum training by Grover adaptive search, the number of "controlled-QNN" runs is exponential in the number of parameters in QNN.

Number of "controlled-QNN" runs	Number of QNN runs	Number of measurements
$O\left(\frac{1}{\epsilon_2\epsilon_1}\left(\frac{1}{\epsilon_0}\right)^{r/2}s^{-0.5}\right)$	$O\left(\frac{1}{\epsilon_2}\left(r\log\left(\frac{1}{\epsilon_0}\right)\right)^{1.5}\right)$	$O\left(\left(r\log\left(\frac{1}{\epsilon_0}\right)\right)^{1.5}\right)$

Table 4.2: *Performance of the Quantum training by Grover Adaptive Search.* Here we present the result for the number of "controlled-QNN" runs, the number of QNN runs and the number of measurements needed in the quantum training by Grover Adaptive Search. In this table, r is the number of parameters (rotation angles) in QNN, $1-\epsilon_1$ is the probability of success of the phase estimation, ϵ_2 is the precision we set up for the evaluation of the cost function using amplitude estimation, ϵ_0 is the precision of each angle value, s is the number of global optima of the QNN cost function.

Even in the case where the number of parameters scales only linearly with the number of qubits in a QNN, the quantum training by Grover takes excessive runtime. Moreover, it invokes very deep circuit.

- Training by Grover adaptive search does not circumvent the noise-induced barren plateau. When the entire cost landscape is flatten in the case of noise-induced barren plateau [84], it requires exponential precision of the amplitude estimation. That is, ϵ_2 should be exponentially small. According to Table. 4.2, this adds another exponentially large factor to the number of "controlled-QNN" runs and QNN runs.

While these disadvantages most probably rule out Grover adaptive search for NISQ-era devices, it still represents a maximally quantum solution. For fault-tolerant devices, this method is the provably optimal approach for QNN cost function with no structure, it enjoys a quadratic speed-up which is a significant improvement compared to the exponential "slow-down" of the classical training methods due to the barren plateau issue.

Chapter 5

QNN training by Adaptive QAOA

In this chapter we discuss using Adaptive QAOA to perform the training of QNNs. As depicted in Fig. 3.2, our framework for quantum training using Adaptive QAOA of QNNs consists of two major components.

- **Phase oracle.** This coherently encodes the cost function of QNNs onto a relative phase of a superposition state in the Hilbert space of the parameters [4].
- **Adaptive Mixers.** These exploit hidden structures in QNN optimisation problems, hence can achieve short-depth circuit [91].

Iterations of the phase oracle and the adaptive mixers constitute a QAOA routine which quantumly homing in on optimal network parameters of QNNs. This section presents the details of our framework.

5.1 Phase Oracle

We aim to coherently achieve the phase encoding for the cost function of the QNN by a phase oracle O_{Phase}, which acts as

$$O_{Phase} |\boldsymbol{\theta}\rangle |\mathbf{0}\rangle_{QNN+ancillas} \rightarrow e^{-i\gamma C(\boldsymbol{\theta})} |\boldsymbol{\theta}\rangle |\mathbf{0}\rangle_{QNN+ancillas} \tag{5.1}$$

in which γ is a parameter to be optimized. When O_{Phase} is acting on a superposition state of parameters $\sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle$, we have

$$O_{Phase} \sum_{\boldsymbol{\theta}} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle |\mathbf{0}\rangle_{QNN+ancillas} \rightarrow \sum_{\boldsymbol{\theta}} e^{-i\gamma C(\boldsymbol{\theta})} \omega_{\boldsymbol{\theta}} |\boldsymbol{\theta}\rangle |\mathbf{0}\rangle_{QNN+ancillas} \tag{5.2}$$

As detailed in Ref. [4], this phase oracle can be constructed based on the amplitude encoding which we have implemented in Section 4.1.1. Next, we present the details of how to construct the phase oracle from the amplitude encoding by amplitude estimation or Linear Combination of Unitaries (LCU) [6].

Phase oracle by amplitude estimation The procedure to achieve O_{Phase} by amplitude estimation is very similar to that of $\mathcal{O}_{Grover}$, the only difference is that the threshold U_O (defined in Eq. 4.17) needs to be replaced by U'_O which acts as

$$U'_O \left|\pm 2\theta_j\right\rangle = e^{-i\gamma C(\boldsymbol{\theta})} \left|\pm 2\theta_j\right\rangle. \tag{5.3}$$

Recall Eq. 2.8, Eq. 2.16 and the form of the cost function in Table 4.1, the cost function $C(\boldsymbol{\theta})$ is encoded in θ_j as $C(\boldsymbol{\theta}) = -\cos 2\theta_j$, therefore U'_O acts as

$$U'_O \left|\pm 2\theta_j\right\rangle = e^{i\gamma \cos 2\theta_j} \left|\pm 2\theta_j\right\rangle. \tag{5.4}$$

Once we have chosen the specific value of γ, U'_O can be constructed according to Eq. 5.4.

Phase oracle by LCU For this approach, we start with constructing an operator G^* defined similarly as in Eq. 4.13: [1]

$$G^* := C_2 U^{-1} C_1 U, \tag{5.5}$$

It has been shown in Ref. [4] that

$$e^{-i\frac{1}{2}(C(\boldsymbol{\theta})-1)} \cdot I \approx \sum_{m=-M}^{M} \beta_m G^{*m}, \tag{5.6}$$

where $\beta_m = \sum_{k=|m|}^{M} \begin{pmatrix} 2k \\ k-m \end{pmatrix} \frac{(-1)^m i^k}{k! 2^{2k}}$, $M \in \mathbb{N}_+$.

Define a new cost function $C'(\boldsymbol{\theta}) := \frac{1}{2}(C(\boldsymbol{\theta}) - 1)$ (optimizing $C'(\boldsymbol{\theta})$ is equivalent to optimizing $C(\boldsymbol{\theta})$), we have

$$e^{-iC'(\boldsymbol{\theta})} \cdot I \approx \sum_{m=-M}^{M} \beta_m G^{*m}. \tag{5.7}$$

$\sum_{m=-M}^{M} \beta_m G^{*m}$ can be implemented using the LCU technique (together with the subsequent "Oblivious Amplitude Amplification") [6] in which the number of calls to G^* needed is only logarithmic of the inverse of the desired precision [4]. Using the techniques in Ref. [7], we can convert phase oracle with $e^{-iC'(\boldsymbol{\theta})}$ into phase oracle with $e^{-i\gamma C'(\boldsymbol{\theta})}$ for arbitrary γ bounded from $[-1, 1]$), by only logarithmic (of the inverse of the desired precision) number of queries of phase oracle with $e^{-iC'(\boldsymbol{\theta})}$.

In Fig. 5.1 we summarise the two approaches for the Phase encoding of the cost function.

[1] Note that here our definition of G^* is slightly different from the G_U in Ref. [4]: C_1 and C_2 being negative to their counterpart in the definition of G_U. However the two negative signs cancel, therefore we have $G^* = G_U$.

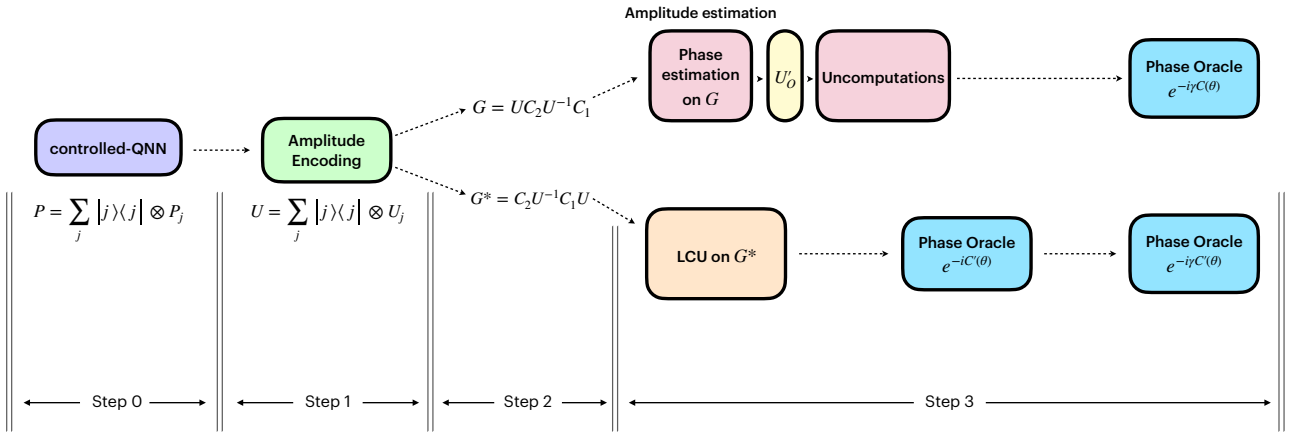

Figure 5.1: *Pipeline of the construction of the phase oracle.* Here we summarise the two approaches by amplitude estimation and by LCU for the Phase encoding of the cost function. Step 0: Creating superposition for QNN with different parameters, which is implemented by "controlled-QNN" (see Fig. 2.8), denoted by $P = \sum_j |j\rangle\langle j| \otimes P_j$. Step 1: Amplitude encoding of the cost function, by the unitary operation $U = \sum_j |j\rangle\langle j| \otimes U_j$. Step 2: Constructing the "Grover Operator" upon the amplitude encoding unitary. In the approach using amplitude estimation, the Grover Operator G is constructed as $G = UC_2U^{-1}C_1$. In the approach using LCU, the Grover Operator G^* is constructed as $G^* = C_2U^{-1}C_1U$. Step 3: Phase encoding of the cost function, by amplitude estimation(upper path) or by LCU(lower path). In the upper path, the Phase Oracle is achieved by phase estimation on G, threshold oracle U_O', and uncomputation. In the lower path, LCU on G^* (together with the subsequent "Oblivious Amplitude Amplification") [6] realizes $e^{-iC'(\boldsymbol{\theta})}$ which is then converted to the Phase Oracle with arbitrary γ —$e^{i\gamma C'(\boldsymbol{\theta})}$ using the method in Ref. [7]. $C'(\boldsymbol{\theta}) := \frac{1}{2}(C(\boldsymbol{\theta}) - 1)$ is a new cost function, optimizing $C'(\boldsymbol{\theta})$ is equivalent to optimizing $C(\boldsymbol{\theta})$.

5.2 Adaptive Mixers

As in Section 2.2, we designed a new variant of QAOA — "Adaptive-Continuous(AC-QAOA)" — to be the ansatz of our quantum training for QNN. We summarise the reason of this choice as follows:

1. [**Why "Continuous"**] In our optimisation problem of QNN training, the parameters we are optimizing (the angles of rotation gates) are continuous variables (real values), hence the choice of mixer Hamiltonian has to be designed for continuous variables. For example, the mixer Hamiltonian of the original QAOA (X rotations) generates shifts in the computational basis, here in the continuous case, the corresponding mixer should shift the value for each digitized continuous variable stored in independent registers.

2. [**Why "Adaptive"**] The Cost function of QNNs is complicated and task-specific (given by the learning objectives). Hence it is non-trivial to analytically determine good mixers for our optimisation problem of QNN training. Therefore, we would want to take advantage of including alternative mixers and allowing adaptive mixers for different layer (as in

ADAPT-QAOA).

Adopting "AC-QAOA" could exploit hidden structures in the QNN optimisation problem and dramatically shorten the depth of QAOA layers while significantly improving the quality of the solution [91].

Generally, the mixer pool of AC-QAOA should include two types of Mixer Hamiltonians for continuous variables:

1. Quadratic functions of the position operator and the momentum operator for single continuous variables, e.g. the squeezing operator [92].
2. Entangling mixers that acts on two continuous variables, e.g. the two-mode squeezing operator [92].

These operators could be carried out in continuous-variable quantum systems. However, we focus on the circuit implementation of these mixers when using a collection of qubits to approximate the behaviour of continuous variables.

When using a qudit of dimension d to digitally simulate a continuous variable, the position operator can be written as

$$J_d := \sum_{j=0}^{d-1} j \left|j\right\rangle\!\left\langle j\right|, \tag{5.8}$$

in which j is the digitized value of the continuous variable.

We can use N qubits to simulate the qudit and construct J_d for $d = 2^N$ as [3],

$$J_{2^N} = \sum_{n=1}^{N} 2^{n-2}(I^{(n)} - Z^{(n)}), \tag{5.9}$$

where $I^{(n)}$ and $Z^{(n)}$ are the identity and the Pauli-Z operator (respectively) for the n^{th} qubit.

The momentum operator, which act as generator of shifts in the value of a continuous variable (denoted as S) can be written as the discrete Fourier transform of J_d [3],

$$S := F_d^\dagger J_d F_d, \tag{5.10}$$

in which the discrete Fourier transform F_d is defined by

$$F_d \left|j\right\rangle := \frac{1}{\sqrt{d}} \sum_{k=0}^{d-1} \omega_d^{-jk} \left|k\right\rangle, \tag{5.11}$$

where $\omega_d := e^{2\pi i/d}$.

As mentioned above, a general mixer Hamiltonian is the quadratic functions of the position operator J_d and the momentum operator S, therefore using Eq. 5.9 and Eq. 5.10 (set $d = 2^N$) we can rewrite a mixer Hamiltonian as a summation of simple unitaries. Hence utilising the Hamiltonian simulation technique in [6], the Mixer operator can be efficiently implemented. For instance, the digitized version of the generator of the squeezing operator (denote as T) is defined as:

$$T := J_d S + S J_d. \tag{5.12}$$

Plugging Eq. 5.10 into Eq. 5.12, together with Eq. 5.9 (set $d = 2^N$), we can see that $T = J_d F_d^\dagger J_d F_d + F_d^\dagger J_d F_d J_d$ can be expressed as the summation of simple unitaries. Therefore the corresponding Mixer with Hamiltonian T can be efficiently implemented using the Hamiltonian simulation technique in [6]. Similarly, the entangling Mixers on two continuous variables with Hamiltonian $S_i S_j, S_i T_j, T_i T_j$ (The subscript i, j indicate that they are for specific variables) can be implemented in the same manner. In Fig. 5.2, we depict the schematic diagram of applying AC-QAOA to QNN training.

Due to the fact that the non-Gaussian operators are costly to implement, we only consider up-to-second-order polynomial functions of the position operator J_d and the momentum operator S for the Mixer Hamiltonian. The Mixer pool can generally include mixers with Hamiltonians: J_d, S, $J_d S$, $S J_d$, ${J_d}^2$, S^2, ${J_d}^2 + S^2$ for one continuous variable and the entangling Mixers for two continuous variables. Comparing to the Mixer pool of ADAPT-QAOA for discrete variables, we can have the following analogy:

1. The momentum operator S is the (digitized) continuous version of X mixers that shift the value for each digitized continuous variable stored in independent registers.

2. $J_d S$ is the (digitized) continuous version of Y mixers which 'unlock' geodesics in parameter space, allowing the QAOA iterations to reach the target state faster. [20]

We note that quadratic Hamiltonians are efficiently simulatable (classically), but only when the initial state is from a special class of Gaussian states (e.g. the vacuum state) [93]. Here, the initial state in the qubit encoding is far from Gaussian and a continuous variable analog of our technique would use an equivalent encoding.

By making the mixers flexible and adaptive to specific optimisation problems, it is demanding to find an efficient way of determining the mixers' sequence and optimizing the hyper-parameters. There are several research works on using machine learning approaches (Recurrent Neural Networks (RNN) and Reinforcement Learning(RL)) to determine the mixers' sequence and optimize the hyper-parameters. These works achieved significantly fewer measurements than the conventional approach(e.g. gradient-based methods). We list the papers in the following table:

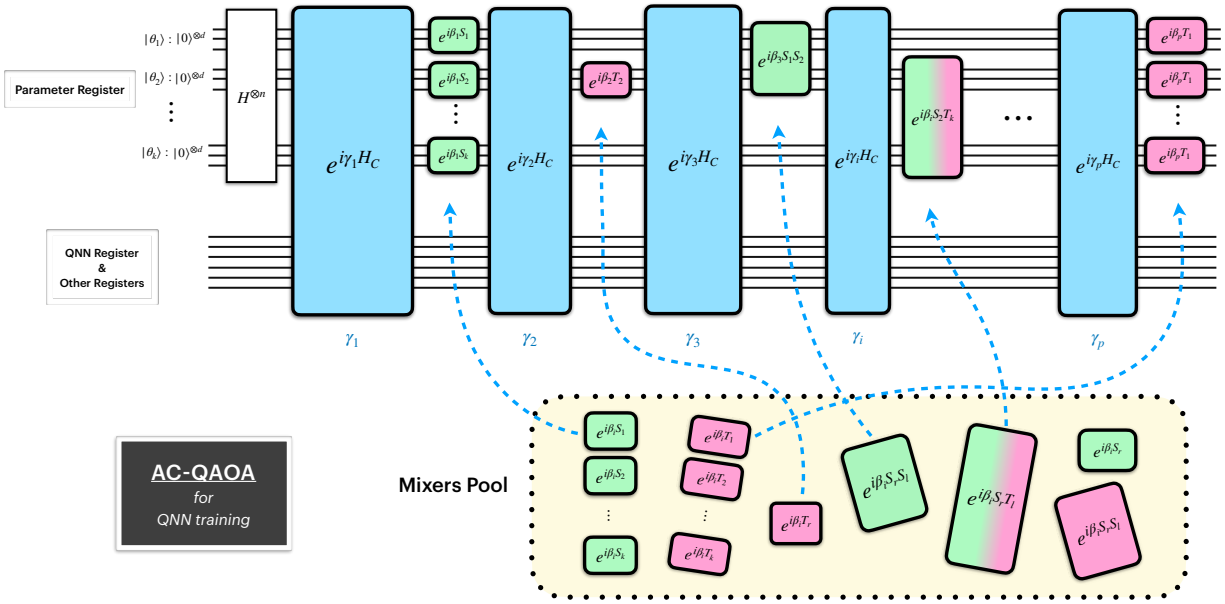

Figure 5.2: *Schematic diagram of applying AC-QAOA to QNN training.* AC-QAOA is a variant of QAOA we designed for solving optimisation of continuous variables with the short-depth advantage of QAOA layers, see Fig. 2.7. This figure illustrates applying AC-QAOA to QNN training, following the scheme in Fig. 3.2. The quantum training protocol consists of alternating operations in a QAOA fashion — the first operation acts on both the parameter register and QNN register to encode the cost function of QNN onto a relative phase of the parameter state. This operation is represented by the blue blocks in the figure. The other operations are the Mixers (green and pink boxes) which act only on the parameter register. In the parameter register, θ_i are the continuous variables to be optimized in the training, each θ_i is digitized into binary form and stored in an independent register. The overall process of AC-QAOA is similar to that of the original QAOA, with the difference being as follows. *1.* The mixers of AC-QAOA with Hamiltonians S_i and T_i are acting on the registers of θ_i (rather than single qubits as in the original QAOA). *2.* The mixers of AC-QAOA contain alternative mixers taken from a *mixers pool* and can vary from layer to layer.

	RNN	**RL**
Determining mixers sequence	Ref. [89]	Ref. [20]
Optimizing hyper-parameter	Ref. [80]	Refs. [88, 20, 94, 95]

We adopt the approaches developed in these works to our quantum training of QNNs for efficiently determining the mixers' sequence and optimizing the hyper-parameters.

5.3 Advantages of training by QAOA

As we have discussed in 4.2, due to the global-search nature of Grover's algorithm, the quantum training using Grover Adaptive Search can circumvent the noise-free barren plateau, however it has certain limitations and disadvantages such as: 1. cannot handle the noise-induced barren plateau; 2. requires an exponential number of calls to the "controlled-QNN" with excessive lengths of circuit and run time.

In contrast, our quantum training using adaptive continuous QAOA could eliminate the limitations of using Grover Adaptive Search and the advantages come in the following two folds:

1. The phase oracle by LCU approach does not explicitly evaluate/store the value of the cost function at any stage of the algorithm and the number of calls to the "controlled-QNN" scales only logarithmic with respect to the inverse of the desired precision [4]. Therefore the phase encoding is not affected by the noise-induced barren plateau for which the precision required is exponentially small. This is better than the case using Grover Adaptive Search.

2. The adaptive mixers can dramatically reduce the number of QAOA iterations while significantly increasing the quality of the output solution. This will enable our quantum training to achieve high performance within relatively shallow circuit and short run time. Thanks to the phase encoding faithfully conserving all the information and structure in the cost function, our adaptive QAOA protocol can exploit hidden structures in the QNN training problem. (Whereas, the Grover Oracle 'cuts off' the cost function with the threshold effectively losing some information and structure in the cost function.) Therefore, adaptive QAOA can offer beyond-Grover speed-up. Moreover, numerical experiments in [20] show that when using the adaptive approach, the depth of the QAOA steps can be independent of the problem size (number of qubits), this would yield even more advantage when system size scales up.

5.4 Applications

In this section, we discuss several applications of QNN to which our quantum training algorithm can be applied. For each application, we first briefly illustrate the usage of QNN and the corresponding cost function for the task, then we present the way of amplitude encoding tailored for this application. Based on the amplitude encoding, the construction of the full quantum training algorithm is similar for every application.

5.4.1 Training VQE

Variational quantum eigensolvers (VQEs) utilize QNN to estimate the eigenvalue corresponding to some eigenstate of a Hamiltonian. The most common instance is ground state estimation in

which the QNN (a parameterized circuit ansatz) is applied to an initial state (the zero state) over multiple qubits to generate the ground state. The parameters in the QNN are optimized so that the generated state of the QNN possesses the lowest expectation value of the given Hamiltonian. A schematic of VQE for ground state estimation is presented in Fig. 5.3.

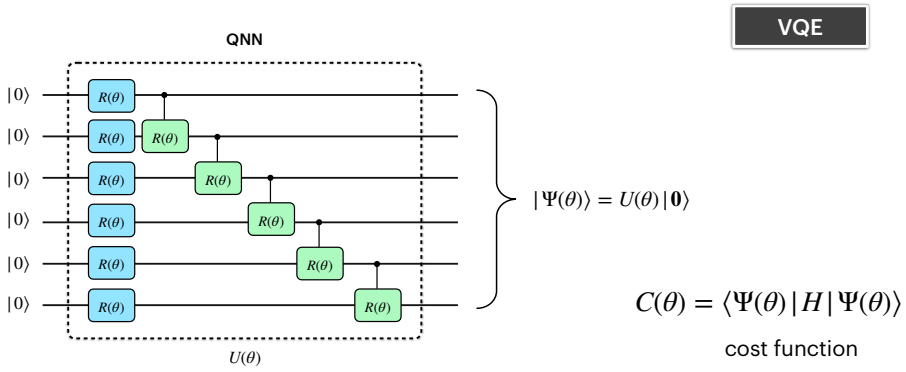

Figure 5.3: *Schematic of VQE for ground state estimation.* The QNN (a parameterized circuit ansatz) is applied to an initial state (e.g. the zero state) over multiple qubits to generate the ground state of a given Hamiltonian H. The parameters in the QNN, i.e. the rotation angles of the parametrized gates (here for simplicity we use the same symbol θ for all the angles of different gates), are optimized so that the generated state of the QNN possess the lowest expectation value of the given Hamiltonian.

Consider a Hamiltonian

$$H = \sum_i a_i U_i$$

where U_i is a unitary, $a_i > 0$ and $\sum_j a_i = 1$. (This assumption can be made without loss of generality by renormalizing the Hamiltonian and absorbing signs into the unitary matrix.) Let the state $|\psi(\boldsymbol{\theta})\rangle$ for $\boldsymbol{\theta} \in \mathbb{R}^m$ be the variational state prepared by the QNN. (m is the number of parameters in the QNN.) The cost function of the QNN is:

$$C(\boldsymbol{\theta}) = \langle\psi(\boldsymbol{\theta})| \sum_i a_i U_i |\psi(\boldsymbol{\theta})\rangle \,. \tag{5.13}$$

Our goal is then to estimate

$$\boldsymbol{\theta}^* = \underset{\boldsymbol{\theta}}{argmin} \left(\langle\psi(\boldsymbol{\theta})| \sum_i a_i U_i |\psi(\boldsymbol{\theta})\rangle \right). \tag{5.14}$$

Here we use the technique "Linear Combinations of Unitaries"(LCU) [8] to implement the Hamiltonian. Define new unitary oracles W, H_{LCU} such that

$$W|0\rangle = \sum_i \sqrt{a_i}\ |i\rangle \,, \tag{5.15}$$

$$H_{LCU} = \sum_i ii \otimes U_i. \tag{5.16}$$

The amplitude encoding of the cost function of the QNN can be implemented using the following circuit in Fig. 5.4 [4]:

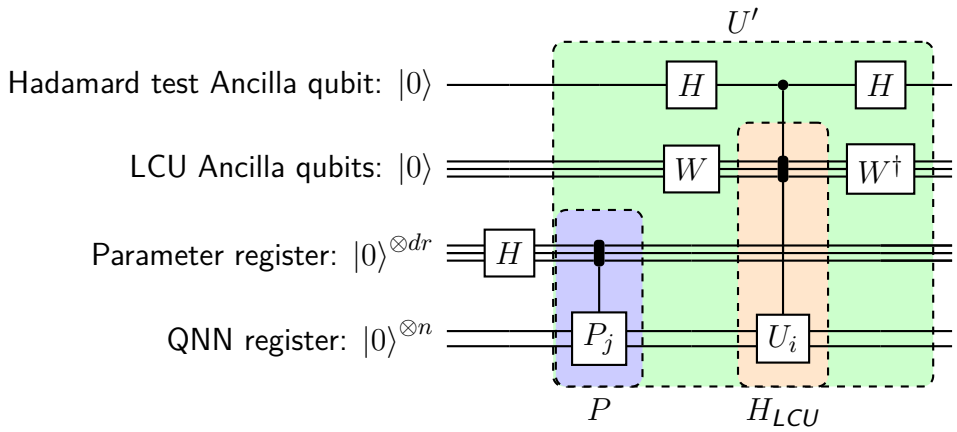

Figure 5.4: *Circuit for the amplitude encoding of the cost function for VQE.* Here we use the Hadamard Test Circuit for the amplitude encoding of the cost function, as detailed in 4.1.1. We use a technique "Linear Combinations of Unitaries"(LCU) [8] to implement the given Hamiltonian $H = \sum_i a_i U_i$. The unitary oracles W, H_{LCU} are defined as $W|0\rangle = \sum_i \sqrt{a_i}\ |i\rangle, H_{LCU} = \sum_i ii \otimes U_i$.

5.4.2 Learning to generate a pure state

Another application of our quantum training is when QNN is served as a generative model to learn a pure state. In our scenario, the target state is generated by a given unitary (e.g. a given sequence of gates), the QNN serves as another generator circuit for the target state. The parameters in QNN are optimized such that the generated state of QNN matches the target state. This approach can be used to transform a given sequence of gates to a different/simpler sequence (e.g. translating circuits from superconducting gate sets to ion trap gate sets) A schematic of this application is presented in Fig. 5.5.

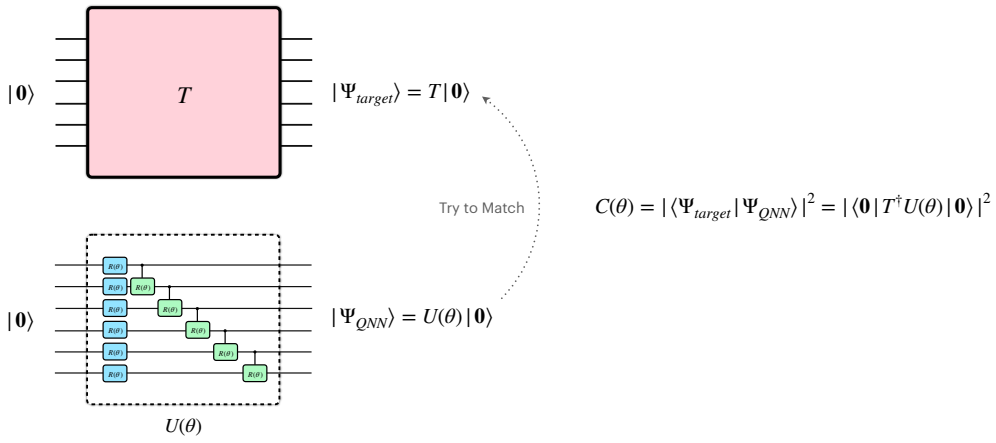

Figure 5.5: *Schematic of using QNN to generate a pure state.* In our scenario, the target state is generated by a given Unitary T, i.e. $|\Psi_{target}\rangle = T|0\rangle$), the QNN (denoted as $U(\theta)$) serves as another generator circuit for the target state. The parameters in QNN are optimized such that the generated state of QNN $\left|\Psi_{QNN}\right\rangle$ matches the target state. The cost function is the fidelity between the target state and the generated state by QNN.

The amplitude encoding for this application has been given in section 4.1.1.

5.4.3 Training a Quantum Classifier

Finally, we discuss the application of QNN as a quantum classifier that performs supervised learning which is a standard problem in machine learning.

To formalise the learning task, let $\mathcal{X}$ be a set of inputs and $\mathcal{Y}$ a set of outputs. Given a dataset $\mathcal{D} = \{(x_1, y_1), ..., (x_M, y_M)\}$ of pairs of so called *training inputs* $x_m \in \mathcal{X}$ and *target outputs* $y_m \in \mathcal{Y}$ for $m = 1, ..., M$, the task of the model is to predict the output $y \in \mathcal{Y}$ of a new input $x \in \mathcal{X}$. For simplicity, we will assume in the following that $\mathcal{X} = \mathbb{R}^N$ and $\mathcal{Y} = \{0, 1\}$, which is a binary classification task on a N-dimensional real input space. In summary, the quantum classifier aims to learn an effective labeling function $\ell : \mathcal{X} \to \{0, 1\}$.

Given an input x_i and a set of parameters $\boldsymbol{\theta}$, the quantum classifier first embeds x_i into the state of a n-qubit quantum system via a state preparation circuit S_{x_i} such that $S_{x_i} |0\rangle = |\varphi(x_i)\rangle$, and subsequently uses a learnable quantum circuit $U(\boldsymbol{\theta})$ (QNN) as a predictive model to make inference. The predicted class label $y^{(i)} = f(x_i, \boldsymbol{\theta})$ is retrieved by measuring a designated qubit in the state $U(\boldsymbol{\theta}) |\varphi(x)\rangle$. A schematic of the quantum classifier is presented in Fig. 5.6. Note although the variational quantum classifier could be operated as a multiclass classifier, here we limit ourselves to the case of the binary classification discussed above and cast the multi-label tasks as a set of binary discrimination subtasks.

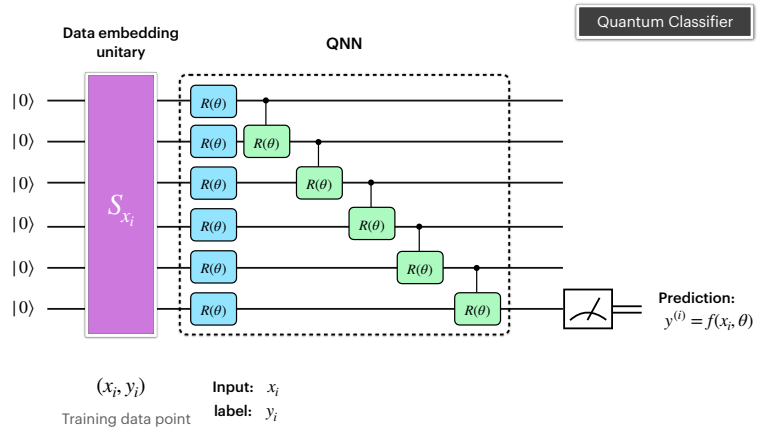

Figure 5.6: *Schematic of a quantum classifier.* For a training data point (x_i, y_i), the quantum classifier first embeds x_i into the state of a n-qubit quantum system via a data embedding circuit S_{x_i} (purple box) such that $S_{x_i} |0\rangle = |\varphi(x_i)\rangle$, and subsequently uses a learnable quantum circuit $U(\boldsymbol{\theta})$ (QNN) as a predictive model to make inference (here for simplicity we use the same symbol θ for all the angles of different gates). The predicted class label $y^{(i)} = f(x_i, \boldsymbol{\theta})$ is retrieved by measuring a designated qubit in the state $U(\boldsymbol{\theta}) |\varphi(x)\rangle$.

Denote $p(\lambda)$ as the probability of the measurement result on the designated qubit being λ ($\lambda \in \{0,1\}$). The cost function of each training data point $L_i(\boldsymbol{\theta})$, as a function of y_i and $y^{(i)}$ and hence a function of $y_i, x_i, \boldsymbol{\theta}$ which we denote as $L(x_i, y_i, \boldsymbol{\theta})$, is chosen to be the probability of the measurement result on the designated qubit being identical to the given label [69], namely:

$$L_i(\boldsymbol{\theta}) = L(x_i, y_i, \boldsymbol{\theta}) := p(y_i). \tag{5.17}$$

Note here the larger the probability is, the more correct the prediction is, so we want to maximize the cost (in this paper, to be coherent with the former narrative, we use "cost" instead of the commonly used "likelihood" of inferring the correct label for a data sample.)

On the other hand, the quantum state of the system after the state preparation and QNN inference can be written as:

$$|\Psi_i(\boldsymbol{\theta})\rangle = U(\boldsymbol{\theta})\,|\varphi(x_i)\rangle = \sqrt{p(0)}\,|0\rangle|u_{\boldsymbol{\theta}}\rangle + \sqrt{1-p(0)}\,|1\rangle|v_{\boldsymbol{\theta}}\rangle \tag{5.18}$$

$$= \begin{cases} \sqrt{p(y_i)}\,|0\rangle|u_{\boldsymbol{\theta}}\rangle + \sqrt{1-p(y_i)}\,|1\rangle|v_{\boldsymbol{\theta}}\rangle, & y_i = 0 \\ \sqrt{p(y_i)}\,|1\rangle|u_{\boldsymbol{\theta}}\rangle + \sqrt{1-p(y_i)}\,|0\rangle|v_{\boldsymbol{\theta}}\rangle, & y_i = 1 \end{cases} \tag{5.19}$$

in which $|u_{\boldsymbol{\theta}}\rangle, |v_{\boldsymbol{\theta}}\rangle$ are some normalized state that depend on $\boldsymbol{\theta}$.

From Eqs. 5.17 and 5.19 we can see that the cost of each data sample $L(x_i, y_i, \boldsymbol{\theta})$ is naturally encoded in the amplitude of the output state of QNN $|\Psi_i(\boldsymbol{\theta})\rangle$. We illustrate the amplitude encoding of the cost function for the quantum classifier in Fig. 5.7. Constructing the "controlled-QNN" will achieve the amplitude encoding for all the parameter configurations in parallel. Based on this amplitude encoding, we can construct $e^{-i\gamma L(x_i, y_i, \boldsymbol{\theta})}$ using the methods discussed in section 5.1. [2]

The total cost function of the whole training set can be defined as: (for simplicity we omit $\frac{1}{M}$ here)

$$C(\boldsymbol{\theta}) = \sum_i L(x_i, y_i, \boldsymbol{\theta}). \tag{5.20}$$

It follows immediately

$$e^{-i\gamma C(\boldsymbol{\theta})} = \Pi_i e^{-i\gamma L(x_i, y_i, \boldsymbol{\theta})}. \tag{5.21}$$

Therefore the phase encoding of the total cost function can be implemented by accumulating individual phase encoding for each training sample, this process can be illustrated in Fig. 5.8.

Armed with the phase encoding of the total cost function, we can now construct the full quantum training protocol as in Fig. 5.9.

[2] Note that for the training data point with $y_i = 0$, C_1 in the Grover Operator G and G^* has to be adjusted to $-Z$

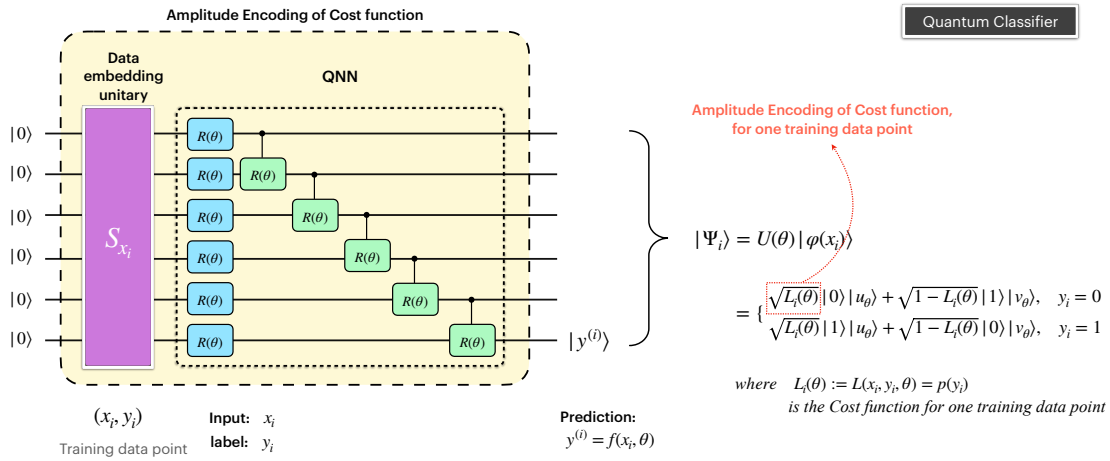

Figure 5.7: *Amplitude encoding of the cost function for the quantum classifier.* For a training data point (x_i, y_i), the quantum classifier first embeds x_i into the state of a n-qubit quantum system via a data embedding circuit S_{x_i} (purple box) such that $S_{x_i}|0\rangle = |\varphi(x_i)\rangle$, and subsequently uses a learnable quantum circuit $U(\boldsymbol{\theta})$ (QNN) as a predictive model to make inference (here for simplicity we use the same symbol θ for all the angles of different gates). The predicted class label $y^{(i)} = f(x_i, \boldsymbol{\theta})$ is retrieved by measuring a designated qubit in the state $U(\boldsymbol{\theta})|\varphi(x)\rangle$. Denote $p(\lambda)$ as the probability of the measurement result on the designated qubit being λ ($\lambda \in \{0,1\}$). The cost function of each training data point $L_i(\boldsymbol{\theta})$, as a function of y_i and $y^{(i)}$ and hence a function of $y_i, x_i, \boldsymbol{\theta}$ which we denote as $L(x_i, y_i, \boldsymbol{\theta})$, is chosen to be the probability of the measurement result on the designated qubit being identical to the given label, namely: $L_i(\boldsymbol{\theta}) = L(x_i, y_i, \boldsymbol{\theta}) := p(y_i)$. We can see that the cost of each data sample is naturally encoded in the amplitude of the output state of QNN.

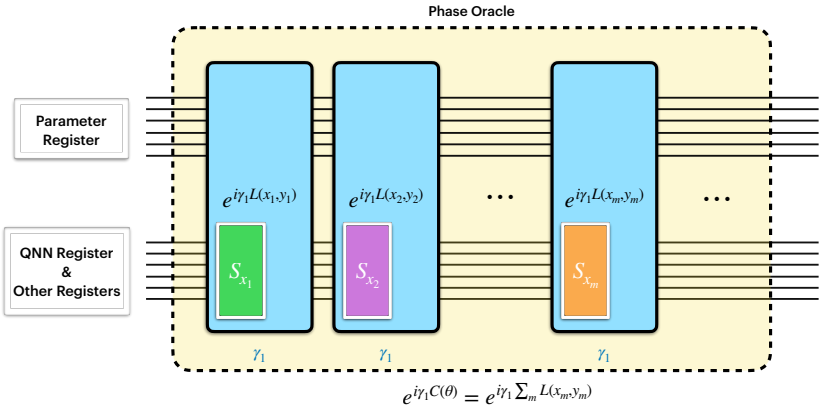

Figure 5.8: *Phase encoding of the total cost function of quantum classifier.* The total cost function of the whole training set can be defined as: $C(\boldsymbol{\theta}) = \sum_i L(x_i, y_i, \boldsymbol{\theta})$. It follows immediately $e^{-i\gamma C(\boldsymbol{\theta})} = \Pi_i e^{-i\gamma L(x_i, y_i, \boldsymbol{\theta})}$. Therefore the phase encoding of the total cost function (the overall yellow box) can be implemented by accumulating individual phase encoding for each training sample(blue boxes). In this figure, we omit $\boldsymbol{\theta}$ in $L(x_i, y_i, \boldsymbol{\theta})$ for simplicity. The inner boxes in the blue boxes represent different data embedding unitaries for the training data points.

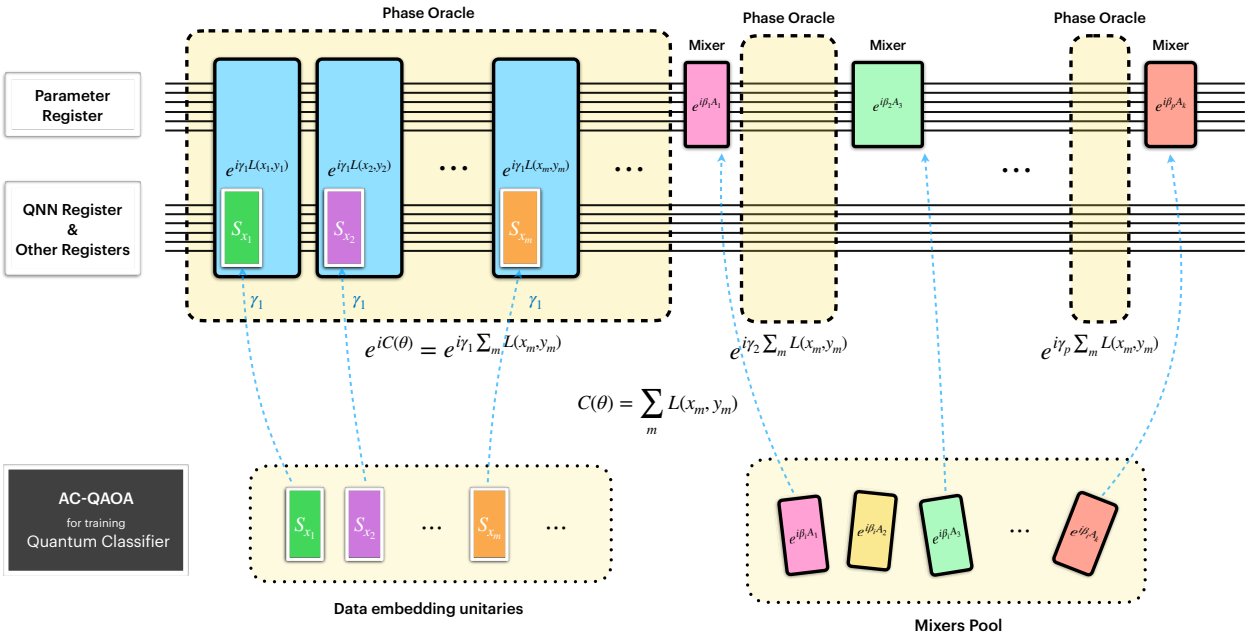

Figure 5.9: *Schematic of our quantum training protocol for the quantum classifier.* The full quantum training protocol consists of the alternation of the Phase Oracle that achieves coherent phase encoding of the cost function and the Adaptive Mixers chosen from a Mixers pool. The phase encoding of the total cost function for the quantum classifier is detailed in Fig. 5.8. The total cost function of the whole training set can be defined as $C(\boldsymbol{\theta}) = \sum_i L(x_i, y_i, \boldsymbol{\theta})$. It follows that $e^{-i\gamma C(\boldsymbol{\theta})} = \Pi_i e^{-i\gamma L(x_i,y_i,\boldsymbol{\theta})}$. Therefore the Phase Oracle for the total cost function (the yellow boxes in the upper part of this figure) can be implemented by accumulating individual phase encoding for each training sample(blue boxes). In this figure, we omit $\boldsymbol{\theta}$ in $L(x_i, y_i, \boldsymbol{\theta})$ for simplicity. The colourful boxes with white borders represent different data embedding unitary for the training data points. The colourful boxes with a black border (excluding the blue ones for the Phase encoding) represent different Mixers chosen from a Mixers Pool.

Part II

QNN for graph-structured data

Chapter 6

Quantum Graph Neural Networks: Overview

Quantum Neural Networks (QNNs) lacking prior knowledge of target problems do not exploit the underlying structure of data, which leads to issues on trainability and generalization[30], they also encounter a quantum version of the no-free lunch theorem[96]. As a result, there is a demand for designing problem-tailored QNN model. In the second part of the thesis including Chapter 6 7 8 9, we design problem-tailored QNNs for graph-structured data by incorporating the inductive biases into their architectures. Specifically, we devise QNN architectures in accordance with three major types of classical Graph Neural Networks(GNNs): Graph Convolutional Neural Networks, Graph Attention Neural Networks, Message-Passing GNNs. This chapter gives an overview of the background. A brief introduction of the three classical GNNs are given in Section 6.1. Section 6.2 presents the related works and Section 6.3 describes our framework for Quantum Graph Neural Networks.

6.1 Graph Neural Networks

Graphs are a ubiquitous representation of data for a variety of real-world applications in social networks, biological networks, transportation systems, etc.[97] Graph Neural Networks (GNNs) are powerful machine learning models for analyzing structured data represented as graphs. They have shown remarkable success in various applications including social network analysis, recommendation systems, antibacterial discovery, physics simulations, fake news detection and traffic prediction[97]. In this section, we provide a brief introduction to classical Graph Neural Networks, which serve as the foundation for the development of our quantum GNNs.

Graphs are made up of nodes and edges, with nodes representing entities or objects and edges representing the connections between them. We can represent a graph with an adjacency matrix $\mathbf{A}$ and node features $\mathbf{X}$. GNN architectures are permutation-equivariant functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$

constructed by applying shared permutation-invariant functions ϕ over local neighborhoods[1]. This local function ϕ is sometimes referred to as "graph diffusion", "propagation", or "message passing", and the overall computation of such $\mathbf{F}$ is known as a "GNN layer"[1]. The design and study of GNN layers is a rapidly expanding area of deep learning, and the literature can be divided into three "flavours" (of GNN layers)[1]: convolutional, attentional, and message-passing (see Figure 6.1). These flavours determine the extent to which ϕ transforms the neighbourhood features, allowing for varying levels of complexity when modelling interactions across the graph.

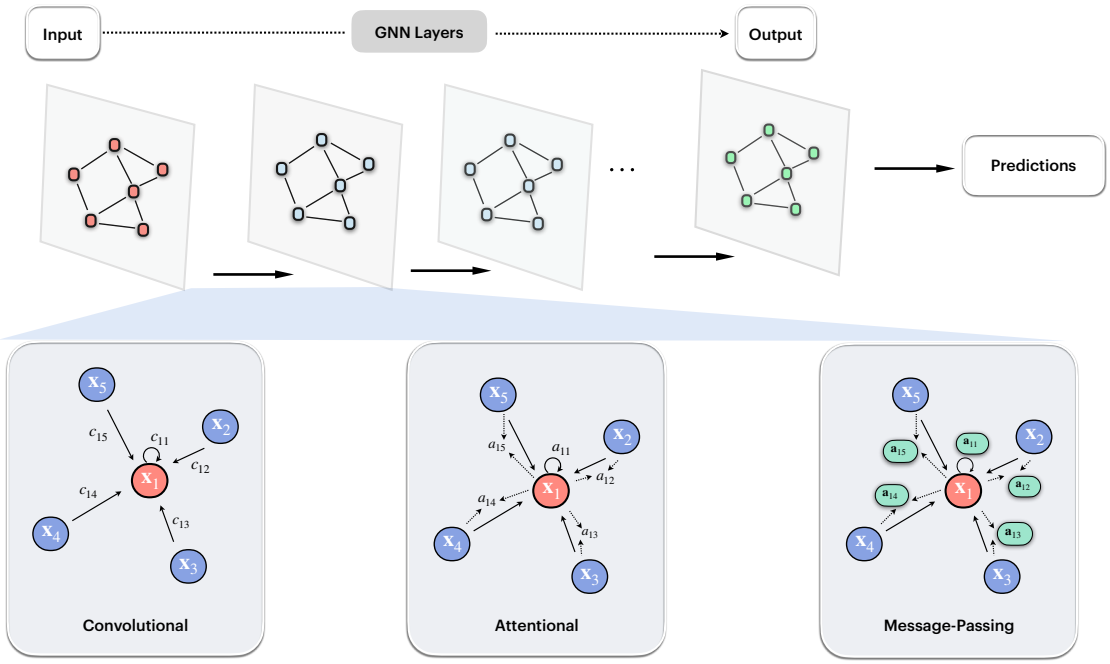

Figure 6.1: *GNN pipeline and three "flavours" of GNN layers[1]* GNN architectures are permutation equivariant functions $\mathbf{F}(\mathbf{X}, \mathbf{A})$ constructed by applying shared permutation invariant functions ϕ over local neighbourhoods. This local function ϕ is sometimes referred to as "diffusion", "propagation", or "message passing", and the overall computation of such $\mathbf{F}$ is known as a "GNN layer". The design and study of GNN layers is a rapidly expanding area of deep learning, and the literature can be divided into three "flavours" (of GNN layers): convolutional, attentional, and message-passing. These flavours determine the extent to which ϕ transforms the neighbourhood features, allowing for varying levels of complexity when modelling interactions across the graph.

In the convolutional flavour (e.g.[11, 98]), the features of the neighbouring nodes are directly combined with fixed weights,

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} c_{uv}\psi\left(\mathbf{x}_v\right)\right).$$

Here, c_{uv} is a constant indicating the significance of node v to node u' s representation. $\bigoplus$ is the aggregation operator which is often chosen to be the summation. ψ and ϕ are learnable

affine transformations[1]: $\psi(\mathbf{x}) = \mathbf{W}\mathbf{x} + \mathbf{b}$, $\phi(\mathbf{x}, \mathbf{z}) = \mathbf{W}\mathbf{x} + \mathbf{U}\mathbf{z} + \mathbf{b}$, where $\mathbf{W}, \mathbf{U}, \mathbf{b}$ are learnable parameters. For simplicity, we omit $\mathbf{b}$ in our quantum case.

In the attentional flavour (e.g.[17]), a learnable self-attention mechanism is used to compute the importance coefficients $\alpha_{uv} = a(\mathbf{x}_u, \mathbf{x}_v)$. When $\bigoplus$ is the summation, the aggregation is still a linear combination of the neighbourhood node features, but the weights are now dependent on the features. This is represented by

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} a(\mathbf{x}_u, \mathbf{x}_v)\, \psi(\mathbf{x}_v)\right).$$

Finally, the message-passing flavour (e.g.[99]) involves computing arbitrary vectors ("messages") across edges,

$$\mathbf{h}_u = \phi\left(\mathbf{x}_u, \bigoplus_{v \in \mathcal{N}_u} \psi(\mathbf{x}_u, \mathbf{x}_v)\right).$$

Here, ψ is a trainable message function, which computes the vector sent from v to u, and the aggregation can be thought of as a form of message passing on the graph.

Classical GNNs have been shown to be highly effective in a variety of graph-related tasks including node classification(where the goal is to assign labels to nodes based on their attributes and the graph structure), link prediction(where the objective is to predict whether an edge exists between two nodes in the graph), graph classification(where the goal is to classify entire graphs based on their structures and attributes). Despite their success, classical GNNs also face challenges such as scalability—GNNs can be computationally expensive, particularly for large graphs.[97]

In the following three chapters of this thesis, we will devise and analyze QNN architectures in accordance with the three major types of classical GNNs(corresponding to the three flavours): Graph Convolutional Networks, Graph Attention Networks, Message-Passing GNNs. We term our QNN architectures as Quantum Graph Convolutional Networks, Quantum Graph Attention Networks, and Quantum Message-Passing GNNs which fall into the research area of Quantum Graph Neural Networks.

6.2 Related works

The area of Quantum Graph Neural Networks (QGNNs) has recently emerged as a promising avenue to leverage the power of quantum computing for graph representation learning. In this section, we provide an overview of relevant works in this area and highlight the key contributions

[1]Note we omitted the activation function in the original definition in [1], the quantum implementation of the activation function is described in Section 7.1.3

of our work.

Verdon et al. [100] proposed one of the first QGNN architectures, introducing a general framework based on Hamiltonian evolutions. While their work demonstrated the use of QGNNs for tasks like *learning quantum dynamics, creating multipartite entanglement in quantum networks, graph clustering, and graph isomorphism classification*, our architectures are specifically designed for tasks like *node classification* on classical graph-structured data. Furthermore, [100] suggests several future research directions for QGNNs, including quantum-optimization-based training and extending their QSGCNN to multiple features per node. Our work makes progress on both of these aspects: The design of our architectures enables quantum-optimization-based training [101] for our quantum GNNs, and our quantum GNN architectures natively support multiple features per node. While [100] provides a general framework for QGNNs, our work delves into the specifics of designing quantum circuits that closely mimic the functionality of classical GNNs and analyzes their potential quantum advantages, thus advancing the field in a complementary direction.

Beer et al. [102] designed quantum neural networks specifically for graph-structured quantum data. In contrast, our QGNNs are primarily designed to handle classical graph-structured data. Skolik et al. [103] proposed a PQC ansatz for learning on weighted graphs that respect equivariance under node permutations. In their ansatz, the node features are encoded in the rotation angles of the R_x gates, whereas in our GNN architecture, the node features are encoded directly in the amplitudes of the quantum state, enabling the usage of quantum linear algebra for the subsequent transformation.

Ai et al. [104] proposed DQGNN, which decomposes large graphs into smaller subgraphs to handle the limited qubit availability on current quantum hardware, however, subsampling techniques have reliability issues–it is challenging to guarantee that the subgraphs preserve the semantics of the entire graph and provide reliable gradients for training GNNs[105]. Tuysuz et al. [106] introduced a hybrid quantum-classical graph neural network (HQGNN) for particle track reconstruction. Mernyei et al. [107] proposed equivariant quantum graph circuits (EQGCs) as a unifying framework for QGNNs. In the niche of a quantum graph *convolutional* neural networks, detailed comparisons between our work (specifically, quantum GCN/SGC/LGC) and three other related works are provided in section 6.3.

6.3 Our Framework

Here in this section, we give an overview of our approach for Quantum Graph Convolutional Networks, as illustrated in Fig.6.2.

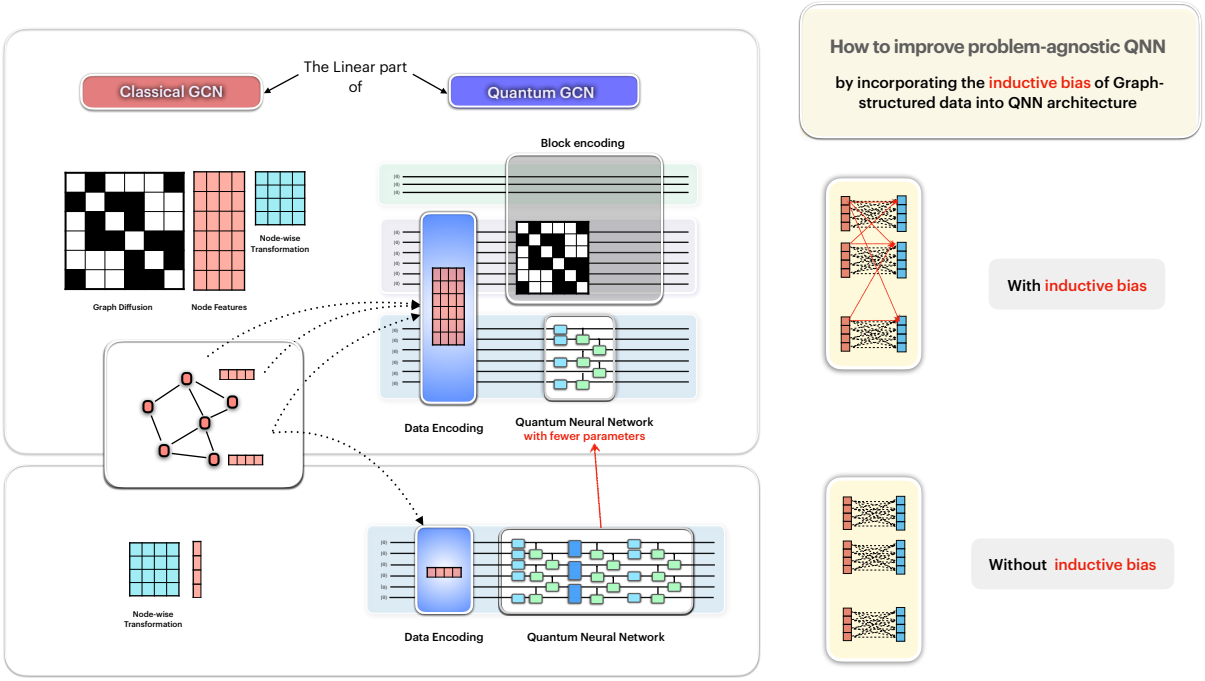

Figure 6.2: *Overview of our approach for Graph Convolutional Networks* We start with the scenario where neural networks (classical and quantum) process data without inductive bias, depicted as the lower part of this figure. In this scenario, the classical and quantum neural networks process each data point individually without acknowledging the connections between them. Here for a classical neural network, we depicted a linear layer represented as a matrix acting on a single data point as a vector. For the quantum neural network, we depicted a parametrized quantum circuit for implementing the linear layer as described in Section 2.1. In the upper part of this figure, we illustrate the scenario where classical and quantum GNNs process data with inductive bias of graph-structured data. In this scenario, the classical and quantum GNN process all the data points for every node on a graph, with cross-node connections between them. Here for classical GNN, we depicted the layer-wise linear transformation for multi-channel Graph Convolutional Networks (Section 7.2 provides the details): the trainable weight matrix(for node-wise transformation) and the renormalized adjacency matrix(for Graph diffusion) multiplied on the node feature matrix. In our Quantum GNN Architecture, this layer-wise linear transformation is implemented by applying the block-encoding of the renormalized adjacency matrix and a parameterized quantum circuit following a data encoding procedure(Section 7.2 provides the details). By incorporating graph inductive bias(here, the graph diffusion)into the architecture, our Quantum GNN can potentially operate with fewer parameters than its problem-agnostic counterpart. This can potentially lead to more efficient training and less overfitting, improving the conventional QNNs.

We start with the scenario where neural networks (classical and quantum) process data without inductive bias, depicted as the lower part of Fig.6.2. In this scenario, the classical and quantum neural networks process each data point individually without acknowledging the connections between them. Here for a classical neural network, we depicted a linear layer represented as a matrix acting on a single data point as a vector. For the quantum neural

network, we depicted a parametrized quantum circuit for implementing the linear layer as described in Section 2.1.

In the upper part of Fig.6.2, we illustrate the scenario where classical and quantum GNNs process data with inductive bias of graph-structured data. In this scenario, the classical and quantum GNN process all the data points for every node on a graph, with cross-node connections between them. Here for classical GNN, we depicted the layer-wise linear transformation for multi-channel Graph Convolutional Networks (Section 7.2 provides the details): the trainable weight matrix(for node-wise transformation) and the renormalized adjacency matrix(for Graph diffusion) multiplied on the node feature matrix. In our Quantum GNN Architecture, this layer-wise linear transformation is implemented by applying the block-encoding of the renormalized adjacency matrix and a parameterized quantum circuit following a data encoding procedure(Section 7.2 provides the details). By incorporating graph inductive bias(here, the graph diffusion)into the architecture, our Quantum GNN can potentially operate with fewer parameters than its problem-agnostic counterpart. This can potentially lead to more efficient training and less overfitting, improving the conventional QNNs.

In summary, while the related works share the high-level goal of developing quantum neural networks for graph-structured data, our work makes distinct contributions in the following aspects:

First, we propose novel QGNN architectures that are specifically designed to mirror the structure and functionality of popular classical GNN variants, namely Graph Convolutional Networks (GCNs), Graph Attention Networks (GATs), and Message Passing Neural Networks (MPNNs). This allows us to leverage the proven effectiveness of these architectures while harnessing the power of quantum computing.

Second, our quantum GNN architectures go beyond generic parameterized quantum circuits: For example, in our quantum graph convolutional networks, we employ quantum singular value transformation (QSVT) circuits to implement the spectral graph convolutions, enabling the application of learnable convolutional filters; in our quantum graph attention networks, we construct quantum circuits to evaluate and store attention scores, allowing the incorporation of self-attention mechanisms.

In the niche of quantum graph *convolutional* neural networks, we can compare our work with three other related works:

Hu et al. [108] designed a quantum graph convolutional neural network for semi-supervised node classification. While both works design quantum circuits to implement the graph convolu-

tional neural network, there are significant differences in the approaches. For data encoding, Hu et al. use N separate quantum circuits (N is the number of nodes), with each circuit encoding the features of a single node. In contrast, our work coherently encodes all N node features into a single quantum state on two entangled registers. For node-wise transformations, Hu et al. apply N subsequent parameterized quantum circuits (PQCs) acting on each of the separate circuits to implement the trainable weight matrix. For aggregation over neighborhood nodes, they first perform measurements on all N separate circuits to obtain the transformed node features, then regroup the features into different channels. For each channel, an N-dimensional vector is encoded into the amplitudes of a quantum state, resulting in C separate quantum circuits (C is the number of features/channels per node). They then utilize Givens rotations to perform aggregation over neighborhood nodes. In contrast, thanks to our data encoding scheme, we are able to simultaneously apply the block encoding of the normalized adjacency matrix (and further, QSVT for spectral convolution) and a single PQC for node-wise transformation on the two entangled registers, achieving both node-wise transformation and aggregation over neighborhood nodes simultaneously.

Zheng et al. [109] proposed a quantum graph convolutional neural network model to accomplish graph-level classification tasks. While both works aim to develop quantum versions of GCNs, there are several key differences. For data encoding, Zheng et al. use separate quantum circuits for each node, whereas our work coherently encodes all node features into a single quantum state on two entangled registers. Moreover, Zheng et al. focus on graph classification tasks, while our work focuses on node classification tasks (although our architectures are also well suited for graph classification tasks).

Chen et al. [110] proposed a parameterized quantum circuit architecture for quantum graph convolutional networks. Although both works design quantum circuits for implementing the adjacency matrix and the learnable weight matrix, the approaches differ. For aggregation over neighborhood nodes, Chen et al. use LCU to implement the adjacency matrix. In contrast, we utilize block encoding for the normalized adjacency matrix which enables the usage of QSVT for spectral graph convolutions and the corresponding higher order neighborhood propagation. (Although LCU effectively implements certain block-encoding, but a general block-encoding can accommodate more matrix–without the limitations of the approach used by Chen et al.) Notably, for cost-function evaluation, we only perform measurements on a single ancillary qubit, whereas they perform measurements on all the qubits. Another differentiation is that for implementation of the nonlinear activation function, we utilize NTCA for a two-layer GCN.

To conclude, our work makes significant contributions to the field of QGNNs by introducing beyond-generic-parameterized-quantum-circuits architectures aligned with classical GNNs. These advances complement and extend the existing literature on Quantum Graph Neural

Networks and lay the foundation for further research in this promising area.

Chapter 7

Quantum Graph Convolutional Networks

In this chapter, we present our quantum algorithm for the problem of node classification with Graph Convolutional Networks (GCN) [11]. We start by restating some notations: Let $G = (V, E)$ be a graph, where V is the set of nodes and E is the set of edges. $A \in \mathbb{R}^{N \times N}$ is the adjacency matrix, with N being the total number of nodes, and $X \in \mathbb{R}^{N \times C}$ is the node attribute matrix, with C being the number of features for each node. The node representations at the l-th layer is denoted as $H^{(l)} \in \mathbb{R}^{N \times F_l}, l \in \{0, 1, 2, \cdots, K\}$, where F_l being the dimension of node representation for each node. These notations are summarised in the following table.

Concept	Notation
Graph	$G = (V, E)$
Adjacency matrix	$A \in \mathbb{R}^{N \times N}$
Node attributes	$X \in \mathbb{R}^{N \times C}$
Total number of GCN layers	K
Node representations at the l-th layer	$H^{(l)} \in \mathbb{R}^{N \times F_l}, l \in \{0, 1, 2, \cdots, K\}$

7.1 Single-Channel GCN

The "GNN layer" (described in Section 6.1) in a Graph Convolutional Network, often termed as "Graph Convolution", can be carried out either in spectral domain[98] or spatial domain[11]. Ref.[111] demonstrated the equivalence of these two types of Graph Convolution. Here we present the case of spectral Graph Convolution.

We will begin by considering the situation in which each node has a single feature, i.e. $C = 1$ and $X \in \mathbb{R}^{N \times C}$ becomes a vector $\mathbf{x} \in \mathbb{R}^N$. The spectral convolutions on graphs can be defined as a multiplication of $\mathbf{x} \in \mathbb{R}^N$ with a convolutional filter $g_\theta = \text{diag}(\theta)$ ($\theta \in \mathbb{R}^N$ is the parameter of the filter) in the spectral domain[98]:

$$g_\theta \star \mathbf{x} = U g_\theta U^T \mathbf{x}.$$

in which U is the matrix of the eigenvectors of the normalized graph Laplacian matrix $L = I_N - D^{-\frac{1}{2}} A D^{-\frac{1}{2}}$ that has the eigen-decomposition $L = U \Lambda U^T$ (Λ is a diagonal matrix of its eigenvalues).

The Graph Convolution consists of three steps: 1) Graph Fourier Transform by multiplying U^Ton the input signal $\mathbf{x}$, 2) Element-wise product with the convolutional filter g_θ, 3) Inverse Graph Fourier Transform. For machine learning applications, directly learning the parameter of the filter $\theta \in \mathbb{R}^N$ is typically inappropriate for a number of reasons such as 1) The resulting convolution of a general filter is often not localised, 2) The Graph convolution of a general filter is computationally expensive, etc. A common solution is to make the filter a function of Λ, typically a degree-k polynomial function such that

$$g_{\theta'}(\Lambda) = \sum_{k=0}^{K} \theta'_k \Lambda^k,$$

in which the coefficients $\{\theta'_k\}$ are the trainable parameters. The graph convolution then becomes:

$$g_{\theta'} \star \mathbf{x} = \sum_{k=0}^{K} \theta'_k L^k \mathbf{x}. \tag{7.1}$$

A graph convolutional network comprises multiple convolution layers defined according to Eqn.7.1 with each layer followed by a nonlinear activation function. Next, we present the quantum implementation of the layer-wise transformation for Single-Channel GCN.

7.1.1 Data Encoding

We adopt a commonly used data encoding method— Amplitude encoding[112]—for our graph data: The node features $\mathbf{x} \in \mathbb{R}^N$ (after normalization) are encoded in the amplitudes of a n-qubit quantum state $|\psi_x\rangle$ as

$$|\psi_x\rangle = \sum_{i=1}^{N} x_i |i\rangle$$

where $N = 2^n, x_i$ is the i-th element of $\mathbf{x}$, and $|i\rangle$ is the i-th computational basis state. Although the focus of this thesis is not on the specific implementation of the data encoding process, numerous quantum state preparation techniques can be utilized to achieve it [113, 114, 115, 116, 117, 118, 119, 120, 121, 9, 122, 123].

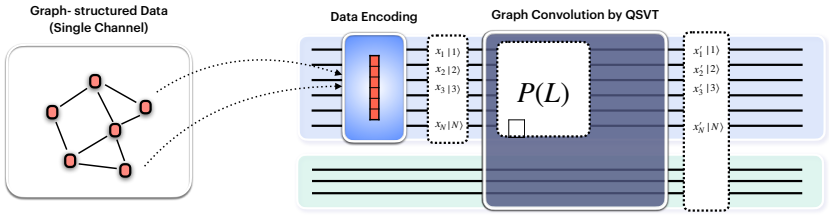

Figure 7.1: *Quantum implementation of the layer-wise linear transformation for Single-Channel GCN* The node features $\mathbf{x} \in \mathbb{R}^N$ (after normalization) are encoded in the amplitudes of a n-qubit quantum state $|\psi_x\rangle$ as $|\psi_x\rangle = \sum_{i=1}^{N} x_i|i\rangle$ via a quantum state preparation process[9]. The main operation of GCN – graph convolution – is achieved by Quantum Singular Value Transformation (QSVT): applying a block encoding of the polynomial of the Laplacian on the state in which data is encoded in.

7.1.2 Graph Convolution by QSVT

Once data has been encoded into the quantum state $|\psi_x\rangle$, the main operation of GCN – graph convolution – can be achieved by Quantum Singular Value Transformation (QSVT). We apply a block encoding of the polynomial of the Laplacian on the state in which data is encoded in, resulting in the following transformation:

$$\sum_{i=1}^{N} x_i|i\rangle \otimes |0\rangle \rightarrow \sum_{i=1}^{N} x_i'|i\rangle \otimes |0\rangle + |\text{garbage}\rangle \tag{7.2}$$

in which $|\text{garbage}\rangle$ is some garbage state, the transformed data features, denoted as $\mathbf{x}'$, satisfy

$$\mathbf{x}' = P(L)\mathbf{x} \tag{7.3}$$

where $P(L)$ is the polynomial function in graph convolution with trainable parameters

$$P(L) = \sum_{k=0}^{K} \theta_k' L^k \tag{7.4}$$

Fig. 7.1 summarises this step and the previous step (Data encoding).

As mentioned in section 2.5, the coefficients in the polynomial are determined by the Pauli angles(phases) in the QSVT circuit. Hence parametrization of the polynomial is equivalent to parametrization of the Pauli angles(phases) in the QSVT circuit, that is, the phases are the tunable weights to be trained in our Quantum implementation of Graph convolutional networks. This is illustrated in Fig. 7.2

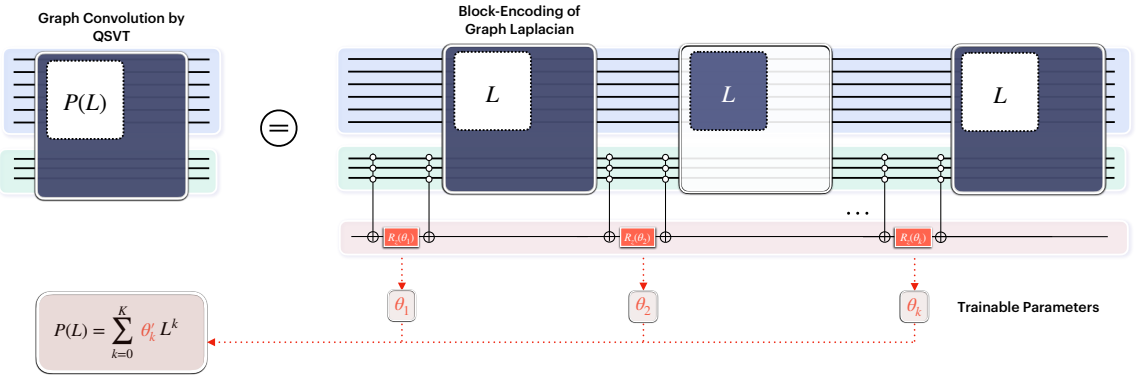

Figure 7.2: *Graph Convolution by QSVT* Graph Convolution is parametrized by the coefficients in the polynomial function of the graph Laplacian. In our quantum implementation of Graph Convolution by QSVT, the coefficients are determined by the Pauli angles(phases) in the QSVT circuit. Hence parametrization of the polynomial is equivalent to parametrization of the Pauli angles(phases) in the QSVT circuit, that is, the phases are the tunable weights to be trained in our Quantum implementation of Graph convolutional networks.

7.1.3 Nonlinear Activation Function

The step after the graph convolution is applying an element-wise non-linear activation function to the updated node features. In our Quantum GCN, this step aims to achieve:

$$\sum_{i=1}^{N} x'_i|i\rangle \rightarrow \sum_{i=1}^{N} f(x'_i)|i\rangle$$

in which $f(x)$ is a non-linear activation function, e.g. ReLU function, Sigmoid function etc.

We utilise algorithm "Nonlinear transformation of complex amplitudes(NTCA)" introduced in Ref.[10] for this step. The overall procedure of NTCA is taking a unitary that produces the initial state(to be transformed) as components, to build a new unitary that generates the desired state whose amplitudes are transformed by certain nonlinear functions. This overall routine is illustrated in Fig.7.3.

Using NTCA in our Quantum GCN to implement a non-linear activation function, we take the unitary of data encoding and graph convolution as components to build a new unitary that generates the desired state whose amplitudes are transformed by certain nonlinear functions. The full quantum circuit is depicted in Fig.7.4:

Note that the activation function in our Quantum GCN is trainable via tuning the Pauli angles(phases) in the NTCA circuit. This could become an advantage over classical GCN where having a trainable cost function can be hard/expensive to implement.

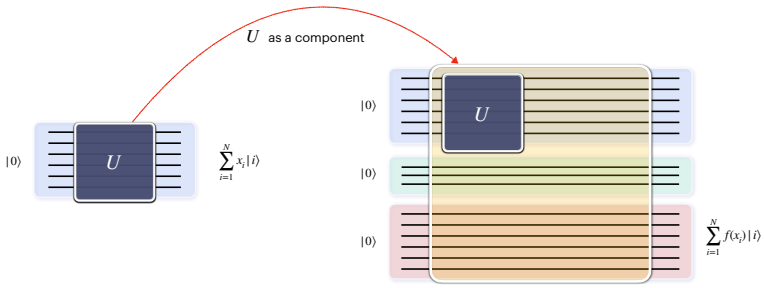

Figure 7.3: *Overall procedure of "Nonlinear transformation of complex amplitudes(NTCA)"[10]* The overall procedure of NTCA is taking a unitary that produces the initial state $\sum_{i=1}^{N} x_i|i\rangle$ as components, to build a new (larger) unitary that generate the desired state $\sum_{i=1}^{N} f(x_i)|i\rangle$ whose amplitudes are transformed by certain nonlinear function $f(x)$. Note that the transformed state $\sum_{i=1}^{N} f(x_i)|i\rangle$ does not sit on the original register where the initial state $\sum_{i=1}^{N} x_i|i\rangle$ exist on, instead it sits at the exit of some other register introduced when constructing the larger unitary(NTCA operations).

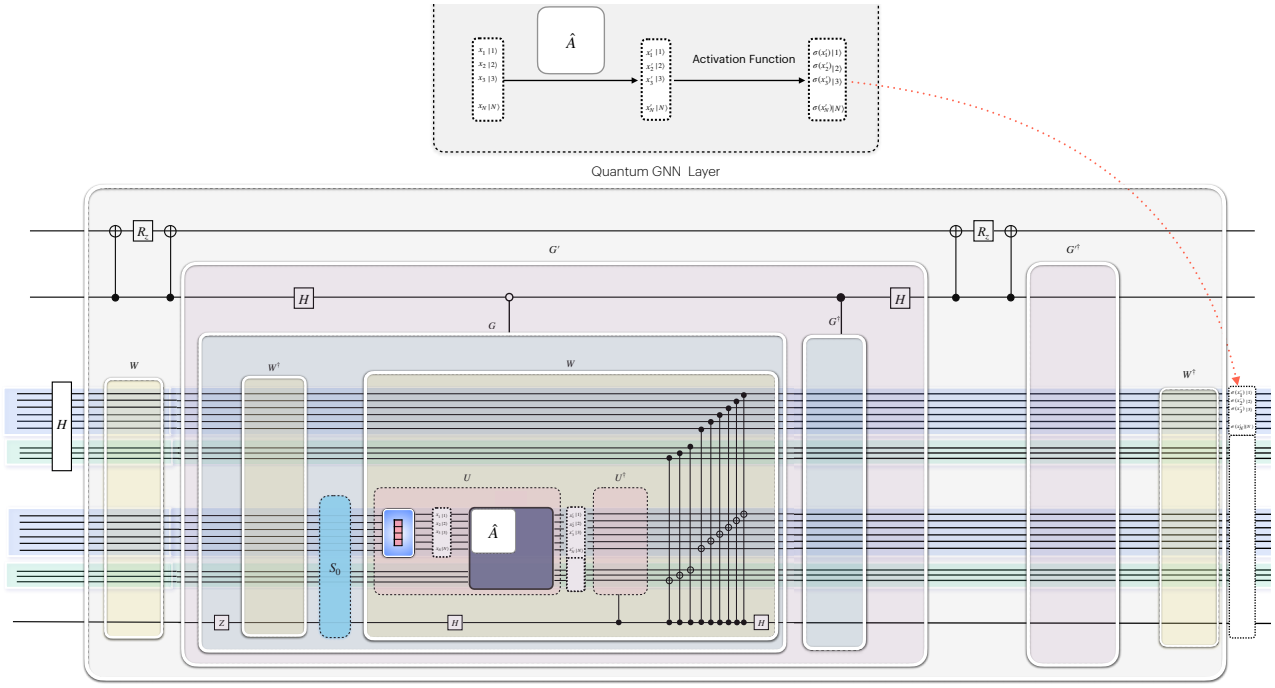

Figure 7.4: *Example of the full quantum circuit for a GNN layer ($C = 1$, single channel).* Utilising NTCA in our Quantum GCN to implement a non-linear activation function, we take the unitary of data encoding and graph convolution as components to build a new unitary that generates the desired state whose amplitudes are transformed by certain nonlinear functions. Note that the schematics in this figure are for illustration purposes only.

7.2 Multi-Channel GCN

For a multi-channel GCN, each node has multiple features such that the node features $X \in \mathbb{R}^{N \times C}$ is a matrix with C ($C > 1$) columns and the Node representations at the l-th layer $H^{(l)} \in \mathbb{R}^{N \times F_l}$

is a matrix whose F_l columns corresponds to F_l feature maps of each node. In addition to the graph convolution in the layer-wise linear transformation of a single-channel GCN, an extra layer-specific trainable weight matrix is applied to that of multi-channel GCN, yielding the layer-wise propagation rule of a multi-channel GCN (Here, we present the architecture proposed in Ref.[11] — the graph convolution is chosen to be a localized first-order approximation of spectral graph convolutions discussed in Section 7.1) :

$$H^{(l+1)} = \sigma\left(\hat{A}H^{(l)}W^{(l)}\right)$$

Here, $\hat{A} = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}$ in which $\tilde{A} = A + I_N$ is the adjacency matrix of the graph G with added self-connections(I_N is the identity matrix), $\tilde{D}_{ii} = \sum_j \tilde{A}_{ij}$. $W^{(l)}$ is a layer-specific trainable weight matrix. $\sigma(\cdot)$ denotes an activation function. The linear layer-wise transformation (excluding the nonlinear activation function) can be illustrated in Fig.7.5.

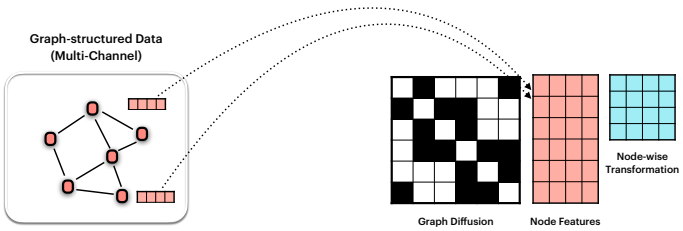

Figure 7.5: *The linear layer-wise transformation for a Multi-Channel GCN* For a multi-channel GCN, each node has multiple features such that the Node representations at the l-th layer $H^{(l)} \in \mathbb{R}^{N\times F}$ is a matrix whose F columns corresponds to F feature maps of each node. In addition to the graph convolution in the layer-wise linear transformation of a single-channel GCN, an extra layer-specific trainable weight matrix is applied to that of a multi-channel GCN. Here for brevity, we present the architecture proposed in Ref.[11] — the graph convolution is chosen to be a localized first-order approximation of spectral graph convolutions, yielding the linear layer-wise transformation for a Multi-Channel GCN to be the layer-specific trainable weight matrix $W^{(l)}$ and renormalized adjacency matrix $\hat{A}$ multiplied on the node feature matrix $H^{(l)}$. Note that the trainable weight matrix $W^{(l)}$ does not have to be a square matrix.

At the output of the last layer, softmax activation function, defined as $\text{softmax}(x_i) = \frac{1}{\mathcal{Z}}\exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$, is applied row-wise to the node feature matrix, producing the final output of the network:

$$Z = \text{softmax}(\hat{A}H^{(K-1)}W^{(l)}) \tag{7.5}$$

For semi-supervised multi-class classification, the cost function is defined by the cross-entropy error over all labelled examples [11]:

$$L = -\sum_{s \in Y_L} \sum_{f=1}^{F_K} Y_{sf} \ln Z_{sf}, \tag{7.6}$$

where Y_L is the set of node indices that have labels, $Y \in \mathbb{B}^{N \times F_K}$ denotes the one-hot encoding of the labels. The GCN pipeline mentioned above is summarised in Fig.7.6.

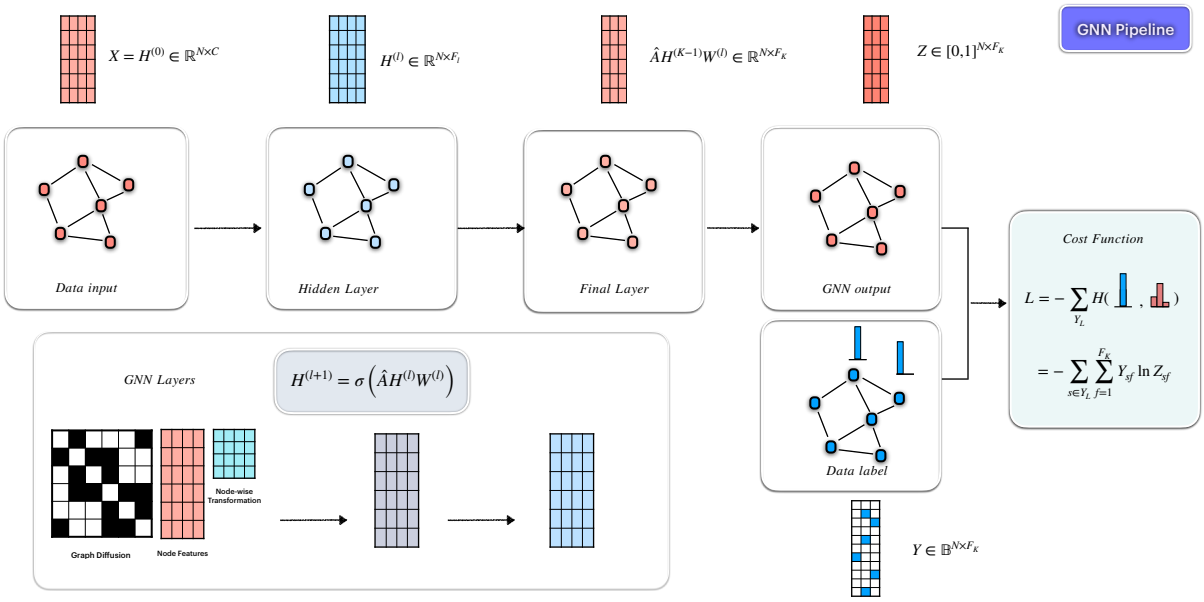

Figure 7.6: *GCN Pipeline.* A GCN consists of a series of layers in which graph convolution and non-linear activation functions are applied to the node features. (Note that the schematics in this figure are for illustration purposes only, e.g. the normalized adjacency matrix depicted here does not include the added self-connections) At the output of the last layer, softmax activation function, defined as $\text{softmax}(x_i) = \frac{1}{\mathcal{Z}} \exp(x_i)$ with $\mathcal{Z} = \sum_i \exp(x_i)$, is applied row-wise to the node feature matrix, producing the final output of the network: $Z = \text{softmax}(\hat{A} H^{(K-1)} W^{(K-1)})$. For semi-supervised multi-class classification, the cost function is defined by the cross-entropy error over all labelled examples [11]:$L = -\sum_{s \in Y_L} \sum_{f=1}^{F_K} Y_{sf} \ln Z_{sf}$, where Y_L is the set of node indices that have labels, $Y \in \mathbb{B}^{N \times F_K}$ denotes the one-hot encoding of the labels.

Next, we present the Quantum implementation of the Multi-Channel GCN.

7.2.1 Data Encoding

For multi-channel GCN, the node features $X \in \mathbb{R}^{N \times C}$ of which the entries are denoted as $X_{ik} = x_i^{(k)}$(after normalization) can be encoded in a quantum state $|\psi_X\rangle$ [1] as follows:

$$|\psi_X\rangle = \sum_{i=1}^{N} |i\rangle |\mathbf{x}_i\rangle \tag{7.7}$$

[1]Note throughout this thesis we often omit the normalization factors in quantum states.

where $|\mathbf{x}_i\rangle = \sum_{k=1}^{C} x_i^{(k)} |k\rangle$, being the amplitude encoding of the features for node i over the channels(indexed by k), is entangled with an "address" state $|i\rangle$. The entire state is prepared on two quantum registers hosting the channel index k and node index i which are denoted as $Reg(k)$ and $Reg(i)$ respectively. The data encoding, represented as the blue box in Fig.7.7, can be achieved by the "Controlled Quantum State Preparation" process [9].

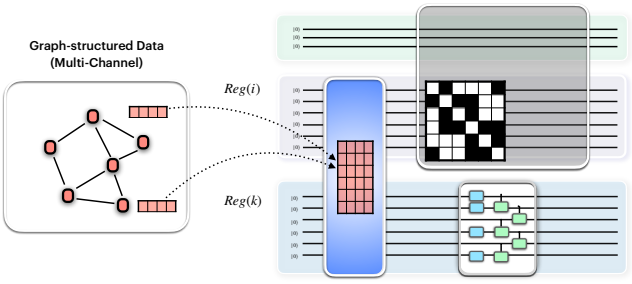

Figure 7.7: *Quantum implementation of linear layer-wise transformation for multi-channel GCN* The linear layer-wise transformation for multi-channel GCN (i.e. the layer-specific trainable weight matrix and the normalized adjacency matrix multiplied on the node feature matrix), can be implemented by applying the block-encoding of the normalized adjacency matrix and a parametrized quantum circuit on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively. Here we depicted the first layer of GCN — the linear layer-wise transformation is applied on the state prepared by the data encoding procedure (the blue box) described in Section 7.2.1. Note that the schematics in this figure are for illustration purposes only, e.g. 1) the normalized adjacency matrix depicted here does not include the added self-connections; 2) the ancillary qubits used in the quantum state preparation for the data encoding is not depicted in this figure.

7.2.2 Layer-wise transformation and Cost function

The layer-wise linear transformation for multi-channel GCN (i.e. $H'^{(l)} = \hat{A}H^{(l)}W^{(l)}$ [2]: the layer-specific trainable weight matrix and renormalized adjacency matrix multiplied on the node feature matrix), can be implemented by applying the block-encoding of the renormalized adjacency matrix and a parameterized quantum circuit on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively, as depicted in Fig.7.7. This can be proven as follows:

From $H'^{(l)} = \hat{A}H^{(l)}W^{(l)}$ we have

$$H'^{(l)^T} = W^{(l)^T} H^{(l)^T} \hat{A}^T \tag{7.8}$$

Using $vec(ABC) = (C^T \otimes A)vec(B)$ (A, B,C are matrices), we have

[2]Here we use $H'^{(l)}$ to denote the transformed feature matrix.

$$vec({H'^{(l)}}^T) = vec(W^{(l)^T} H^{(l)^T} \hat{A}^T) = (\hat{A} \otimes W^{(l)^T}) vec(H^{(l)^T}) \tag{7.9}$$

For an arbitrary matrix M, define vectors $\boldsymbol{\psi}_M = vec(M)$, and Eqn.7.9 becomes

$$\boldsymbol{\psi}_{H'^{(l)^T}} = (\hat{A} \otimes W^{(l)^T}) \boldsymbol{\psi}_{H^{(l)^T}} \tag{7.10}$$

Similar to Eqn.7.7, we can define the quantum state on the two quantum registers $Reg(i)$ and $Reg(k)$ for $H^{(l)}$ as

$$\left|\psi_{H^{(l)}}\right\rangle = \sum_{i=1}^{N} |i\rangle \otimes \left|\mathbf{x}_i^{(l)}\right\rangle \tag{7.11}$$

and for $H'^{(l)}$:

$$\left|\psi_{H'^{(l)}}\right\rangle = \sum_{i=1}^{N} |i\rangle \otimes \left|\mathbf{x}'^{(l)}_i\right\rangle \tag{7.12}$$

Writing the quantum states in Eqn.7.11 and 7.12 as vectors we note that

$$\boldsymbol{\psi}_{H^{(l)^T}} = \left|\psi_{H^{(l)}}\right\rangle \tag{7.13}$$

$$\boldsymbol{\psi}_{H'^{(l)^T}} = \left|\psi_{H'^{(l)}}\right\rangle \tag{7.14}$$

Substituting Eqs.7.11 - 7.14 into Eq. 7.10 we find

$$\left|\psi_{H'^{(l)}}\right\rangle = (\hat{A} \otimes W^{(l)^T}) \left|\psi_{H^{(l)}}\right\rangle \tag{7.15}$$

in which $(\hat{A} \otimes W^{(l)^T})$ corresponds to applying the block-encoding of $\hat{A}$ and a parameterized quantum circuit implementing $W^{(l)^T}$ on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively. That is, $H'^{(l)} = \hat{A} H^{(l)} W^{(l)}$ — the layer-specific trainable weight matrix and renormalized adjacency matrix multiplied on the node feature matrix can be implemented by applying the block-encoding of the renormalized adjacency matrix and a parameterized quantum circuit on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively. The proof can be summarised in Fig.7.8.

After the linear layer-wise transformation, the element-wise non-linear activation function can be applied in a similar way as described in section7.1.3. In practice, GCNs often involve two or three layers[97], therefore we only need to apply the non-linear activation function two or three times. This enables our quantum GCN that utilize NTCA (for the non-linear activation function) to maintain certain advantages—in the case of more layers, NTCA's complexity-theoretic benefit will vanish[10]. [3]

[3]However there are proposals of classical GCN architectures (e.g.[124]) in which the non-linear activation functions are ommited.

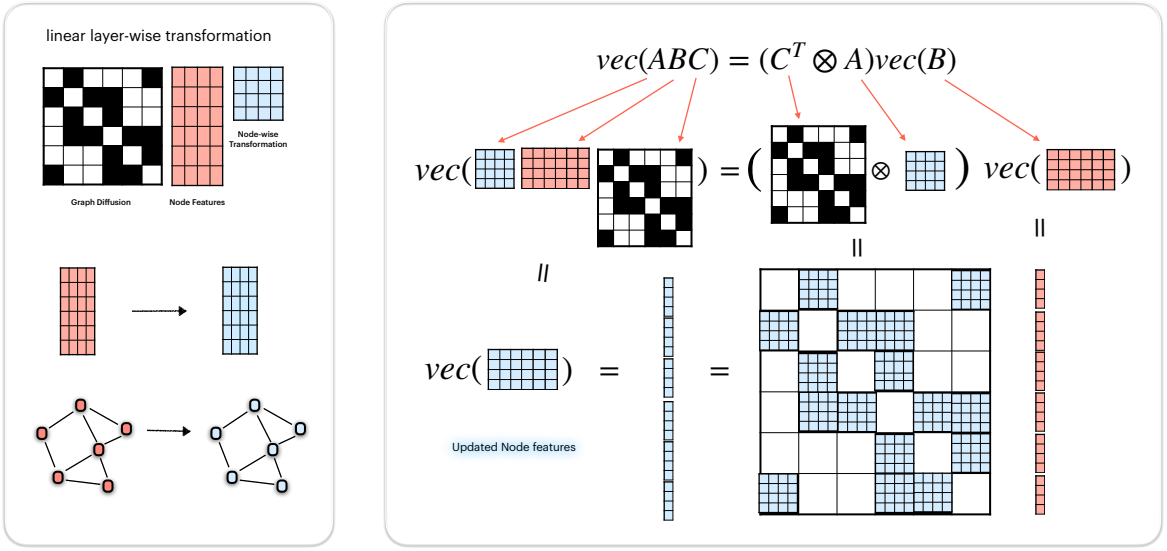

Figure 7.8: *Proof of our Quantum implementation of linear layer-wise transformation for multi-channel GCN* The linear layer-wise transformation for multi-channel GCN (i.e. the layer-specific trainable weight matrix and adjacency matrix multiplied on the node feature matrix), can be implemented by applying the block-encoding of the renormalized adjacency matrix and a parametrized quantum circuit on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively. The figure summarises the proof from Eqn.7.8 to 7.15.

For semi-supervised multi-class classification, the cost function used in our framework is defined as the negative inner product between the outcome state of our quantum GCN and a target label state $|\psi_Y\rangle := \sum_{s\in Y_L}\sum_{f=1}^{C} Y_{sf}\,|s\rangle\,|f\rangle \otimes |0\rangle$. The cost function can be evaluated via the "Modified Hadamard test" [125, 126].

Chapter 8

Quantum Graph Attention Network

Attention mechanisms [16] have become effectively a standard component in a variety of tasks in natural language processing and computer vision. Veličković et al [127] introduced an attention-based architecture, Graph Attention Network, to perform node classification of graph-structured data. As mentioned in Section 6.1, the building block layer of Graph Attention Network achieves the following transformations which we refer as "Graph attention operation":

$$\mathbf{h}_j = \phi\left(\mathbf{x}_j, \bigoplus_{i\in\mathcal{N}_j} a\left(\mathbf{x}_i, \mathbf{x}_j\right)\psi\left(\mathbf{x}_i\right)\right), \tag{8.1}$$

where $a\left(\mathbf{x}_j, \mathbf{x}_i\right)$ is a scalar that indicates the relationship strength between node i and j, often referred as attention coefficients or attention scores [16]. The following sections present our quantum implementation of Eqn.8.1. In Section 8.1 we design a Quantum Attention Mechanism to evaluate and store attention score $a(\mathbf{x}_i, \mathbf{x}_j)$ on quantum circuit, which serves as a crucial component for the subsequent construction described in Section 8.2.2.

8.1 Quantum Attention Mechanism

The quantum attention mechanism aims to coherently evaluate and store attention score $a(\mathbf{x}_i, \mathbf{x}_j)$ for each pair of the nodes, which can be defined as a quantum oracle $O_{\mathsf{attention}}$ such that:

$$O_{\mathsf{attention}} \left|i\right\rangle \left|j\right\rangle \left|0\right\rangle \rightarrow \left|i\right\rangle \left|j\right\rangle \left|a(\mathbf{x}_i, \mathbf{x}_j)\right\rangle \tag{8.2}$$

In this section we present the construction of the quantum attention oracle consisting of the following two steps:

8.1.1 Evaluating Attention score in superposition

The Attention score $a(\mathbf{x}_i, \mathbf{x}_j)$ in our Quantum Attention Mechanism can take one of the standard forms in classical literature [16] — the inner product of the linearly transformed feature vectors of each pair of nodes

$$a(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T W_K^T W_Q \mathbf{x}_j, \tag{8.3}$$

in which W_K, W_Q are trainable linear transformations. In terms of Dirac notation, this can be written as:

$$a(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i | U_K^\dagger U_Q | \mathbf{x}_j \rangle \tag{8.4}$$

in which U_K, U_Q are trainable unitaries.

In our Quantum Attention Mechanism, the attention score can be evaluated on quantum circuit by parallel Swap Test (mentioned in Chapter 4) as depicted in Fig. 8.1 which we will discuss in detail below.

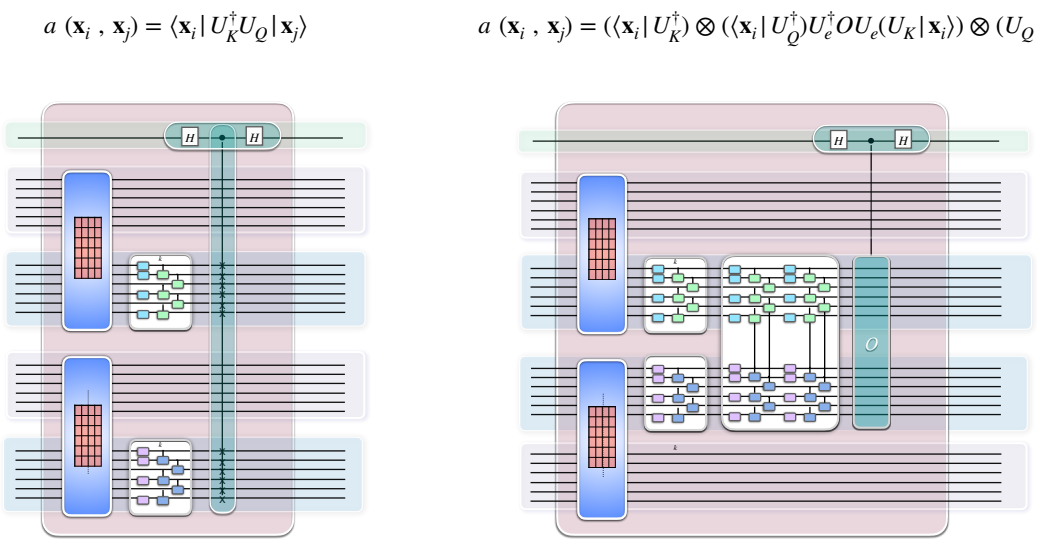

Figure 8.1: *Quantum Attention Mechanism* The Attention score $a(\mathbf{x}_i, \mathbf{x}_j)$ in our Quantum Attention Mechanism can take the form of the inner product of the linearly transformed feature vectors of each pair of nodes $a(\mathbf{x}_i, \mathbf{x}_j) = \mathbf{x}_i^T W_K^T W_Q \mathbf{x}_j$, in which W_K, W_Q are trainable linear transformations. In terms of Dirac notation, this can be written as: $a(\mathbf{x}_i, \mathbf{x}_j) = \langle \mathbf{x}_i | U_K^\dagger U_Q | \mathbf{x}_j \rangle$, in which U_K, U_Q are trainable unitaries. In our Quantum Attention Mechanism, this attention score can be evaluated in superposition on quantum circuit by parallel Swap Test(mentioned in Chapter 4), depicted as the left side of this figure. On the left side of this figure, we illustrate an alternative form of the Attention score, which can be evaluated by parallel Hadamard Test (mentioned in Chapter 4)

We denote the unitary for the parallel swap test circuit, as circled by the pink box on the left panel of Fig. 8.1, as U. The input to U, $|\Psi_0\rangle$, can be written as (note here and throughout

the thesis, we omit the normalization factor):

$$|\Psi_0\rangle = |0\rangle \otimes (\sum_i |i\rangle) \otimes |0\rangle_{\mathtt{K}}^n \otimes (\sum_j |j\rangle) \otimes |0\rangle_{\mathtt{Q}}^n, \tag{8.5}$$

where $|0\rangle_{\mathtt{K}}^n, |0\rangle_{\mathtt{Q}}^n$ are the initial states of two copies of data registers on which the node features $a(\mathbf{x}_i, \mathbf{x}_j)$ will be loaded. The data encoding via Controlled Quantum State Preparation[9], depicted as the blue boxes in Fig.8.1, can be written as $\sum_i |i\rangle\langle i| \otimes U_{\mathbf{x}_i}$ where $U_{\mathbf{x}_i}|0\rangle = |\mathbf{x}_i\rangle$.

Applying this data encoding on the two copies of data registers yields the overall state:

$$|\Psi_1\rangle = |0\rangle \otimes (\sum_i |i\rangle \otimes |\mathbf{x}_i\rangle^n) \otimes (\sum_j |j\rangle \otimes |\mathbf{x}_j\rangle^n) \tag{8.6}$$

Node-wise linear transformation U_K, U_Q(trainable unitaries) implemented by PQC are then applied to the node feature registers, yielding the following state:

$$|\Psi_2\rangle = |0\rangle \otimes (\sum_i |i\rangle \otimes U_K|\mathbf{x}_i\rangle^n) \otimes (\sum_j |j\rangle \otimes U_Q|\mathbf{x}_j\rangle^n) \tag{8.7}$$

We further define $\mathcal{K}_i, \mathcal{Q}_j$ and corresponding state $|k_i\rangle, |q_j\rangle$ as $\mathcal{K}_i|0\rangle_{\mathtt{K}}^n = U_K|\mathbf{x}_i\rangle = |k_i\rangle, \mathcal{Q}_j|0\rangle_{\mathtt{Q}}^n = U_Q|\mathbf{x}_j\rangle = |q_j\rangle$. Then U can be written explicitly as

$$\begin{aligned} U \coloneqq\ & [H \otimes I \otimes I \otimes I \otimes I] \cdot \\ & [|0\rangle\langle 0| \otimes (\sum_i \sum_j |i\rangle\langle i| \otimes \mathcal{K}_i \otimes |j\rangle\langle j| \otimes \mathcal{Q}_j) + |1\rangle\langle 1| \otimes (\sum_i \sum_j |i\rangle\langle i| \otimes \mathcal{Q}_j \otimes |j\rangle\langle j| \otimes \mathcal{K}_i)] \\ & \cdot [H \otimes I \otimes I \otimes I \otimes I], \end{aligned} \tag{8.8}$$

which can be rewritten as

$$U = \sum_i \sum_j |i\rangle\langle i| \otimes |j\rangle\langle j| \otimes U_{ij}, \tag{8.9}$$

where

$$U_{ij} \coloneqq [H \otimes I \otimes I] \cdot [|0\rangle\langle 0| \otimes \mathcal{K}_i \otimes \mathcal{Q}_j + |1\rangle\langle 1| \otimes \mathcal{Q}_j \otimes \mathcal{K}_i\ ] \cdot [H \otimes I \otimes I], \tag{8.10}$$

Define $|\phi_{ij}\rangle \coloneqq U_{ij}|0\rangle|0\rangle_{\mathtt{K}}^n|0\rangle_{\mathtt{Q}}^n$ and we have:

$$|\phi_{ij}\rangle = \frac{1}{\sqrt{2}}(|+\rangle|k_i\rangle|q_j\rangle + |-\rangle|q_j\rangle|k_i\rangle). \tag{8.11}$$

Expanding and rearranging the terms in Eq. 8.11 we find

$$|\phi_{ij}\rangle = \frac{1}{2}(|0\rangle \otimes (|k_i\rangle|q_j\rangle + |q_j\rangle|k_i\rangle) + |1\rangle \otimes (|k_i\rangle|q_j\rangle - |q_j\rangle|k_i\rangle). \tag{8.12}$$

Denote $|u_{ij}\rangle$ and $|v_{ij}\rangle$ as the normalized states of $|k_i\rangle|q_j\rangle + |q_j\rangle|k_i\rangle$ and $|k_i\rangle|q_j\rangle - |q_j\rangle|k_i\rangle$ respectively. Then there is a real number $\theta_{ij} \in [\pi/4, \pi/2]$ such that

$$|\phi_{ij}\rangle = \sin\theta_{ij}|0\rangle|u_{ij}\rangle + \cos\theta_{ij}|1\rangle|v_{ij}\rangle. \tag{8.13}$$

θ_{ij} satisfies $\cos\theta_{ij} = \sqrt{1 - |\langle k_i|q_j\rangle|^2}\,/\sqrt{2}$, $\sin\theta_{ij} = \sqrt{1 + |\langle k_i|q_j\rangle|^2}\,/\sqrt{2}$.

The final output state from U, $|\Psi_3\rangle = U|\Psi_0\rangle$, can then be written as

$$|\Psi_3\rangle = \sum_i \sum_j |i\rangle|j\rangle \underbrace{(\sin\theta_{ij}\,|u_{ij}\rangle\,|0\rangle + \cos\theta_{ij}\,|v_{ij}\rangle\,|1\rangle)}_{|\phi_{ij}\rangle} = \sum_i \sum_j |i\rangle|j\rangle\,|\phi_{ij}\rangle \tag{8.14}$$

Note that $\langle k_i|q_j\rangle = \langle \mathbf{x}_i|\,U_K^\dagger U_Q\,|\mathbf{x}_j\rangle = a(\mathbf{x}_i, \mathbf{x}_j)$ being the attention scores are encoded in the amplitudes of the output state $|\Psi_3\rangle$ of swap test as:

$$|\langle k_i|q_j\rangle|^2 = -\cos 2\theta_{ij}. \tag{8.15}$$

8.1.2 Storing Attention score

The second step is to use amplitude estimation [90] to extract and store the attention scores into an additional register which we call the "amplitude register".

After step 1, we introduce an extra register $|0\rangle^t_{\mathit{amplitude}}$ and the output state $|\Psi_3\rangle$ (using the same notation) becomes

$$|\Psi_3\rangle = \sum_i \sum_j |i\rangle|j\rangle\,|\phi_{ij}\rangle\,|0\rangle^t_{\mathit{amplitude}}, \tag{8.16}$$

where $|\phi_{ij}\rangle$ can be decomposed as

$$|\phi_{ij}\rangle = \frac{-i}{\sqrt{2}}\left(e^{i\theta_{ij}}\,|\omega_+\rangle_{ij} - e^{i(-\theta_{ij})}\,|\omega_-\rangle_{ij}\right). \tag{8.17}$$

Hence, we have

$$|\Psi_3\rangle = \sum_i \sum_j \frac{-i}{\sqrt{2}}\left(e^{i\theta_{ij}}|i\rangle\,|j\rangle\,|\omega_+\rangle_{ij} - e^{i(-\theta_{ij})}|i\rangle\,|j\rangle\,|\omega_-\rangle_{ij}\right)|0\rangle^t_{\mathit{amplitude}}. \tag{8.18}$$

The overall Grover operator G is defined as

$$G := UC_2U^\dagger C_1, \tag{8.19}$$

where C_1 is the Z gate on the swap ancilla qubit, and $C_2 = I - 2|0\rangle\langle 0|$ is the "flip zero state" on registers other than the two registers hosting indices i, j (represented as S_0 in Fig.8.2). It can be shown that G can be expressed as

$$G = \sum_i \sum_j |i\rangle\,|j\rangle\langle j|\,\langle i| \otimes G_{ij}, \tag{8.20}$$

where G_{ij} is defined as

$$G_{ij} = (I - 2|\phi_{ij}\rangle\langle\phi_{ij}|))C_1 \tag{8.21}$$

It is easy to check that $|w_\pm\rangle_{ij}$ are the eigenstates of G_{ij}, that is,

$$G_{ij}|w_\pm\rangle_{ij} = e^{\pm i 2\theta_{ij}}|w_\pm\rangle_{ij}. \tag{8.22}$$

The overall Grover operator G possess the following eigen-relation:

$$G\,|i\rangle\,|j\rangle\,|\omega_\pm\rangle_{ij} = e^{i(\pm 2\theta_{ij})}\,|i\rangle\,|j\rangle\,|\omega_\pm\rangle_{ij}\,. \tag{8.23}$$

Next, we apply phase estimation of the overall Grover operator G on the input state $|\Psi_3\rangle$. The resulting state $|\Psi_4\rangle$ can be written as

$$|\Psi_4\rangle = \sum_i \sum_j \frac{-i}{\sqrt{2}} \left(e^{i\theta_{ij}}\,|i\rangle\,|j\rangle\,|\omega_+\rangle_{ij}\,|2\theta_{ij}\rangle - e^{i(-\theta_{ij})}\,|i\rangle\,|j\rangle\,|\omega_-\rangle_{ij}\,|-2\theta_{ij}\rangle\right). \tag{8.24}$$

Note here in Eq. 8.24, $|\pm 2\theta_{ij}\rangle$ denotes the eigenvalues $\pm 2\theta_{ij}$ being stored in the amplitude register with some finite precision.

Next, we apply an oracle U_O on the amplitude register and an extra ancilla register, which acts as

$$U_O\,|0\rangle\,|\pm 2\theta_{ij}\rangle = |a(\mathbf{x}_i, \mathbf{x}_j)\rangle\,|\pm 2\theta_{ij}\rangle\,, \tag{8.25}$$

The state after the oracle can be written as

$$|\Psi_5\rangle = \sum_i \sum_j \frac{-i}{\sqrt{2}}\,|a(\mathbf{x}_i, \mathbf{x}_j)\rangle \left(e^{i\theta_{ij}}\,|i\rangle\,|j\rangle\,|\omega_+\rangle_{ij}\,|2\theta_{ij}\rangle - e^{i(-\theta_{ij})}\,|i\rangle\,|j\rangle\,|\omega_-\rangle_{ij}\,|-2\theta_{ij}\rangle\right). \tag{8.26}$$

Then we perform the uncomputation of Phase estimation, the resulting state is

$$|\Psi_6\rangle = \sum_i \sum_j \frac{-i}{\sqrt{2}}\,|a(\mathbf{x}_i, \mathbf{x}_j)\rangle \left(e^{i\theta_{ij}}\,|i\rangle\,|j\rangle\,|\omega_+\rangle_{ij}\,|0\rangle^t_{\mathit{amplitude}} - e^{i(-\theta_{ij})}\,|i\rangle\,|j\rangle\,|\omega_-\rangle_{ij}\,|0\rangle^t_{\mathit{amplitude}}\right) \tag{8.27}$$

$$= \sum_i \sum_j |a(\mathbf{x}_i, \mathbf{x}_j)\rangle\,|i\rangle|j\rangle\,|\phi_{ij}\rangle\,|0\rangle^t_{\mathit{amplitude}} \tag{8.28}$$

Finally, we perform the uncomputation of the swap test and the resulting state is

$$|\Psi_7\rangle = \sum_i \sum_j |a(\mathbf{x}_i, \mathbf{x}_j)\rangle\,|i\rangle|j\rangle\,|0\rangle\,|0\rangle^t_{\mathit{amplitude}}\,. \tag{8.29}$$

The above steps, as illustrated in Fig. 8.2, implemented the quantum attention oracle $O_{\mathsf{attention}}$ such that:

$$O_{\mathsf{attention}}\,|i\rangle\,|j\rangle\,|0\rangle \rightarrow |i\rangle\,|j\rangle\,|a(\mathbf{x}_i, \mathbf{x}_j)\rangle \tag{8.30}$$

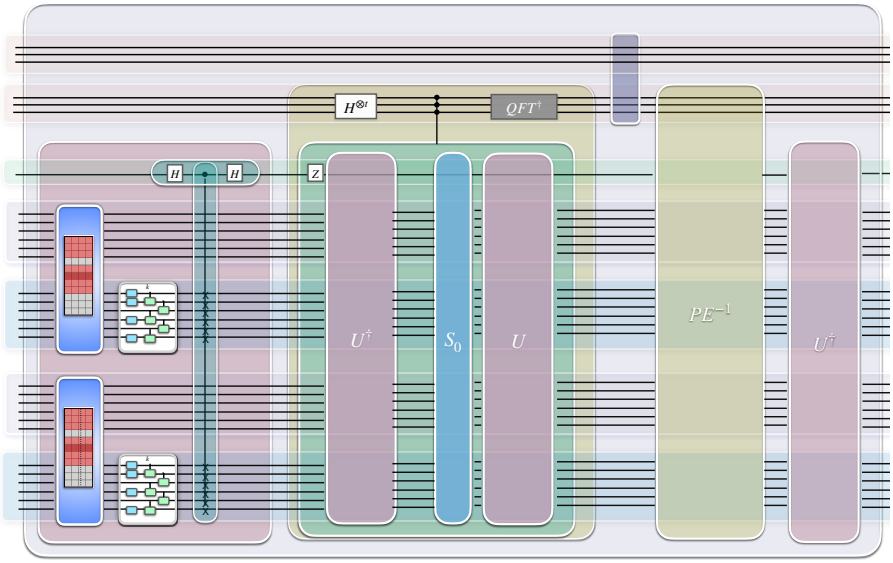

Figure 8.2: *Quantum attention oracle* $O_{\mathsf{attention}}$ The quantum attention mechanism aims to coherently evaluate and store attention score $a(\mathbf{x}_i, \mathbf{x}_j)$ for each pair of the nodes, which can be defined as a quantum oracle $O_{\mathsf{attention}}$ such that $O_{\mathsf{attention}} |i\rangle |j\rangle |0\rangle \to |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle$. The construction of the quantum attention oracle, depicted in this figure, is detailed in section 8.1.1 and 8.1.2.

8.2 Quantum algorithm for Graph Attention

The Graph Attention operation defined in Eq.8.1 can also be described similar to the layer-wise linear transformation for multi-channel GCN in Section 7.2.2, i.e. $H'^{(l)} = \hat{A} H^{(l)} W^{(l)}$: the layer-specific trainable weight matrix and renormalized adjacency matrix multiplied on the node feature matrix. Here in the Graph Attention operation, the non-zero elements in the renormalized adjacency matrix $\hat{A}$ are modified to be the attention score of the corresponding node pairs [17]. Therefore on quantum circuit, similar to the case of multi-channel GCN, the Graph Attention operation can be implemented by applying the block-encoding of the modified renormalized adjacency matrix(which we refer as "weighted adjacency matrix" and a parameterized quantum circuit. In the following Section 8.2.2 we present how to achieve the block-encoding of the weighted adjacency matrix. As a preliminary, the block-encoding of certain sparse[1] matrix is illustrated in Section 8.2.1.

8.2.1 Block encoding of certain sparse matrices

The block encoding of a general sparse matrix[65] requires a certain oracle that it's hard to construct in our case for the Graph Attention operation. In this thesis, we investigate the sparse matrices that can be decomposed as the summation of 1-sparse matrices (A "1-sparse matrix" is defined as there is only one nonzero entry in each row or column of the matrix). We start with

[1]For many practical applications, the adjacency matrix of a Graph is often sparse.

the block encoding of 1-sparse matrices.

For each column j of a 1-sparse matrix A, there is a single row index $c(j)$ such that $A_{c(j),j} \neq 0$, and the mapping c is a permutation.[66] Therefore, A can be expressed as the product of a diagonal matrix (whose diagonal entries are the non-zero entries of the 1-sparse matrix) and a permutation matrix. Ref.[66] showed that the block encoding of a 1-sparse matrix can be constructed by multiplying the block encoding of a diagonal matrix and the block encoding of a permutation matrix: the permutation matrix, denoted as O_c act as

$$O_c|j\rangle = |c(j)\rangle.$$

and the block encoding of the diagonal matrix, denoted as O_A, act as:

$$O_A|0\rangle|j\rangle = \left(A_{c(j),j}|0\rangle + \sqrt{1 - \left|A_{c(j),j}\right|^2}\,|1\rangle\right)|j\rangle.$$

$U_A = (I \otimes O_c)\, O_A$ is a block encoding of the 1-sparse matrix A [66].

Now we consider the case for the sparse matrices that can be decomposed as the summation of 1-sparse matrices, below we also use A to denote such matrix(for reason will be clear soon). After the decomposition, we index the 1-sparse matrices by l. For the l-th 1-sparse matrix, the row index of the nonzero entry in each column j, is denoted by $c(j,l)$. There exist O_c^l and O_A^l and corresponding U_A^l such that [66]

$$O_c^l|j\rangle = |c(j,l)\rangle. \tag{8.31}$$

$$O_A^l|0\rangle|j\rangle = \left(A_{c(j,l),j}|0\rangle + \sqrt{1 - \left|A_{c(j,l),j}\right|^2}\,|1\rangle\right)|j\rangle \tag{8.32}$$

It can be shown that $\sum_l U_A^l = \sum_l \left(I \otimes O_c^l\right) O_A^l$ is a block encoding of the sparse matrix A [66]. The summation over l can be carried out by Linear Combination of Unitaries(LCU)[8].

For the construction of O_A^l, assume that there is an oracle [66]

$$\widetilde{O}_A^l \left|0^{d'}\right\rangle |j\rangle = \left|\widetilde{A}_{c(j,l),j}\right\rangle |j\rangle,$$

where $\widetilde{A}_{c(j,l),j}$ is a d'-bit representation of $A_{c(j,l),j}$. By arithmetic operations, we can convert this oracle into another oracle

$$O_A^{l\prime} \left|0^{d}\right\rangle |j\rangle = \left|\widetilde{\theta}_{c(j,l),j}\right\rangle |j\rangle,$$

where $0 \leqslant \tilde{\theta}_{c(j,l),j} < 1$, and $\tilde{\theta}_{c(j,l),j}$ is a d-bit representation of $\theta_{c(j,l),j} = \arccos\left(A_{c(j,l),j}\right)/\pi$.

Next, using the "Controlled rotation given rotation angles" (Proposition 4.7 in Ref.[66], denoted as "CR" below) and uncomputation of ${O_A^l}'$ we can achieve the construction of O_A^l:

$$|0\rangle \underbrace{\left|0^d\right\rangle}_{\text{work register}} |j\rangle \xrightarrow{{O_A^l}'} |0\rangle \left|\tilde{\theta}_{c(j,l),j}\right\rangle |j\rangle \tag{8.33}$$

$$\xrightarrow{\text{CR}} \left(A_{c(j,l),j}|0\rangle + \sqrt{1-\left|A_{c(j,l),j}\right|^2}\,|1\rangle\right)\left|\tilde{\theta}_{c(j,l),j}\right\rangle |j\rangle \tag{8.34}$$

$$\xrightarrow{({O_A^l}')^{-1}} \left(A_{c(j,l),j}|0\rangle + \sqrt{1-\left|A_{c(j,l),j}\right|^2}\,|1\rangle\right)\left|0^d\right\rangle |j\rangle \tag{8.35}$$

8.2.2 Quantum Graph Attention operation

As mentioned in section 8.2.1, in this thesis we investigate certain sparse matrices(here in Graph attention operation, the weighted adjacency matrices) that can be decomposed as the summation of 1-sparse matrices. From the preliminary discussion in section 8.2.1, the block encoding of such matrices can be boiled down to the $\tilde{O}_A^l$ for each 1-sparse matrix. That is, the core task is to construct the following operation for each 1-sparse matrix(indexed by l):

$$O_l^{\mathsf{diagonal}} : |j\rangle |0\rangle \to |j\rangle \left|A_{c(j,l),j}\right\rangle, \tag{8.36}$$

where $\left|A_{c(j,l),j}\right\rangle$ denotes $A_{c(j,l),j}$ being stored in a quantum register with some finite precision, and for simplicity we use $|0\rangle$ to represent a state of the register that all qubits in the register being in the state of $|0\rangle$. We also adopt this kind of notion in the rest of the thesis: for a scalar a, $|a\rangle$ denotes a being stored in a quantum register with some finite precision, and in contexts where there is no ambiguity, $|0\rangle$ represent a state of a quantum register that all qubits in the register being in the state of $|0\rangle$.

In our case of Graph attention operation, the elements of the weighted adjacency matrix are the attention scores, i.e. $A_{i,j} = a(\mathbf{x}_i, \mathbf{x}_j)$, and we have

$$O_l^{\mathsf{diagonal}} : |j\rangle |0\rangle \to |j\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle. \tag{8.37}$$

Note that in section 8.1 we have already construct a quantum oracle $O_{\mathsf{attention}}$ such that:

$$O_{\mathsf{attention}} : |i\rangle |j\rangle |0\rangle \to |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle \tag{8.38}$$

In the following of this section, we present how to construct an alternative version[2] of O_l^{diagonal} utilising $O_{\mathsf{attention}}$.

[2]Note that we are not strictly constructing O_l^{diagonal} here and the following operations do not strictly achieve a block-encoding of the weighted adjacency matrix, however the alternative versions do generate a quantum state that resembles the Graph attention operation.

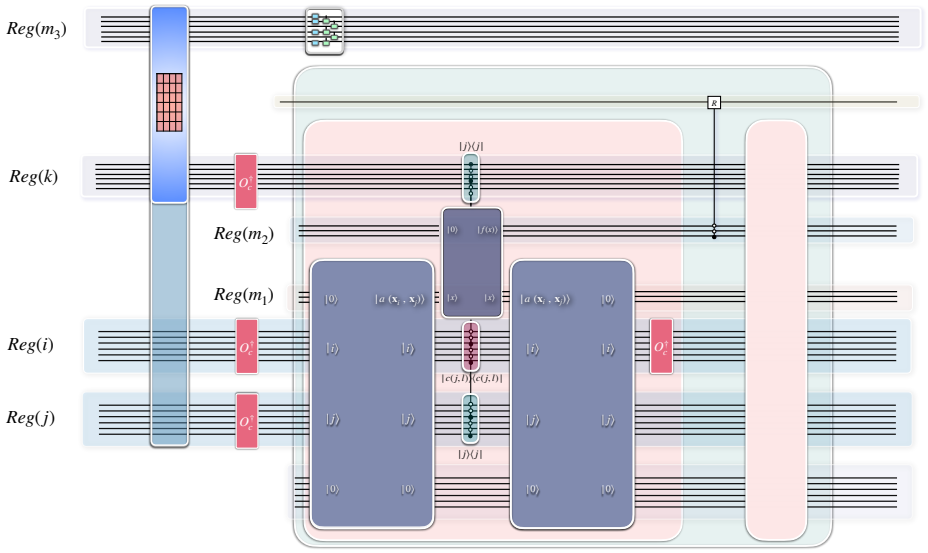

Figure 8.3: *Quantum implementation of linear layer-wise transformation for Graph Attention Networks.* The initial data state $\left|\psi_{X0}^{3}\right\rangle = \sum_i |i\rangle^{\otimes 3} |\mathbf{x}_i\rangle$ is prepared by the blue box on the left. The QNN module, denoted as U_w, transform the state to $\left|\psi_{X}^{3}\right\rangle = \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle$. The pale green box together with the three red boxes which achieve $M_l' = \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0| + ...$, are then applied to the transformed initial data state, resulting $\sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle |0\rangle$. The pale green box consists of the following Modules: Module 1(the first pink box). $\mathcal{O}_l^{\text{diagonal}}$ Module 2. the "Conditional Rotation" (Theorem 3.5 in Ref. [12]) Module 3 (the second pink box) is the uncomputation of Module 1.

Step 1: Attention oracle loading the attention scores $A_{i,j} = a(\mathbf{x}_i, \mathbf{x}_j)$

The first component is the attention oracle $O_{\text{attention}}$, depicted as the navy box in Fig.8.3. When being applied onto the three address register $Reg(i)$, $Reg(j)$ and the corresponding memory register $Reg(m_1)$, $O_{\text{attention}}$ loads the attention scores $A_{i,j} = a(\mathbf{x}_i, \mathbf{x}_j)$ for each pair of the nodes i, j into $Reg(m_1)$, while the other memory register $Reg(m_2)$ stays $|0\rangle$. $O_{\text{attention}}$ act as:

$$O_{\text{attention}} : |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle . \tag{8.39}$$

If $O_{\text{attention}}$ is applied onto an input state as $\sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle$, it transform the state as:

$$\sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow \sum_i \sum_j \sum_k |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle . \tag{8.40}$$

Step 2: Selective copying of the attention scores $A_{i,j} = a(\mathbf{x}_i, \mathbf{x}_j)$

The second component is a multi-controlled unitary which performs the "selective copying" of the attention scores onto $Reg(m_2)$, depicted as the blue-navy-red-blue combo boxes following

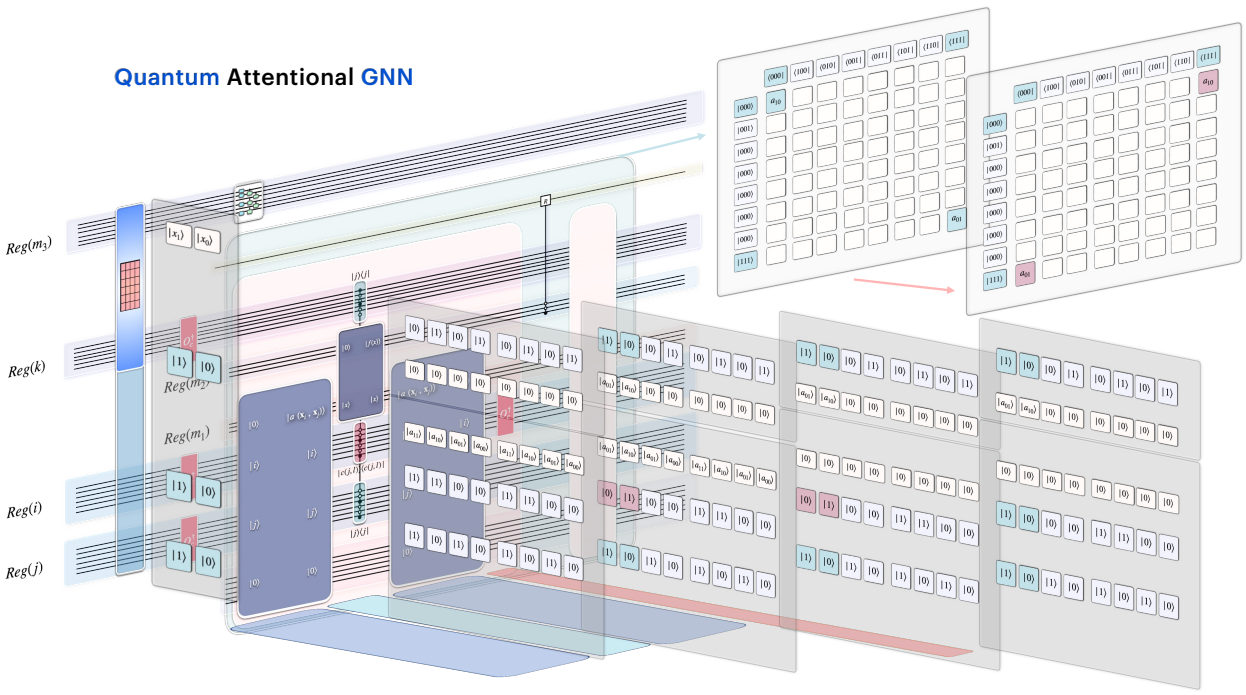

Figure 8.4: *Quantum implementation of linear layer-wise transformation for Graph Attention Networks.* This figure provides a small example of the corresponding states and matrices in Fig. 8.3. The panels perpendicular to the circuit plane represent the quantum states, while the panels parallel to the circuit plane represent the corresponding matrices.

the attention oracle in Fig.8.3. The copying is implemented by a quantum oracle that acts as $|0\rangle|x\rangle \to |f(x)\rangle|x\rangle$ where f can be a nonlinear activation function, however, we still name the operation as "copying."

Consider the branches indexed by i,j,k in the state $\sum_i\sum_j\sum_k |i\rangle|j\rangle|a(\mathbf{x}_i,\mathbf{x}_j)\rangle|0\rangle|k\rangle$, the copying is defined[3] to happen only for the branches $i=c(j,l); k=j$, that is, the "selective copying" operation transform the branches in the state $\sum_i\sum_j\sum_k |i\rangle|j\rangle|a(\mathbf{x}_i,\mathbf{x}_j)\rangle|0\rangle|k\rangle$ as follows:

For branches $i=c(j,l); k=j$:

$$\sum_j |c(j,l)\rangle|j\rangle\left|a(\mathbf{x}_{c(j,l)},\mathbf{x}_j)\right\rangle|0\rangle|j\rangle \to \sum_j |c(j,l)\rangle|j\rangle\left|a(\mathbf{x}_{c(j,l)},\mathbf{x}_j)\right\rangle\left|a(\mathbf{x}_{c(j,l)},\mathbf{x}_j)\right\rangle|j\rangle\,. \tag{8.41}$$

For other branches :

$$\sum_{i\neq c(j,l)}\sum_j\sum_{k\neq j}|i\rangle|j\rangle|a(\mathbf{x}_i,\mathbf{x}_j)\rangle|0\rangle|k\rangle \to \sum_{i\neq c(j,l)}\sum_j\sum_{k\neq j}|i\rangle|j\rangle|a(\mathbf{x}_i,\mathbf{x}_j)\rangle|0\rangle|k\rangle\,, \tag{8.42}$$

combining all branches we have the selective copying of the attention scores $A_{i,j}=a(\mathbf{x}_i,\mathbf{x}_j)$

[3]For an implementation of the "selective copying" operation, see AppendixA.1

as:

$$\sum_j |c(j,l)\rangle |j\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |0\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |c(j,l)\rangle |j\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle .$$

Step 3: Uncomputation of attention oracle $O^{\dagger}_{\mathsf{attention}}$

The third component is the uncomputation of attention oracle $O_{\mathsf{attention}}$ which act as

$$O^{\dagger}_{\mathsf{attention}} : |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j) \rightarrow |i\rangle |j\rangle |0\rangle\rangle . \tag{8.43}$$

When acting on the output state of Step 2, it transforms the state as follows:

$$\sum_j |c(j,l)\rangle |j\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |c(j,l)\rangle |j\rangle |0\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle .$$

Step 4: Permutation of basis on register $Reg(i)$

The fourth component is the permutation of basis in the register $Reg(i)$ by applying the unitary $O_c^{l\dagger}$ (defined in Eq.8.31) which act as

$$O_c^{l\dagger} |c(j,l)\rangle = |j\rangle .$$

When acting on the output state of Step 3, it transforms the state as follows:

$$\sum_j |c(j,l)\rangle |j\rangle |0\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |j\rangle |j\rangle |0\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |P(i)\rangle |j\rangle |0\rangle |0\rangle |k\rangle ,$$

where $|P(i)\rangle := O_c^{l\dagger} |i\rangle$.

The state evolution during the four steps can be summarized as follows:

$$\sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle = \sum_j |c(j,l)\rangle |j\rangle |0\rangle |0\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |c(j,l)\rangle |j\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |0\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |c(j,l)\rangle |j\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |a(\mathbf{x}_i, \mathbf{x}_j)\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |c(j,l)\rangle |j\rangle |0\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow$$
$$\sum_j |j\rangle |j\rangle |0\rangle \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |P(i)\rangle |j\rangle |0\rangle |0\rangle |k\rangle .$$

Gathering all four steps above, the pink box in Fig.8.3 implements an alternative version of O_l^{diagonal} denoted as $\mathcal{O}_l^{\text{diagonal}}$ which act as:

$$\mathcal{O}_l^{\text{diagonal}} : |j\rangle^{\otimes 3} |0\rangle \to |j\rangle^{\otimes 3} \left|a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle, \tag{8.44}$$

in which we neglected some registers that are unchanged before/after. Note Eq.8.44 does not specify the transformation of $\mathcal{O}_l^{\text{diagonal}}$ acting on states other than $|j\rangle^{\otimes 3} |0\rangle$.

In terms of the elements of the weighted adjacency matrices, $\mathcal{O}_l^{\text{diagonal}}$ which act as:

$$\mathcal{O}_l^{\text{diagonal}} |j\rangle^{\otimes 3} |0\rangle \to |j\rangle^{\otimes 3} \left|A_{c(j,l),j}\right\rangle \tag{8.45}$$

Armed with $\mathcal{O}_l^{\text{diagonal}}$, we can then construct the Graph attention operation using the recipe discussed in the previous section 8.2.1, which is based on the following modules.

Module 1. $\mathcal{O}_l^{\text{diagonal}}$

Module 2. the "Conditional Rotation" (Theorem 3.5 in Ref. [13]), to convert $A_{c(j,l),j}$ from basis to amplitude.

Module 3. Uncomputation of module 1

These three modules achieve the following unitary:

$$M_l = \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle j|^{\otimes 3} \langle 0| + ... \quad . \tag{8.46}$$

Module 4. Permutation of basis

Three $O_c^{l\dagger}$ which act as $\langle j| O_c^{l\dagger} = \langle c(j,l)|$ are applied before the previous three modules on the addresses, yield

$$M_l' = \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0| + ... \quad . \tag{8.47}$$

When M_l' is applied on the transformed data state $\left|\psi_X^3\right\rangle := \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle$, prepared by the blue box and the QNN module (represented by U_w) in Fig.8.1, it act as follows

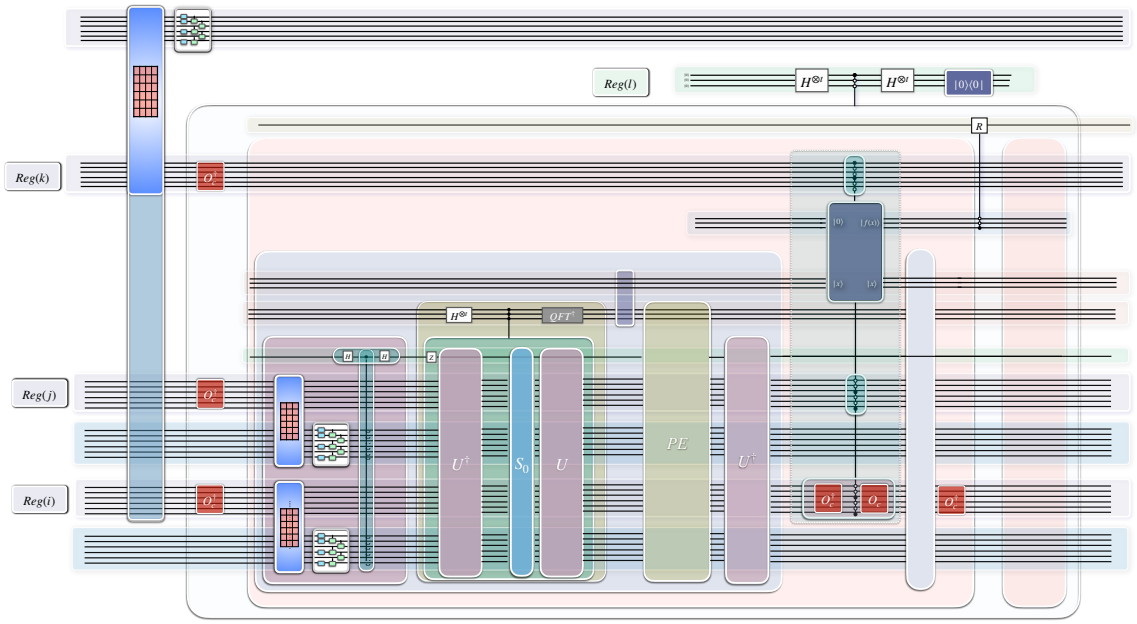

Figure 8.5: *Quantum implementation of linear layer-wise transformation for Graph Attention Networks.* The initial data state $\left|\psi^3_{X0}\right\rangle = \sum_i |i\rangle^{\otimes 3} |\mathbf{x}_i\rangle$ is prepared by the blue box on the left. The QNN module, denoted as U_w, transforms the state to $\left|\psi^3_X\right\rangle = \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle$. The transparent box which achieves $M'_l = \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0| + ...$, consist of four Modules: Module 1(the first pink box) $\mathcal{O}_l^{\text{diagonal}}$. Module 2 the Conditional Rotation (Theorem 3.5 in Ref. [13]), represented as the controlled-R gate between the two pink boxes. Module 3 (the second pink box) Uncomputation of Module 1. Module 4(the three red boxes on the left of module 1) Permutation of basis. An overall LCU is then applied to the four modules, depicted in as the add-on register $Reg(l)$ controlling the transparent box, to achieve the addition over index l: $M = \sum_l M'_l = \sum_l \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0| + ...$ M is then applied on $\left|\psi^3_X\right\rangle = \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle$, producing the outcome state $\sum_j |j\rangle^{\otimes 3} \sum_l A_{c(j,l),j} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle |0\rangle$.

$$M'_l \left|\psi^3_X\right\rangle = (\sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0| + ...) \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle |0\rangle \tag{8.48}$$

$$= (\sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0|) \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle |0\rangle \tag{8.49}$$

$$= \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle |0\rangle . \tag{8.50}$$

The operations constructed so far can be summarised in Fig.8.3, Fig.8.4 provide a small example of the corresponding states and matrices.

To achieve the addition over index l, an overall LCU is applied to the four modules, depicted in Fig.8.5 and 8.6 as the add-on register $Reg(l)$ with the controlled unitaries in the transparent box, achieving the following operation:

$$M := \sum_l M'_l = \sum_l \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} |0\rangle \langle c(j,l)|^{\otimes 3} \langle 0| + ... \tag{8.51}$$

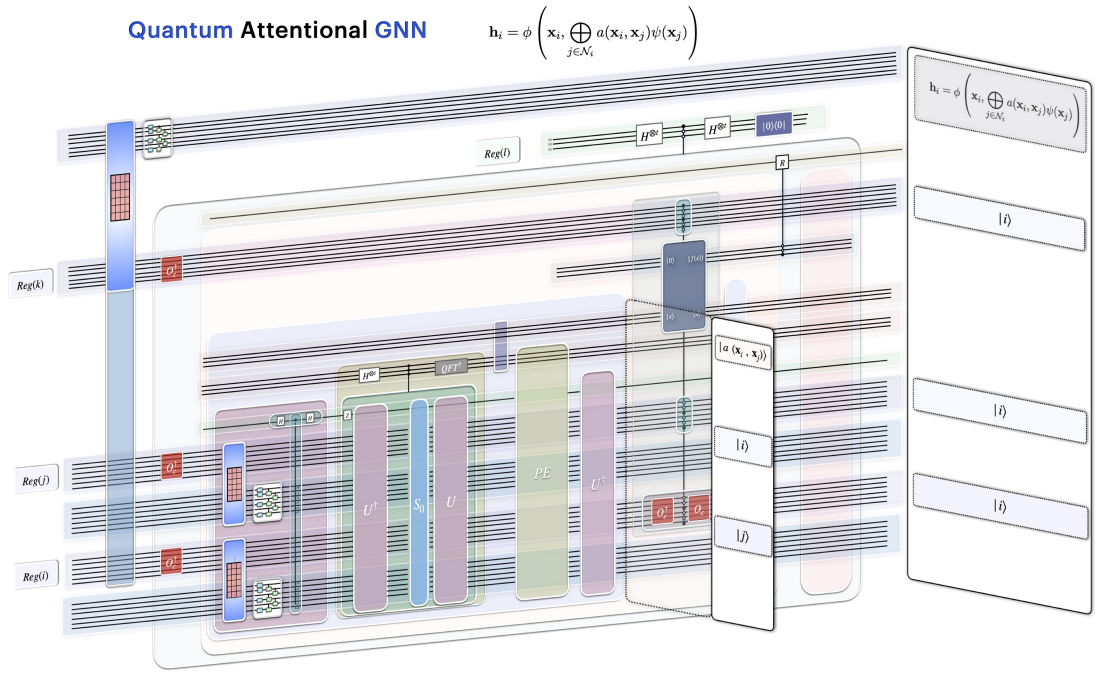

Figure 8.6: *Quantum implementation of linear layer-wise transformation for Graph Attention Networks.* This figure provides a 3D state-circuit view for Fig. 8.5. The panels perpendicular to the circuit plane represent the quantum states generated by corresponding circuits.

When M is applied on $\left|\psi_X^3\right\rangle = \sum_i |i\rangle^{\otimes 3} U_w |\mathbf{x}_i\rangle$, it produces the outcome state as:

$$M\left|\psi_X^3\right\rangle = \sum_l M_l' \left|\psi_X^3\right\rangle = \sum_l \sum_j A_{c(j,l),j} |j\rangle^{\otimes 3} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle |0\rangle \tag{8.52}$$

$$= \sum_j |j\rangle^{\otimes 3} \sum_l A_{c(j,l),j} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle |0\rangle \tag{8.53}$$

We can add an extra Identity unitary I with coefficient r in the LCU that produces M, yielding

$$M'\left|\psi_X^3\right\rangle = (M+rI)\left|\psi_X^3\right\rangle = \sum_j |j\rangle^{\otimes 3} \sum_l A_{c(j,l),j} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle |0\rangle + r\sum_j |j\rangle^{\otimes 3} U_w |\mathbf{x}_j\rangle |0\rangle \tag{8.54}$$

$$= \sum_j |j\rangle^{\otimes 3} \left(\sum_l A_{c(j,l),j} U_w \left|\mathbf{x}_{c(j,l)}\right\rangle + rU_w |\mathbf{x}_j\rangle\right) |0\rangle \tag{8.55}$$

$$= \sum_j |j\rangle^{\otimes 3} \left(\sum_l a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j) U_w \left|\mathbf{x}_{c(j,l)}\right\rangle + rU_w |\mathbf{x}_j\rangle\right) |0\rangle \tag{8.56}$$

$$= \sum_j |j\rangle^{\otimes 3} \left|\mathbf{x}_j'\right\rangle |0\rangle \tag{8.57}$$

where $\left|\mathbf{x}_j'\right\rangle := rU_w |\mathbf{x}_j\rangle + \sum_l a(\mathbf{x}_{c(j,l)}, \mathbf{x}_j) U_w \left|\mathbf{x}_{c(j,l)}\right\rangle$ is the updated node feature in accordance with Eqn. 8.1, by identifying $U_w |\mathbf{x}_i\rangle$ is the amplitude encoding of $\psi(\mathbf{x}_i)$, setting $\phi(\mathbf{x},\mathbf{z}) = \mathbf{W}\mathbf{x}+\mathbf{z}$, and interpreting $c(j,l)$ as the node index for the l-th neighbourhood of a node indexed by j in the graph.

In summary, by the circuit construction described so far, we obtain the following state that resembles the Graph attention operation:

$$\sum_j |j\rangle^{\otimes 3} |\mathbf{h}_j\rangle |0\rangle = \sum_j |j\rangle^{\otimes 3} \left| \phi \left(\mathbf{x}_j, \bigoplus_{i \in \mathcal{N}_j} a(\mathbf{x}_i, \mathbf{x}_j) \psi(\mathbf{x}_i) \right) \right\rangle |0\rangle . \tag{8.58}$$

Chapter 9

Quantum Message-Passing GNN

Similar to the case of Graph Attention Networks in Chapter 8, our Quantum Message-Passing GNN aims to evaluate and store the updated node features

$$\mathbf{h}_j = \phi\left(\mathbf{x}_j, \bigoplus_{i\in\mathcal{N}_j} \psi\left(\mathbf{x}_j, \mathbf{x}_i\right)\right), \tag{9.1}$$

into a quantum state as $\sum_j |j\rangle^{\otimes 3} |\mathbf{h}_j\rangle + ...$, that is, to obtain the following state

$$\sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \bigoplus_{i\in\mathcal{N}_j} \psi(\mathbf{x}_i, \mathbf{x}_j))\right\rangle + ... \quad . \tag{9.2}$$

This can be achieved via the following steps, as illustrated in Fig. 9.1 and 9.2.

Step 1: Data Loading of linearly transformed node features $\mathbf{x}_k$

The first step is to apply the data loading module described in Section 7.2.1 on the address register $Reg(k)$ and the corresponding memory register $Reg(m_1)$ on which a parameterized quantum circuit module is then applied to linearly transform the node features. This step loads the linearly transformed node features $\mathbf{x}_k$ of each node into the memory register associated with address $|k\rangle$. Together with the other two address registers $Reg(i)$, $Reg(j)$ and corresponding memory registers $Reg(m_2)$, $Reg(m_3)$(which will be described in the following steps), the overall state transforms as:

$$\sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow \sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |\mathbf{x}_k\rangle |k\rangle . \tag{9.3}$$

Step 2: Selective LCU

The second step aims to implement updating each node's feature $\mathbf{x}_i$ from the vectors $\psi(\mathbf{x}_i, \mathbf{x}_j)$, as in Eq. 9.1.

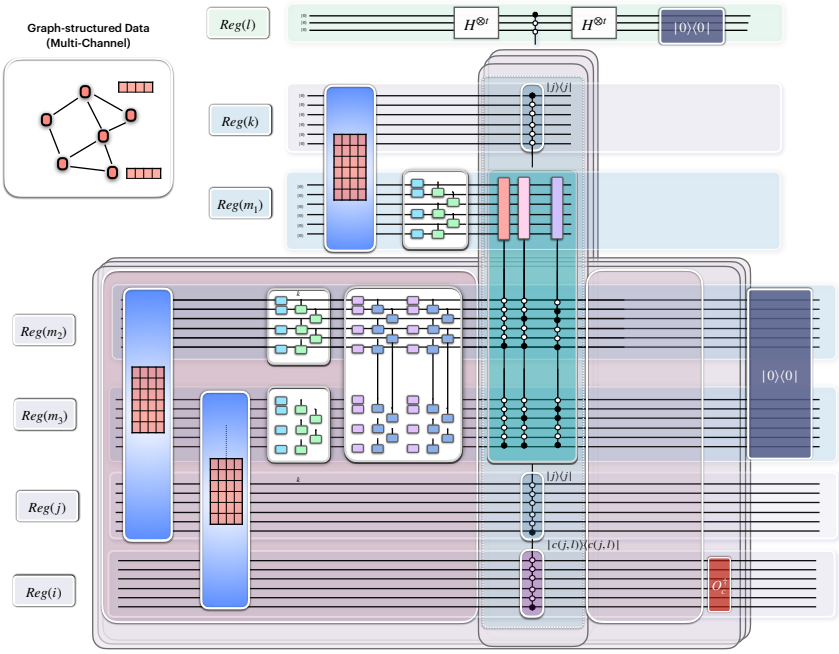

Figure 9.1: *Quantum Algorithm for Message-Passing GNN.* Our Quantum Message-Passing GNN aims to evaluate and store the updated node features $\mathbf{h}_j = \phi\left(\mathbf{x}_j, \bigoplus_{i\in\mathcal{N}_j} \psi\left(\mathbf{x}_j, \mathbf{x}_i\right)\right)$into a quantum state as $\sum_j |j\rangle^{\otimes 3} |\mathbf{h}_j\rangle + ...$ This can be achieved via the following steps: Step 1: Data Loading of linearly transformed node features $\mathbf{x}_k$; Step 2: Selective LCU; Step 3: Permutation of basis; Gathering all steps above, the Quantum Message-Passing GNN loads and transforms the node features as: $\sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow \sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\right\rangle + ...$ Step 4: Overall LCU, we then apply the overall LCU module (depicted as the top add-on register $Reg(l)$ with the controlled unitaries in fainted blue box), to achieve the aggregation over different neighbours, obtaining the following state:$\sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \sum_l \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\right\rangle + ...$, which can also be written as $\sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \bigoplus_{v\in\mathcal{N}_j} \psi(\mathbf{x}_v, \mathbf{x}_j))\right\rangle + ...$

Similar to the case of Graph Attention Networks (mentioned in Section 8.2.1), in this section, we investigate the graphs with certain adjacency matrices that can be decomposed as the summation of 1-sparse matrices. After the decomposition, we index the 1-sparse matrices by l. For the l-th 1-sparse matrix, the row index of the nonzero entry in each column j, is denoted by $c(j,l)$. Interpreting $c(j,l)$ as the node index for the l-th neighbourhood of a node indexed by j in the graph, aggregation over different neighbours can be formulated as summing over l, that is,

$$\phi(\mathbf{x}_j, \bigoplus_{v\in\mathcal{N}_j} \psi(\mathbf{x}_v, \mathbf{x}_j)) := \phi(\mathbf{x}_j, \sum_l \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)). \tag{9.4}$$

Since ϕ is linear in its arguments, we have,

$$\phi(\mathbf{x}_j, \sum_l \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)) = \sum_l \phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)). \tag{9.5}$$

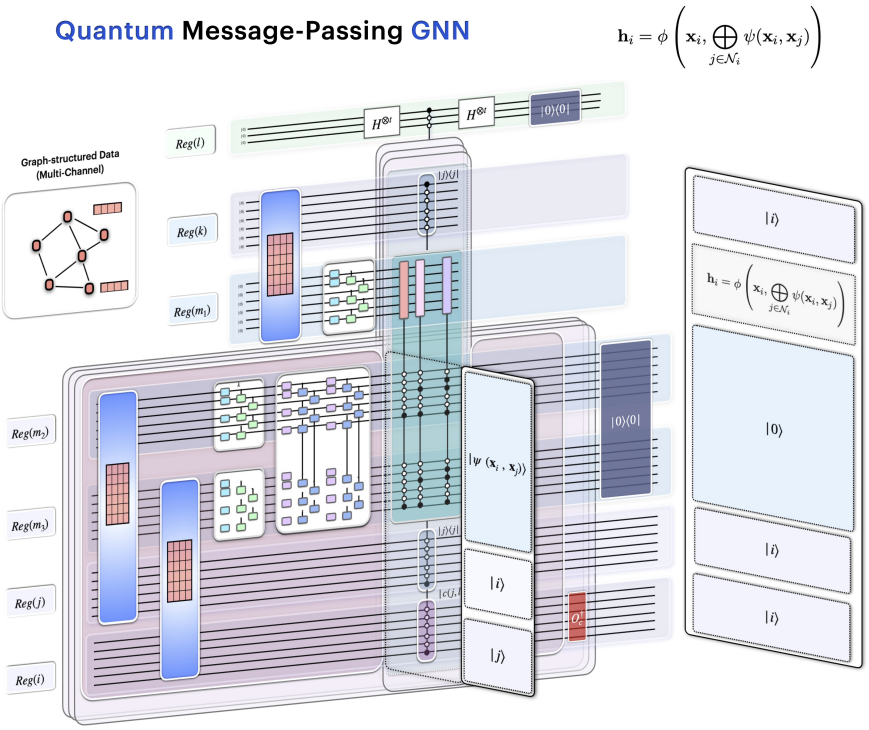

Figure 9.2: *Quantum Algorithm for Message-Passing GNN* This figure provide a 3D state-circuit view for Fig.9.1. The panels perpendicular to the circuit plane represent the quantum states generated by corresponding circuits.

This allows us to achieve the aggregation over different neighbours by the overall LCU module depicted in Fig. 9.1 and 9.2 as the top add-on register $Reg(l)$ with the controlled unitaries in fainted blue box implementing $\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))$ for each l. For a node in the graph, we then first focus on the message-passing from one neighbour of the node represented as $\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))$.

For each neighbour of a node, a "selective LCU" is performed to implement the node updating function $\phi(\mathbf{x}_i, \psi(\mathbf{x}_i, \mathbf{x}_j))$. This is achieved by applying the following modules:

Module 1: A data loading+linear transformation module that evaluates the vector $\psi(\mathbf{x}_i, \mathbf{x}_j)$, depicted in Fig. 9.1 and 9.2 as the pink box. This module comprises two data loading modules on address registers Reg(i), $Reg(j)$ and their corresponding data registers $Reg(m_2)$, $Reg(m_3)$, followed by two parametrized quantum circuits on $Reg(m_2)$, $Reg(m_3)$ respectively and an overall parametrized quantum circuits on $Reg(m_2)$, $Reg(m_3)$.

This module acts as follows:

$$\sum_i \sum_j |i\rangle |j\rangle |0\rangle \rightarrow \sum_i \sum_j |i\rangle |j\rangle |\psi(\mathbf{x}_i, \mathbf{x}_j)\rangle \tag{9.6}$$

Module 2: Selectively controlled unitaries on the three data registers, as gathered in the fainted blue box in Fig. 9.1 and 9.2.

we can write out $|\psi(\mathbf{x}_i, \mathbf{x}_j)\rangle$ as:

$$|\psi(\mathbf{x}_i, \mathbf{x}_j)\rangle = \sum_p w_p^{ij} |p\rangle \tag{9.7}$$

and the controlled unitaries, depicted in Fig. 9.1 and 9.2 as the multi-controlled red/purple boxes, can be written as

$$U_{\mathsf{multi}} = \sum_p |p\rangle \langle p| \otimes U_p \tag{9.8}$$

where U_p are some constant or trainable unitaries.

and the selective controlled unitaries are defined[1] as:

$$U_{\mathsf{Selective}} := \sum_j |j\rangle \langle j| \otimes |j\rangle \langle j| \otimes |c(j,l)\rangle \langle c(j,l)| \otimes U_{\mathsf{multi}}. \tag{9.9}$$

Module 3: Uncomputation of Module 1.

Module 4: Projection onto zero state on $Reg(m_2)$, $Reg(m_3)$.

For each specific combination of i, j, k, the above modules achieve LCU on $Reg(m_1)$ and act as,

$$|\mathbf{x}_k\rangle \to \sum_p |w_p^{ij}|^2 U_p |\mathbf{x}_k\rangle . \tag{9.10}$$

Considering Eq. 9.7 and the definitions of functions ϕ and ψ, We denote the transformed state in Eq. 9.10 as

$$|\phi(\mathbf{x}_k, \psi(\mathbf{x}_i, \mathbf{x}_j))\rangle := \sum_p |w_p^{ij}|^2 U_p |\mathbf{x}_k\rangle . \tag{9.11}$$

Consider the branches indexed by i, j, k in the overall state, according to the action of the selectively controlled unitaries defined in Module 2, the selective LCU only happens for the branches $i = c(j,l); k = j$.

For branches $i = c(j,l); k = j$:

$$\sum_j |c(j,l)\rangle |j\rangle |0\rangle |\mathbf{x}_j\rangle |j\rangle \to \sum_j |c(j,l)\rangle |j\rangle \left|\psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j)\right\rangle |\mathbf{x}_j\rangle |j\rangle \to \sum_j |c(j,l)\rangle |j\rangle |0\rangle \left|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\right\rangle |j\rangle \tag{9.12}$$

in which the node features transform as:

[1]The implementation of the "Selective controlled unitaries" can be achieved in the same way as the implementation of the "selective copying" operation described in Section 8.2.2.

$$|\mathbf{x}_j\rangle \to \Big|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\Big\rangle. \tag{9.13}$$

That is, the node features $|\mathbf{x}_j\rangle$ are updated by the "message" $\psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))$ from one of its neighbours indexed by l.

For other branches:

$$\sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle \to \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|\psi(\mathbf{x}_i, \mathbf{x}_j)\rangle\,|\mathbf{x}_k\rangle\,|k\rangle \to \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle\,. \tag{9.14}$$

All branches combined together:

$$\begin{aligned} &\sum_j |c(j,l)\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_j\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle \to \\ &\sum_j |c(j,l)\rangle\,|j\rangle\,|0\rangle\,\Big|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\Big\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle\,. \end{aligned}$$

Step 3: Permutation of basis

We then apply a permutation of basis on register $Reg(i)$ via applying the unitary $O_c^{l\dagger}$ (defined in Eq.8.31) as,

$$O_c^{l\dagger}\,|c(j,l)\rangle = |j\rangle.$$

when acting on the output state of Step 2, it transforms the state as follows:

$$\begin{aligned} &\sum_j |c(j,l)\rangle\,|j\rangle\,|0\rangle\,\Big|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\Big\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle \to \\ &\sum_j |j\rangle\,|j\rangle\,|0\rangle\,\Big|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\Big\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |P(i)\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle\,. \end{aligned}$$

where $|P(i)\rangle := O_c^{l\dagger}\,|i\rangle$.

The state evolution during the above steps can be summarized as follows:

$$\begin{aligned} &\sum_i \sum_j \sum_k |i\rangle\,|j\rangle\,|0\rangle\,|0\rangle\,|k\rangle \to \sum_i \sum_j \sum_k |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle = \\ &\sum_j |c(j,l)\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_j\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle \to \\ &\sum_j |c(j,l)\rangle\,|j\rangle\,|0\rangle\,\Big|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\Big\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |i\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle \to \\ &\sum_j |j\rangle\,|j\rangle\,|0\rangle\,\Big|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\Big\rangle\,|j\rangle + \sum_{i\neq c(j,l)} \sum_j \sum_{k\neq j} |P(i)\rangle\,|j\rangle\,|0\rangle\,|\mathbf{x}_k\rangle\,|k\rangle\,. \end{aligned}$$

Gathering all the steps above, the Quantum message passing GNN load and transforms the node features as:

$$\sum_i \sum_j \sum_k |i\rangle |j\rangle |0\rangle |0\rangle |k\rangle \rightarrow \sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\right\rangle + ... \tag{9.15}$$

where we have neglected some registers that are unchanged.

Step 4: Overall LCU

We then apply the aforementioned overall LCU module (depicted in Fig. 9.1 and 9.2 as the top add-on register $Reg(l)$ with the controlled unitaries in the faint blue box), to achieve the aggregation over different neighbours, obtaining the following state:

$$\sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \sum_l \psi(\mathbf{x}_{c(j,l)}, \mathbf{x}_j))\right\rangle + ... \tag{9.16}$$

which can also be written as,

$$\sum_j |j\rangle^{\otimes 3} \left|\phi(\mathbf{x}_j, \bigoplus_{v \in \mathcal{N}_j} \psi(\mathbf{x}_v, \mathbf{x}_j))\right\rangle + ... \tag{9.17}$$

That is, through our Quantum Message passing GNN, we obtained the desired state in Eq. 9.2.

Part III

GPT on Quantum Computer

Chapter 10

Transformer on Quantum Computer 1: Background

The emergence and rapid advancement of Large Language Models (LLMs) such as ChatGPT has had a significant global impact, revolutionizing our interactions with artificial intelligence and expanding our understanding of its capabilities. Models like GPT-4 have demonstrated the vast potential of LLMs in a wide range of applications across various domains. In the field of natural language processing (NLP), LLMs have proven to be highly proficient in tasks such as machine translation, sentiment analysis, question answering, and text summarization. They excel in identifying intricate language patterns, comprehending context, and generating text that is coherent and contextually appropriate.

Despite the significant achievements in QML, integrating quantum computing with state-of-the-art machine learning models, especially LLMs, is only at its nascent stages. Recent explorations such as Ref. [33, 34, 35, 36, 37] etc. indicate a burgeoning interest in leveraging quantum computing to elevate the capabilities of LLMs. In the third part of the thesis including Chapter 10 11, we explore the quantum implementation of GPT [14] (also referred as GPT-1, the first generation of the GPT series[1]) — the original version of ChatGPT. This chapter gives an overview of the background.

10.1 Transformer Architecture Used in GPT

The Generative Pre-trained Transformer (GPT) [14] is the inaugural version in the series of groundbreaking language models developed by OpenAI, marking the beginning of a new era in NLP. GPT's architecture is predicated on the transformer model [15], a type of neural network that relies on self-attention mechanisms to process sequences of data, such as text. With 117

[1]In this thesis, we use "GPT" instead of "GPT-1"

million parameters, GPT was a large model for its time, capable of capturing complex language patterns and generating coherent and contextually relevant text. GPT's introduction was a pivotal moment in NLP; it paved the way for the development of more advanced models, such as GPT-2 and GPT-3, setting the stage for the rapid advancement of AI technologies in the years that followed. The remainder of this section gives an overview of GPT's architecture and its training, and the next section 10.2 presents its application in language modeling.

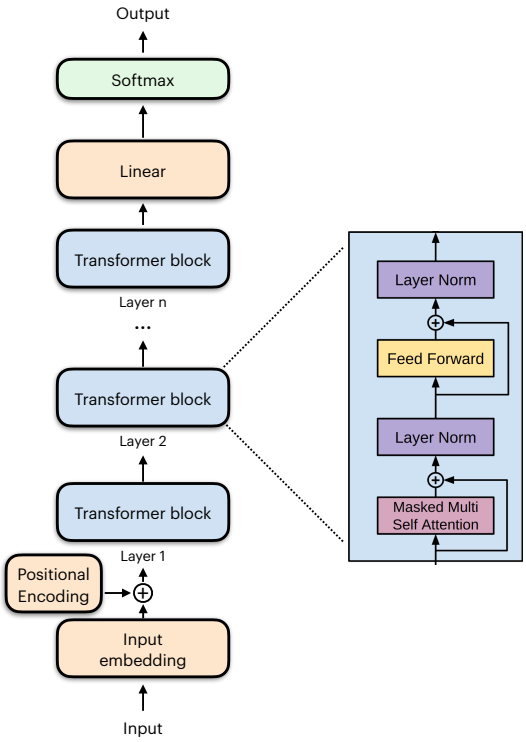

Figure 10.1: *GPT's Architecture, adapted from [14].* GPT's architecture is a multi-layer decoder-only Transformer (a variant of the Transformer[15]). The primary part of the architecture is a stack of transformer blocks [14], each of which is composed of two main components: a (masked) multi-head self-attention mechanism followed by a position-wise fully connected feed-forward network. Layer Normalization and Residual Connections are placed around these two main components. The transformer blocks are stacked on top of each other, with each layer processing the output of the previous one. Prior to the input embedding entering the transformer blocks, positional encoding is added.

GPT's architecture is a multi-layer decoder-only Transformer (a variant of the Transformer[15]). The primary part of the architecture is a stack of transformer blocks [14], each of which is composed of two main components: a (masked) multi-head self-attention mechanism followed by a position-wise fully connected feed-forward network. Layer Normalization and Residual Connections are placed around these two main components. The transformer blocks are stacked on top of each other, with each layer processing the output of the previous one. Prior to the input embedding entering the transformer blocks, positional encoding is added. Fig.10.1 illustrates

these components in GPT's architecture, the following subsubsections briefly[2] introduce each component.

Multi-Head Masked Self-Attention Mechanism

The multi-head self-attention mechanism, with the addition of a masking operation, is a core component of the transformer block in GPT. This attention mechanism allows a model to weigh the importance of different words in a sentence. Unlike previous models (such as Recurrent Neural Networks(RNNs) [128]) that process words in a sequential manner, self-attention enables the model to look at all parts of the sentence simultaneously. This allows for a more nuanced understanding of context and relationships between words, regardless of their position in the sentence. The masking operation is a critical aspect of this mechanism, especially in the context of language modeling(a brief introduction is given in Section 10.2): it ensures that the prediction of a current word does not get influenced by future words. [16]

Layer Normalization and Residual Connections

Each transformer block in GPT includes layer normalization and residual connections. Layer Normalization is applied after the self-attention mechanism and after the feed-forward network within each transformer block. It normalizes the inputs across the features, improving the stability of the model. Residual Connections allow the input of each sub-layer (i.e., the self-attention and feed-forward networks) to be added to its output.

Position-Wise Fully Connected Feed-Forward Network

In each transformer block in GPT, after the attention mechanism together with corresponding Layer Normalization and Residual Connection, the output is passed through a feed-forward network that applies the same transformation to each position separately and identically.

Positional Encoding

Since GPT (and transformer models in general) does not inherently process sequential data in order, it uses positional encodings to incorporate information about the order of the sequence into its inputs. These positional encodings are added to the input embeddings at the bottom of the model stack, providing the model with information about the position of each word in the

[2]Here in this section, for each component in GPT's architecture, we only give an overview, the detailed mathematical descriptions of each component are presented in Section **??**.

sequence.

The training process of GPT consists of two main stages [14]: unsupervised pre-training and supervised fine-tuning. During pre-training, GPT is exposed to a large corpus of text data, learning the underlying structure of the language without any task-specific instructions. This stage allowed the model to develop a broad understanding of language, including grammar, semantics, and common phrases. The fine-tuning stage then adapted the pre-trained model to specific tasks, such as translation, question-answering, and summarization, by training it on smaller, task-specific datasets.

10.2 Language Modeling basics

This section briefly introduces one of GPT's applications in language modeling — text generation, which is the foundation of the services provided by ChatGPT. We start by presenting an overview of language modelling:

Language modeling is a fundamental aspect of natural language processing (NLP) that focuses on the development of probabilistic models capable of understanding, generating, and interpreting human language. At its core, a language model predicts the likelihood of upcoming sequences of words occurring in a text. This predictive capability enables a wide range of applications, from autocomplete systems in smartphones and email platforms to sophisticated chatbots, machine translation, speech recognition, and even content generation tools.

The essence of language modeling lies in capturing the statistical structure of language—learning how words and phrases tend to come together to convey meaning. Early language models were relatively simple n-gram models, where the prediction of the next word in a sequence depended on the previous $n-1$ words. However, these models had limitations, particularly in dealing with long-term dependencies and the vast diversity of linguistic contexts. The advent of neural networks brought a significant leap forward in language modeling. Recurrent Neural Networks (RNNs)[128], and later, Long Short-Term Memory (LSTM) networks[129], provided mechanisms to remember information over longer sequences, improving the model's ability to handle context in language. Despite these advances, RNNs and LSTMs also faced challenges, such as difficulty in training over very long sequences and capturing complex dependencies.

The breakthrough came with the introduction of the Transformer architecture[15], which led to the development of models like OpenAI's GPT series that demonstrated unprecedented capabilities in generating coherent and contextually relevant text over extended passages. The development of language models continues to be a vibrant area of research in AI, with ongoing work aimed at improving their accuracy, efficiency, and ability to understand and generate

human language in all its complexity.

Next, we delve into fundamental concepts of language modeling, such as tokenization, explaining how a language model (here, GPT) operates for text generation.

10.2.1 Tokenization, Word Embedding

Tokenization is the process of converting text into tokens which are the basic units of language models. There are three levels of tokenization:

- Character-level: Processes text one letter at a time.
- Word-level: Segments text into individual words.
- Subword-level: Breaks down words into subunits. For example, the subword tokenization of the phrase "Language Model" may look like ["Lan", "gu", "age ", "Mod", "el"].

Upon establishing the tokens for a language model, we arrange them into a structured vocabulary, assigning a distinct index to each token. These indices are then transformed into input features through various methodologies. Directly inputting these indices into the model is inadvisable, as the sequential order within the vocabulary does not inherently reflect semantic relationships. An alternative is the utilization of one-hot encoding. For a vocabulary encompassing 10,000 words, each word is symbolized by a 10,000-element vector, predominantly composed of zeros, save for the element corresponding to the word's indexed position. The primary benefit of one-hot encoding is its ability to preclude presumptions about word importance, facilitating the model's learning of word relationships during its training phase.

However, one-hot encoding presents scalability issues in the context of extensive vocabularies. Considering the English language, with its repertoire exceeding 100,000 words, representing a single word would necessitate a vector comprising 100,000 elements. In scenarios involving lengthy sequences, this approach demands substantial storage space and computational resources. To mitigate this issue of dimensionality, the use of word embeddings is proposed (e.g.[130, 131, 132, 133]). In this framework, an embedding layer projects the one-hot encoded tokens into a more condensed vector space. These embeddings, essentially denser token representations, are generated through a linear layer equipped with a weight matrix, which the model optimizes during its training process. Fig.10.2 summarizes this section.

10.2.2 Next token prediction

In the inference stage, a language model (here, GPT) takes in a sequence of one-hot encoded tokens, and generates predictions for the next word in a sequence.[3] The sequence of one-hot

[3]Here, we only consider autoregressive language models.

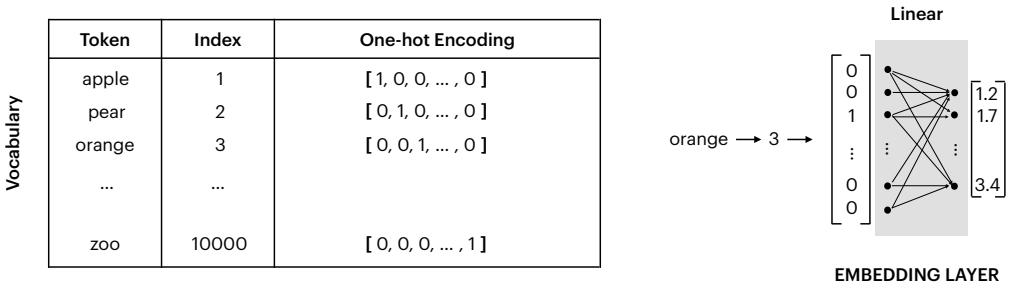

Figure 10.2: *Tokenization, one-hot encoding and word embedding.*

encoded tokens is first transformed through word embedding, as described in the above section. These embeddings, after the positional encodings are added, are then input into the transformer blocks. Then, a final linear layer is applied to map the outputs from the transformer blocks back into the vocabulary space, generating a sequence of transformed vectors. The last transformed vector in the sequence is referred as "logits". The logits are passed through a softmax activation function, yielding a probability distribution across the vocabulary, indicating the likelihood of each word as the next sequence component. Fig.10.3 summarizes this section.

10.2.3 Generative Pre-training

GPT model undergo extensive pre-training on large text corpora using the following loss function:

$$L(\theta) = -\sum_{t} \log P(w_t | w_{1:t-1}; \theta) \tag{10.1}$$

where w_t is the t-th word, and θ represents the model parameters. This pre-training endows GPT with a broad understanding of language, which is then refined for specific tasks through fine-tuning.

Given the unlabeled nature of these sentences, this process is classified as unsupervised learning. It involves the pairing of a text segment as input with its subsequent segment as the target. The training process encompasses processing these input-target pairs in batches. The loss is computed by evaluating the next token in the target output, and this process is repeated for each subsequent token in the sequence. The cumulative loss is calculated across all batches, followed by the execution of backpropagation to adjust the model's parameters.

The culmination of this process is a pre-trained language model, which we can then employ for text generation. This begins with the input of an initial word or phrase, serving as the genesis for text generation. The model assesses this input to predict the next token, which is subsequently reintroduced into the model as the new input. This iterative process engenders a feedback loop, enabling the model to generate continuous text sequences.

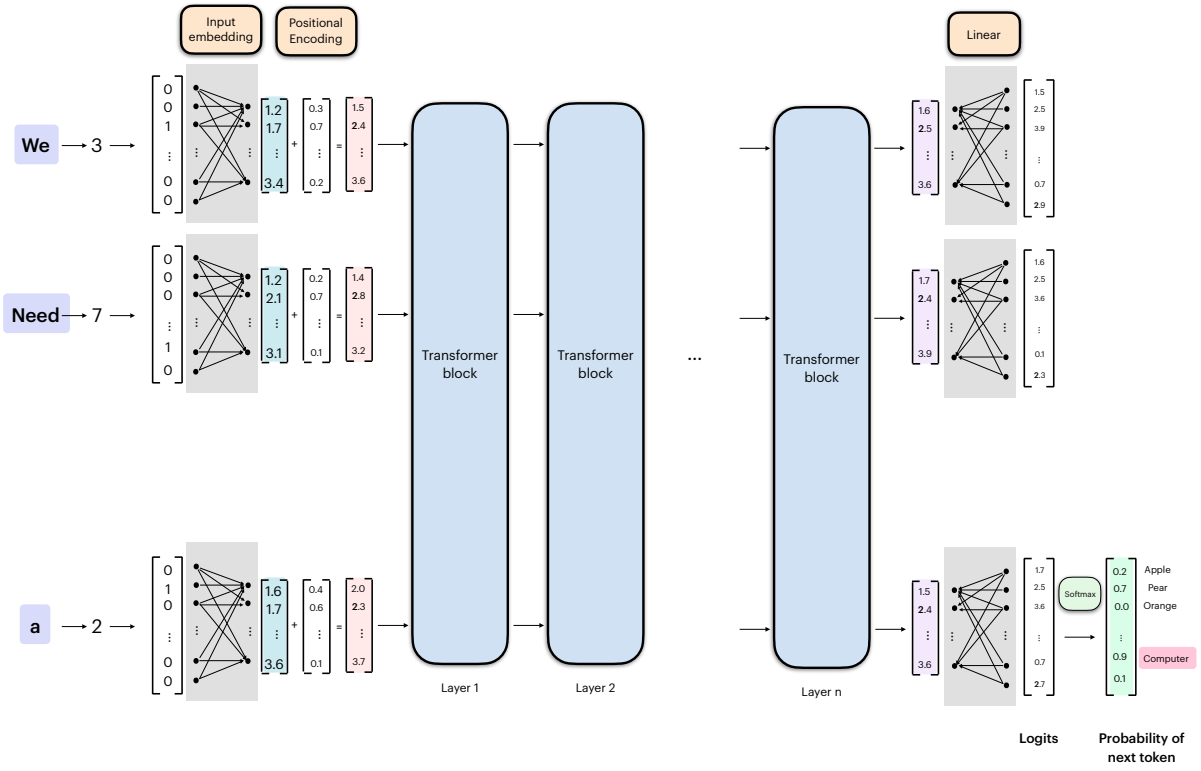

Figure 10.3: *Next token prediction process of GPT*. In the inference stage, a language model (here, GPT) takes in a sequence of one-hot encoded tokens, and generates predictions for the next token in a sequence. The sequence of one-hot encoded tokens is first transformed through word embedding, as described in the above section. These embeddings, after the positional encodings are added, are then input into the transformer blocks. Then, a final linear layer is applied to map the outputs from the transformer blocks back into the vocabulary space, generating a sequence of transformed vectors. The last transformed vector in the sequence is referred as "logits". The logits are passed through a softmax activation function, yielding a probability distribution across the vocabulary, indicating the likelihood of each token as the next sequence component.

Chapter 11

Transformer on Quantum Computer 2: Architecture implementation

As outlined in Section 10.1 and depicted in Figure 10.1, the architecture of GPT encompasses several key elements: input embedding, positional encoding, a series of transformer blocks, and a concluding linear layer followed by a softmax function. Within the context of this thesis, it is assumed that both the input embedding and positional encoding are executed on classical computers. Our focus, however, shifts to detailing the implementation of the transformer blocks' core components on a quantum computer. Additionally, we delve into the methodologies employed for executing Generative Pre-training of the model on a quantum computer. To ensure a holistic presentation, a detailed exposition on the quantum implementation of positional encoding is provided in section 11.2.

11.1 Input encoding by CQSP

The input to the Transformer block is a sequence of vectors $\left\{\boldsymbol{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$ stacked as a matrix $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d \times n}$. It can be encoded in a quantum state(after normalization) $|\psi_{\boldsymbol{X}}\rangle$ as,

$$|\psi_{\boldsymbol{X}}\rangle := \sum_{i=1}^{n} |i\rangle |\boldsymbol{x}_i\rangle , \tag{11.1}$$

where $|\boldsymbol{x}_i\rangle := \sum_{k=1}^{d} \boldsymbol{x}_i^{(k)} |k\rangle$ is the amplitude encoding of the vector $\boldsymbol{x}_i$ whose k-th elements are denoted as $\boldsymbol{x}_i^{(k)}$.

The entire state is prepared on two quantum registers hosting the index k and index i, which are denoted as $Reg(k)$ and $Reg(i)$, respectively. The unitary that realizes the data encoding as,

$$U_{\boldsymbol{X}} : |i\rangle |0\rangle \to |i\rangle |\boldsymbol{x}_i\rangle , \forall i = 1, \cdots n, \tag{11.2}$$

is represented as the blue box in Fig.11.3. $U_{\boldsymbol{X}}$ can be achieved by "Controlled Quantum State Preparation(CQSP)" process[9].

11.2 Positional encoding by multi-controlled R_Y gate

The positional encoding mentioned in Section 10.1 can be described as follows[16]: Corresponding to the i-th vector in the sequence $\boldsymbol{x}_i \in \mathbb{R}^d$, define the position vector $\boldsymbol{p}_i \in \mathbb{R}^d$ as:

$$\begin{cases} \boldsymbol{p}_i^{(2j+1)} := \cos\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \\ \boldsymbol{p}_i^{(2j)} := \sin\left(\frac{i}{10000^{\frac{2j}{d}}}\right) \end{cases} \tag{11.3}$$

for all $j \in \{0, 1, \ldots, \lfloor d/2 \rfloor\}$, where $\boldsymbol{p}_i^{(2j+1)}$ and $\boldsymbol{p}_i^{(2j)}$ denote the odd and even elements of $\boldsymbol{p}_i$, respectively. For encoding of positional information into data, the position vectors are added directly to the input vectors:

$$\boldsymbol{x}_i' = \boldsymbol{x}_i + \boldsymbol{p}_i, \tag{11.4}$$

where $\boldsymbol{x}_i'$ is the appended input vectors that contain positional information, and we define $\boldsymbol{X}' := [\boldsymbol{x}_1', \ldots, \boldsymbol{x}_n'] \in \mathbb{R}^{d \times n}$. as the matrix by stacking $\boldsymbol{x}_i'$.

Our quantum algorithm for positional encoding aims to create the quantum state

$$\left|\psi_{\boldsymbol{X}'}\right\rangle := \sum_{i=1}^{n} |i\rangle \left|\boldsymbol{x}_i'\right\rangle, \tag{11.5}$$

where $|\boldsymbol{x}_i'\rangle := \sum_{k=1}^{d} \boldsymbol{x}_i'^{(k)} |k\rangle$ is the amplitude encoding of the vector $\boldsymbol{x}_i'$ whose k-th elements are denoted as $\boldsymbol{x}_i'^{(k)}$. Similarly, we define $|\boldsymbol{p}_i\rangle := \sum_{k=1}^{d} \boldsymbol{p}_i^{(k)} |k\rangle$ which is the amplitude encoding of the vector $\boldsymbol{p}_i$ whose k-th elements are denoted as $\boldsymbol{p}_i^{(k)}$ and

$$|\psi_{\boldsymbol{P}}\rangle := \sum_{i=1}^{n} |i\rangle |\boldsymbol{p}_i\rangle \tag{11.6}$$

From the above definitions of $|\boldsymbol{x}_i'\rangle, |\boldsymbol{p}_i\rangle$ and Eqn. 11.1,11.4,11.5,11.6 we have

$$\left|\psi_{\boldsymbol{X}'}\right\rangle = |\psi_{\boldsymbol{X}}\rangle + |\psi_{\boldsymbol{P}}\rangle = \sum_{i=1}^{n} |i\rangle |\boldsymbol{x}_i\rangle + \sum_{i=1}^{n} |i\rangle |\boldsymbol{p}_i\rangle \tag{11.7}$$

Note that

$$|\boldsymbol{p}_i\rangle = \sum_{k=1}^{d} \boldsymbol{p}_i^{(k)} |k\rangle = \sum_{j} (\boldsymbol{p}_i^{(2j)} |2j\rangle + \boldsymbol{p}_i^{(2j+1)} |2j+1\rangle) \tag{11.8}$$

Denote the unitary that prepares $|\psi_{\boldsymbol{P}}\rangle$ from $\sum_{i=1}^{n} |i\rangle |0\rangle$ as $U_{\boldsymbol{P}}$, then $\left|\psi_{\boldsymbol{X}'}\right\rangle$ can be achieved by applying LCU to $U_{\boldsymbol{X}}$,$U_{\boldsymbol{P}}$. As the construction of $U_{\boldsymbol{X}}$ is given by the CQSP mentioned in

Section 11.1, we can focus on the construction of $U_{\boldsymbol{P}}$. Next, we present the construction of $U_{\boldsymbol{P}}$ as depicted in Fig.11.1.

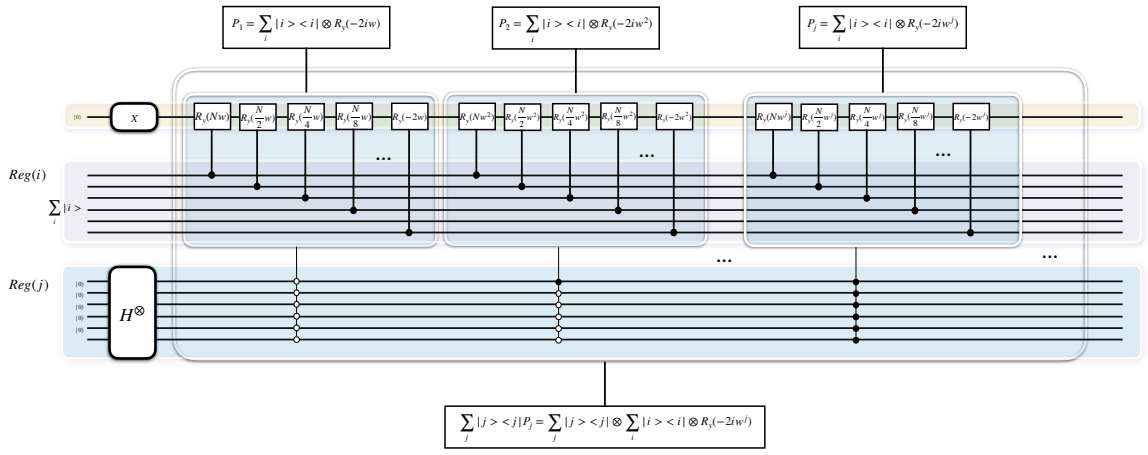

Figure 11.1: *Quantum circuit for positional encoding* Build upon the two registers $Reg(i)$ and $Reg(k)$ hosting the index i and k respectively (illustrated in Fig. 11.3), we set up a register $Reg(j)$ hosting the index j below $Reg(i)$ and an ancillary qubit above $Reg(i)$. The blue boxes that group the series of controlled R_Y gates implement the following unitaries: $P_j = \sum_i |i\rangle \langle i| \otimes R_y(-2iw^j)$, each of which is controlled by the qubits in $Reg(j)$ and the entire controlled sequences grouped in the transparent box implement the unitary $U_c = \sum_j |j\rangle \langle j| \otimes P_j = \sum_j |j\rangle \langle j| \otimes \sum_i |i\rangle \langle i| \otimes R_y(-2iw^j)$. The whole circuit implements $U_{\boldsymbol{P}}$. Note that in this figure, $N = -n$.

Build upon the two registers $Reg(i)$ and $Reg(k)$ hosting the index i and k respectively (illustrated in Fig. 11.3), in Fig. 11.1, we set up a register $Reg(j)$ hosting the index j below $Reg(i)$ and an ancillary qubit above $Reg(i)$. For reasons that will be clear soon, we combine $Reg(j)$ and the ancillary qubit as a single register which coincides with $Reg(k)$. The blue boxes that group the series of controlled R_Y gates implement the following unitaries:

$$P_j = \sum_i |i\rangle \langle i| \otimes R_y(-2iw^j) \tag{11.9}$$

each of which is controlled by the qubits in $Reg(j)$, and the entire controlled sequences grouped in the transparent box implement the unitary

$$U_c = \sum_j |j\rangle \langle j| \otimes P_j = \sum_j |j\rangle \langle j| \otimes \sum_i |i\rangle \langle i| \otimes R_y(-2iw^j) \tag{11.10}$$

The Hadamard gates and X gate before U_c transform the input state to the state $\sum_j |j\rangle \otimes \sum_i |i\rangle \otimes |1\rangle$, after U_c, it becomes

$$U_c(\sum_j |j\rangle \otimes \sum_i |i\rangle \otimes |1\rangle) = \sum_j |j\rangle \otimes \sum_i |i\rangle \otimes R_y(-2iw^j)|1\rangle \tag{11.11}$$

By placing $Reg(j)$ and the ancillary qubit next to each other we can rewrite the above state as:

$$\sum_i |i\rangle \otimes \sum_j |j\rangle \otimes R_y(-2iw^j)\,|1\rangle \tag{11.12}$$

$$= \sum_i |i\rangle \otimes \sum_j |j\rangle \otimes (\sin(iw^j)\,|0\rangle + \cos(iw^j)\,|1\rangle). \tag{11.13}$$

Combining $Reg(j)$ and the ancillary qubit as a single register that coincides with $Reg(k)$, the computational basis transform as $|j\rangle\,|0\rangle \to |2j\rangle$,$|j\rangle\,|1\rangle \to |2j+1\rangle$ and we can write the output state from the circuit in Fig.11.1 as:

$$|\text{output}\rangle = \sum_i |i\rangle \otimes \sum_j (\sin(iw^j)\,|2j\rangle + \cos(iw^j)\,|2j+1\rangle) \tag{11.14}$$

Set $w = \frac{1}{10000^{\frac{1}{d}}}$, and from Eqn. 11.3,11.6,11.8,11.14 we have

$$|\text{output}\rangle = |\psi_{\boldsymbol{P}}\rangle \tag{11.15}$$

That is, the circuit in Fig.11.1 implements $U_{\boldsymbol{P}}$.

11.3 Attentions on Quantum Computer

An attention function can be described as mapping queries, keys, and values to an output, where the queries, keys, values, and output are all vectors[15]. The query $\boldsymbol{q}_i$, key $\boldsymbol{k}_i$, and value $\boldsymbol{v}_i$ are p-dimensional, p-dimensional, and r-dimensional vectors defined as:[16]

$$\mathbb{R}^p \ni \boldsymbol{q}_i = \boldsymbol{W}_Q^\top \boldsymbol{x}_i, \tag{11.16}$$

$$\mathbb{R}^p \ni \boldsymbol{k}_i = \boldsymbol{W}_K^\top \boldsymbol{x}_i, \tag{11.17}$$

$$\mathbb{R}^r \ni \boldsymbol{v}_i = \boldsymbol{W}_V^\top \boldsymbol{x}_i, \tag{11.18}$$

where $\boldsymbol{W}_Q \in \mathbb{R}^{d\times p}, \boldsymbol{W}_K \in \mathbb{R}^{d\times p}$, and $\boldsymbol{W}_V \in \mathbb{R}^{d\times r}$ are trainable matrices.[1] Similar to vectors $\left\{\boldsymbol{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$ being stacked as a matrix $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d\times n}$, we define $\boldsymbol{Q} := [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n] \in \mathbb{R}^{p\times n}$, $\boldsymbol{K} := [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n] \in \mathbb{R}^{p\times n}$, and $\boldsymbol{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n] \in \mathbb{R}^{r\times n}$.

The "Scaled Dot-Product Attention" defined in [15] can be written in matrix form as:

$$\boldsymbol{Z} := \text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{V}\,\text{softmax}\left(\frac{1}{\sqrt{p}}\boldsymbol{Q}^\top \boldsymbol{K}\right), \tag{11.19}$$

where $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n] \in \mathbb{R}^{r\times n}$.

Note that for each i, the queries, keys, and values are all from the same vector $\boldsymbol{x}_i$ in the sequence, this type of attention is referred to as the "self-attention"[16].

[1] The "transpose" in 11.18,11.17,11.16 are in the definition for some reason which will be clear in Section 11.5.

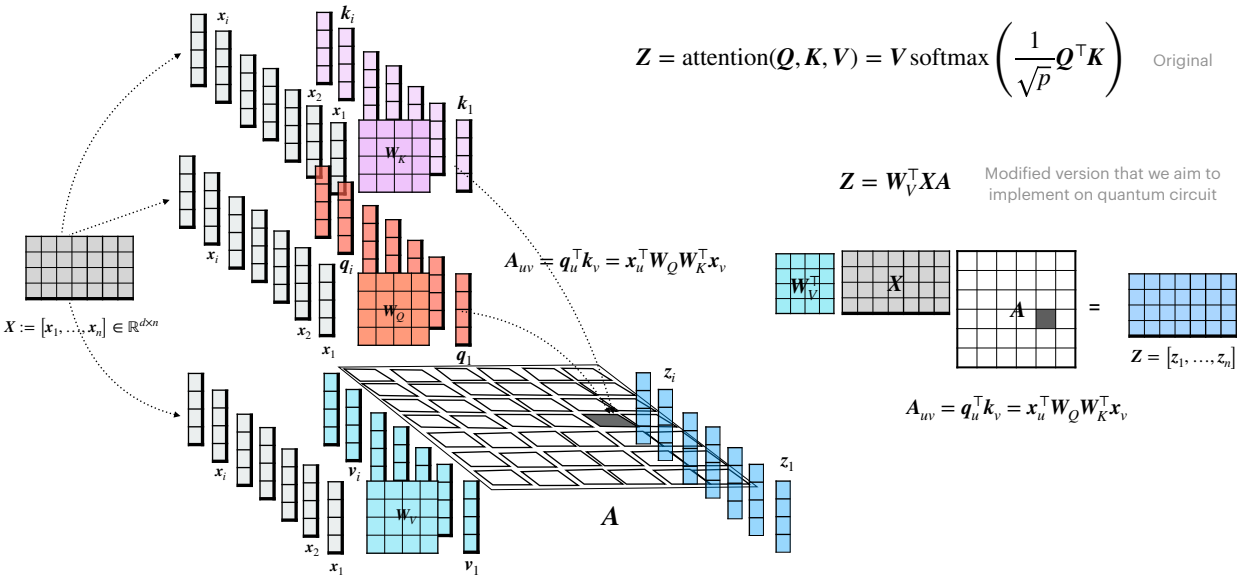

Figure 11.2: *The Classical Self-Attention(modified version that we aim to implement on quantum circuit)* As described in Ref.[12], an attention function can be described as mapping query, keys, values to an output, where the query, keys, values, and output are all vectors. The query $\boldsymbol{q}_i$, key $\boldsymbol{k}_i$, and value $\boldsymbol{v}_i$ are p-dimensional, p-dimensional, and r-dimensional vectors defined as: $\mathbb{R}^p \ni \boldsymbol{q}_i = \boldsymbol{W}_Q^\top \boldsymbol{x}_i, \mathbb{R}^p \ni \boldsymbol{k}_i = \boldsymbol{W}_K^\top \boldsymbol{x}_i, \mathbb{R}^r \ni \boldsymbol{v}_i = \boldsymbol{W}_V^\top \boldsymbol{x}_i$, where $\boldsymbol{W}_Q \in \mathbb{R}^{d\times p}, \boldsymbol{W}_K \in \mathbb{R}^{d\times p}$, and $\boldsymbol{W}_V \in \mathbb{R}^{d\times r}$ are the projection matrices. Similar to vectors $\left\{\boldsymbol{x}_i \in \mathbb{R}^d\right\}_{i=1}^n$ being stacked as a matrix $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n] \in \mathbb{R}^{d\times n}$, we define $\boldsymbol{Q} := [\boldsymbol{q}_1, \ldots, \boldsymbol{q}_n] \in \mathbb{R}^{p\times n}$, $\boldsymbol{K} := [\boldsymbol{k}_1, \ldots, \boldsymbol{k}_n] \in \mathbb{R}^{p\times n}$, and $\boldsymbol{V} := [\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n] \in \mathbb{R}^{r\times n}$, respectively.The "Scaled Dot-Product Attention" defined in [15] can be written in matrix form as:$\boldsymbol{Z} := \text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) = \boldsymbol{V}\,\text{softmax}\left(\frac{1}{\sqrt{p}}\boldsymbol{Q}^\top \boldsymbol{K}\right)$, where $\boldsymbol{Z} = [\boldsymbol{z}_1, \ldots, \boldsymbol{z}_n] \in \mathbb{R}^{r\times n}$. Note that for each i, the queries, keys, and values are all from the same vector $\boldsymbol{x}_i$ in the sequence, this type of attention is referred to as the "self-attention"[16].Denote $\text{softmax}\left(\frac{1}{\sqrt{p}}\boldsymbol{Q}^\top \boldsymbol{K}\right) \equiv \boldsymbol{A}_0$, plugging in $\boldsymbol{V} = \boldsymbol{W}_V^\top \boldsymbol{X}$ we have $\boldsymbol{Z} = \boldsymbol{W}_V^\top \boldsymbol{X}\boldsymbol{A}_0$. Considering it's not straightforward to implement the softmax function using quantum circuit and the scaling will be taken care of in block-encoding procedure, we aim to implement an alternative version of $\boldsymbol{A}_0$ denoted as $\boldsymbol{A} \equiv \boldsymbol{Q}^\top \boldsymbol{K}$, that is, we aim to design quantum circuit implementing the following computation:$\boldsymbol{Z} = \boldsymbol{W}_V^\top \boldsymbol{X}\boldsymbol{A}$ .Note that the matrix elements of $\boldsymbol{A}$ are $\boldsymbol{A}_{uv} = \boldsymbol{q}_u^\top \boldsymbol{k}_v = \boldsymbol{x}_u^\top \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_v$

11.3.1 Self-Attention on Quantum Computer

The "Scaled Dot-Product Attention" defined in 11.19 can also be written as follows, by plugging in $\boldsymbol{V} = \boldsymbol{W}_V^\top \boldsymbol{X}$ and denoting $\boldsymbol{A}_0 \equiv \text{softmax}\left(\frac{1}{\sqrt{p}}\boldsymbol{Q}^\top \boldsymbol{K}\right)$:

$$\boldsymbol{Z} := \text{attention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \tag{11.20}$$

$$= \boldsymbol{W}_V^\top \boldsymbol{X}\boldsymbol{A}_0 \tag{11.21}$$

Considering it's not straightforward to implement the softmax function [2] using quantum circuit and the scaling will be taken care of in the block-encoding procedure described later in this section, we aim to implement an alternative version of $\boldsymbol{A}_0$ denoted as $\boldsymbol{A} \equiv \boldsymbol{Q}^\top \boldsymbol{K}$, that is, we aim to design quantum circuit implementing the following computation:

$$\boldsymbol{Z} = \boldsymbol{W}_V^\top \boldsymbol{X} \boldsymbol{A}, \tag{11.22}$$

Note that the matrix elements of $\boldsymbol{A}$ are

$$\boldsymbol{A}_{uv} = \boldsymbol{q}_u^\top \boldsymbol{k}_v = \boldsymbol{x}_u^\top \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_v \tag{11.23}$$

The above description of classical attention (the modified version that we aim to implement on quantum circuit) can be illustrated in Fig.11.2. Next, we present its quantum implementation.

On quantum circuit, after the input encoding described in 11.1 (here in this subsection and the following subsection 11.3.2, for simplicity we omit the positional encoding described in 11.2[3]), The attention function can be implemented by applying the block-encoding of $\boldsymbol{A}^\top$ and a parameterized quantum circuit for $\boldsymbol{W}_V^\top$ on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively, as depicted in Fig.11.3.

This can be proven as follows: Starting From Eqn. 11.22

$$\boldsymbol{Z} = \boldsymbol{W}_V^\top \boldsymbol{X} \boldsymbol{A},$$

Utilising the formula $vec(ABC) = (C^T \otimes A) vec(B)$

$$vec(\boldsymbol{Z}) = vec(\boldsymbol{W}_V^\top \boldsymbol{X} \boldsymbol{A}) = (\boldsymbol{A}^\top \otimes \boldsymbol{W}_V^\top) vec(\boldsymbol{X}) \tag{11.24}$$

For a matrix M, define vectors $\boldsymbol{\psi}_M = vec(M)$, and Eqn.11.24 becomes

$$\boldsymbol{\psi}_{\boldsymbol{Z}} = (\boldsymbol{A}^\top \otimes \boldsymbol{W}_V^\top) \boldsymbol{\psi}_{\boldsymbol{X}} \tag{11.25}$$

Recall Eqn.11.1:

$$|\psi_{\boldsymbol{X}}\rangle = \sum_{i=1}^{n} |i\rangle |\boldsymbol{x}_i\rangle$$

Writing the quantum states in Eqn.11.1 as vectors we note that

$$\boldsymbol{\psi}_{\boldsymbol{X}} = |\psi_{\boldsymbol{X}}\rangle \tag{11.26}$$

And correspondingly we have

[2] Exploring alternatives to the softmax function in attention mechanisms has garnered interest due to the potential for efficiency gains and improved model performance. Research has demonstrated that it's possible to achieve high performance without the need for softmax normalization [134, 135].

[3] One can consider the input encoding and positional encoding are combined as "appended" input encoding

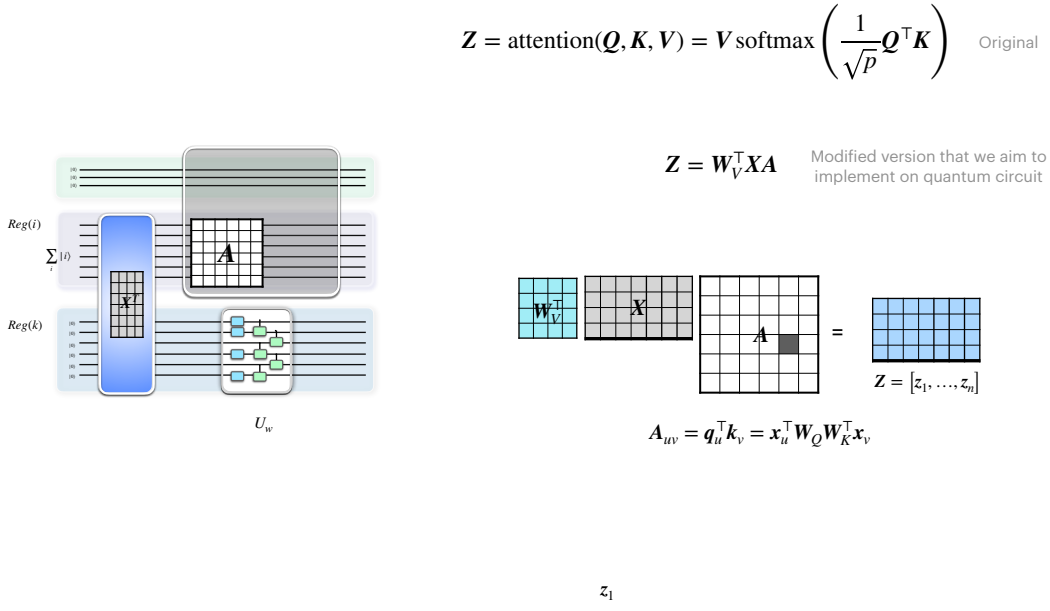

Figure 11.3: *Self-Attention on Quantum Computer* The right side of the figure describes classical attention (the modified version that we aim to implement on quantum circuit), and the left side of the figure depicts its quantum implementation. On the quantum circuit, the input encoding is represented by the blue box, as described in 11.1(here in this subsection and the following subsection 11.3.2, for simplicity we omit the positional encoding described in 11.2 in the presentation). The attention function can be implemented by applying the block-encoding of $\boldsymbol{A}^\top$ and a parameterized quantum circuit for $\boldsymbol{W}_V^\top$ on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively.

$$\boldsymbol{\psi}_{\boldsymbol{Z}} = |\psi_{\boldsymbol{Z}}\rangle \tag{11.27}$$

by defining

$$|\psi_{\boldsymbol{Z}}\rangle := \sum_{i=1}^{n} |i\rangle\,|\boldsymbol{z}_i\rangle \tag{11.28}$$

where $|\boldsymbol{z}_i\rangle := \sum_{k=1}^{d} \boldsymbol{z}_i^{(k)} |k\rangle$ is the amplitude encoding of the vector $\boldsymbol{z}_i$ whose k-th elements are denoted as $\boldsymbol{z}_i^{(k)}$.

From Eqn.11.27 to 11.25 we have

$$|\psi_{\boldsymbol{Z}}\rangle = (\boldsymbol{A}^\top \otimes \boldsymbol{W}_V^\top)\,|\psi_{\boldsymbol{X}}\rangle \tag{11.29}$$

This amounts to the action of the quantum circuit for Self-Attention which can be written as:

$$|\psi_{\boldsymbol{Z}}\rangle \otimes |0\rangle + \ldots = (U_{\boldsymbol{A}^\top} \otimes U_{\boldsymbol{W}_V^\top})(|\psi_{\boldsymbol{X}}\rangle \otimes |0\rangle) \tag{11.30}$$

where $U_{\boldsymbol{A}^\top}$ corresponds to applying the block-encoding of $\boldsymbol{A}^\top$ and $U_{\boldsymbol{W}_V^\top}$ a parameterized quantum circuit implementing $\boldsymbol{W}_V^\top$ on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively, "+..." indicates upon post-selecting we can obtain the desired state $|\psi_{\boldsymbol{Z}}\rangle = vec(\boldsymbol{Z})$.

The block-encoding of $\boldsymbol{A}^\top$ can be constructed using the following lemma from Ref.[136].

Lemma 3.2 from Ref.[136] (*Naive block-encoding of dense matrices with oracle access*). Let $A \in \mathbb{C}^{N\times N}$ (where $N = 2^s$) with a_{ij} being its elements and let $\hat{a} \geqslant \max_{i,j}|a_{ij}|$. Suppose the following oracle is provided

$$O_A : |i\rangle|j\rangle|0\rangle^{\otimes b} \to |i\rangle|j\rangle|\tilde{a}_{ij}\rangle,$$

where $0 \leqslant i, j < N$ and $\tilde{a}_{ij}$ is the (exact) b-qubit description of $a_{ij}/\hat{a}$. Then one can implement a $(N\hat{a}, s+1, \epsilon)$-block-encoding of A with two uses of O_A, $O(\text{polylog}(\hat{a}N/\epsilon))$ one- and two-qubit gates and $O(b, \text{polylog}(\hat{a}N/\epsilon))$ extra qubits (which are discarded before the post-selection step).

Below we explain that the block-encoding of $\boldsymbol{A}^\top \in \mathbb{R}^{n\times n}$ can be constructed using the above lemma:

From Eqn.11.23, the matrix elements of $\boldsymbol{A}^\top$ (which we denote as Λ_{ij}) are

$$\Lambda_{ij} := \boldsymbol{A}^\top{}_{ij} = \boldsymbol{x}_j^\top \boldsymbol{W}_Q \boldsymbol{W}_K^\top \boldsymbol{x}_i \tag{11.31}$$

let $\hat{\Lambda} \geqslant \max_{i,j}|\Lambda_{ij}|$ and $\tilde{\Lambda}_{ij}$ is defined to be the (exact) b-qubit description of $\Lambda_{ij}/\hat{\Lambda}$.

According to the above lemma, we need the following oracle to the block-encoding of $\boldsymbol{A}^\top$

$$O_{\boldsymbol{A}^\top} : |i\rangle|j\rangle|0\rangle^{\otimes b} \to |i\rangle|j\rangle|\tilde{\Lambda}_{ij}\rangle, \tag{11.32}$$

where $0 \leqslant i, j < n$.

This oracle $O_{\boldsymbol{A}^\top}$ can be constructed in the same way(described in section 8.1) as $O_{\mathsf{attention}}$ defined in Eqn. 8.2 with the attention score as Eqn. 8.3. Therefore, substituting A in the above lemma with $\boldsymbol{A}^\top$, we can implement the block-encoding of $\boldsymbol{A}^\top$ with two uses of $O_{\boldsymbol{A}^\top}$, $O(\text{polylog}(\hat{\Lambda}n/\epsilon))$ one- and two-qubit gates.

Masked-Attention

The masked attention is defined as:[16]

$$\begin{aligned}\mathbb{R}^{r\times n} \ni \boldsymbol{Z}_m &:= \text{maskedAttention}(\boldsymbol{Q}, \boldsymbol{K}, \boldsymbol{V}) \\ &= \boldsymbol{V}\,\text{softmax}\left(\frac{1}{\sqrt{p}}\left(\boldsymbol{Q}^\top\boldsymbol{K} + \boldsymbol{M}\right)\right),\end{aligned}$$

where the mask matrix $\boldsymbol{M} \in \mathbb{R}^{n\times n}$ is:

$$\boldsymbol{M}_{ij} := \begin{cases} 0 & \text{if } j \leqslant i \\ -\infty & \text{if } j > i \end{cases}$$

For positions $j \leqslant i$ in a sequence (representing current or previous words), the mask doesn't alter the softmax output, this allows these positions to contribute to the softmax computation. In contrast, for positions $j > i$ (corresponding to future words in the sequence), the nature of softmax function indicates that the mask effectively nullifies their contribution as $e^{-\infty}$ is 0. This selective masking ensures that the model's attention is appropriately focused on the relevant parts of the sequence: considering past and present words while ignoring future words.

Denoting $\boldsymbol{A}_0^{\mathsf{Mask}} \equiv \text{softmax}\left(\frac{1}{\sqrt{p}}\left(\boldsymbol{Q}^\top \boldsymbol{K} + \boldsymbol{M}\right)\right)$, we have its elements as

$$\boldsymbol{A}_0^{\mathsf{Mask}}{}_{ij} := \begin{cases} \boldsymbol{A}_{0ij} & \text{if } j \leqslant i \\ 0 & \text{if } j > i \end{cases}$$

where $\boldsymbol{A}_{0ij}$ is the elements of $\boldsymbol{A}_0 \equiv \text{softmax}\left(\frac{1}{\sqrt{p}}\boldsymbol{Q}^\top \boldsymbol{K}\right)$.

In the quantum case, we aim to implement an alternative version of $\boldsymbol{A}_0^{\mathsf{Mask}}$ denoted as $\boldsymbol{A}^{\mathsf{Mask}}$ whose elements are defined as

$$\boldsymbol{A}^{\mathsf{Mask}}{}_{ij} := \begin{cases} \boldsymbol{A}_{ij} & \text{if } j \leqslant i \\ 0 & \text{if } j > i \end{cases}$$

where $\boldsymbol{A}_{ij}$ is the elements of $\boldsymbol{A} \equiv \boldsymbol{Q}^\top \boldsymbol{K}$.

For Masked-Attention, we aim to design quantum circuit implementing the following computation:

$$\boldsymbol{Z}'_m = \boldsymbol{W}_V^\top \boldsymbol{X} \boldsymbol{A}^{\mathsf{Mask}}, \tag{11.33}$$

This can be done in a similar way as in the case of self-attention where we implemented Eqn. 11.22, the only difference is that we now need the block-encoding of $\boldsymbol{A}^{\mathsf{Mask}^\top}$ (instead of $\boldsymbol{A}^\top$) whose elements are

$$\Lambda_{ij}^{\mathsf{Mask}} := \boldsymbol{A}^{\mathsf{Mask}^\top}{}_{ij} := \begin{cases} 0 & \text{if } j < i \\ \boldsymbol{A}_{ji} & \text{if } j \geqslant i \end{cases}$$

Similar to the case of self-attention, let $\hat{\Lambda}^{\mathsf{Mask}} \geqslant \max_{i,j} |\Lambda_{ij}^{\mathsf{Mask}}|$ and $\tilde{\Lambda}_{ij}^{\mathsf{Mask}}$ is defined to be the (exact) b-qubit description of $\Lambda_{ij}^{\mathsf{Mask}}/\hat{\Lambda}^{\mathsf{Mask}}$. The block-encoding of $\boldsymbol{A}^{\mathsf{Mask}^\top}$ can be constructed using lemma 3.2 from Ref.[12], given the following oracle

$$O_{\boldsymbol{A}^{\mathsf{Mask}^\top}} : |i\rangle|j\rangle|0\rangle^{\otimes b} \to |i\rangle|j\rangle|\tilde{\Lambda}_{ij}^{\mathsf{Mask}}\rangle, \tag{11.34}$$

where $0 \leqslant i, j < n$.

This oracle $O_{\boldsymbol{A}^{\mathsf{Mask}\top}}$ can be constructed by conditionally applying $O_{\boldsymbol{A}^\top}$: on the registers hosting $|i\rangle|j\rangle$, set up a circuit comparing the values of i, j with the result stored in an extra ancillary qubit (using Claim 3.1 in Ref.[13]). Then using this ancillary qubit as control qubit, apply controlled $O_{\boldsymbol{A}^\top}$ if $j \geqslant i$.

11.3.2 Multihead-Attention on Quantum computer

In the multihead attention module, We have H set of queries, values, and keys as: [16]

$$\begin{aligned}
\mathbb{R}^{p\times n} \ni \boldsymbol{Q}_h &= \boldsymbol{W}_{Q,h}^\top \boldsymbol{X}, \quad \forall h \in \{1,\ldots,H\},\\
\mathbb{R}^{p\times n} \ni \boldsymbol{V}_h &= \boldsymbol{W}_{V,h}^\top \boldsymbol{X}, \quad \forall h \in \{1,\ldots,H\},\\
\mathbb{R}^{r\times n} \ni \boldsymbol{K}_h &= \boldsymbol{W}_{K,h}^\top \boldsymbol{X}, \quad \forall h \in \{1,\ldots,H\}.
\end{aligned}$$

Then, the scaled dot product attention are applied to generate the H output $\{\boldsymbol{Z}_h\}_{h=1}^H$

$$\boldsymbol{Z}_h = \text{attention}(\boldsymbol{Q}_h, \boldsymbol{K}_h, \boldsymbol{V}_h) = \boldsymbol{V}_h \,\text{softmax}\left(\frac{1}{\sqrt{p}} \boldsymbol{Q}_h^\top \boldsymbol{K}_h\right), \tag{11.35}$$

and $\boldsymbol{Z}_h = \left[\boldsymbol{z}_{1,h}, \ldots, \boldsymbol{z}_{n,h}\right] \in \mathbb{R}^{r\times n}$. The outputs are concatenated over different heads as

$$\boldsymbol{Z}_{\mathsf{Multi\text{-}heads}} = \left[\|_{h=1}^H \boldsymbol{z}_{1,h}, \|_{h=1}^H \boldsymbol{z}_{2,h}, \ldots, \|_{h=1}^H \boldsymbol{z}_{n,h},\right] \in \mathbb{R}^{rH\times n} \tag{11.36}$$

where $\|$ represents concatenation [17]. Then, by a linear projection $\boldsymbol{W}_O^\top$, the total attention value is obtained:

$$\boldsymbol{z}_i^{\mathsf{Total}} := \boldsymbol{W}_O^\top \|_{h=1}^H \boldsymbol{z}_{i,h} \tag{11.37}$$

$$\boldsymbol{Z}^{\mathsf{Total}} := \boldsymbol{W}_O^\top \boldsymbol{Z}_{\mathsf{Multi\text{-}heads}}$$

and $\boldsymbol{Z}^{\mathsf{Total}} = \left[\boldsymbol{z}_1^{\mathsf{Total}}, \ldots, \boldsymbol{z}_n^{\mathsf{Total}}\right] \in \mathbb{R}^{rH\times n}$.

The above description of classical multihead attention can be illustrated in Fig.11.4. Next, we present its quantum implementation.

On quantum circuit, we aim to obtain the following quantum state

$$\left|\psi_{\boldsymbol{Z}^{\mathsf{Total}}}\right\rangle := \sum_{i=1}^n |i\rangle \otimes \left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle \tag{11.38}$$

where $\left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle$ is the amplitude encoding of the vector $\boldsymbol{z}_i^{\mathsf{Total}}$. This can be achieved via the quantum circuit depicted in Fig.11.4 in which the additional register $Reg(h)$ is hosting index h and the multi-controlled unitary is defined as

$$U_{\mathsf{Multi\text{-}heads}} = \sum_h |h\rangle\langle h| \otimes U_{\mathsf{Single\text{-}head}} \tag{11.39}$$

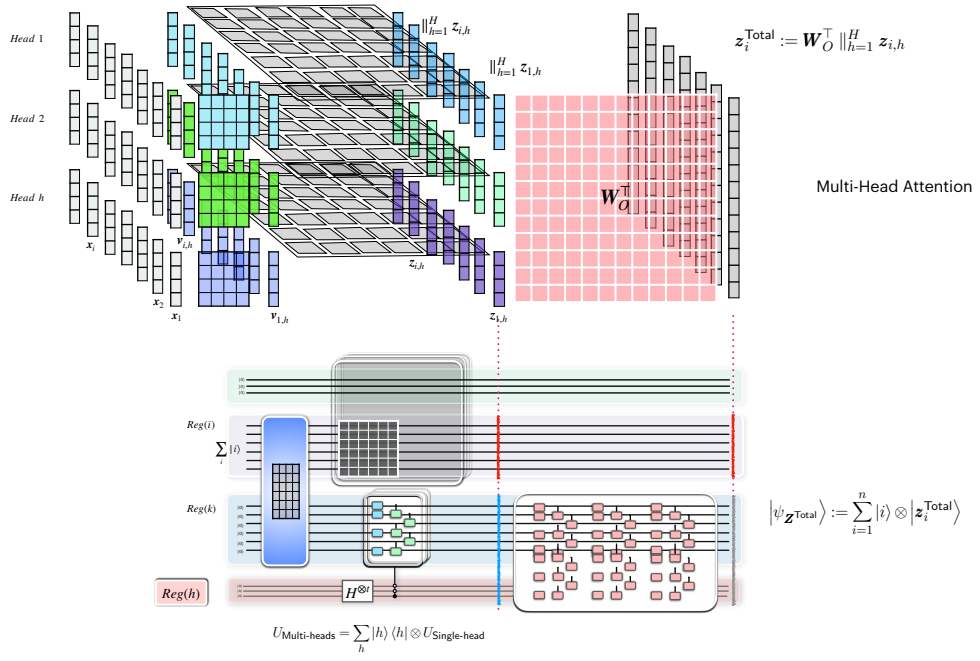

Figure 11.4: *Multihead-Attention on Quantum Computer* The upper part of the figure provides the illustration of classical Multihead-Attention, and the lower part of the figure depicts its quantum implementation. In the multihead attention module, We have H set of queries, values, and keys as:$\mathbb{R}^{p\times n} \ni \boldsymbol{Q}_h = \boldsymbol{W}_{Q,h}^\top \boldsymbol{X}, \quad \forall h \in \{1,\ldots,H\}, \mathbb{R}^{p\times n} \ni \boldsymbol{V}_h = \boldsymbol{W}_{V,h}^\top \boldsymbol{X}, \quad \forall h \in \{1,\ldots,H\}, \mathbb{R}^{r\times n} \ni \boldsymbol{K}_h = \boldsymbol{W}_{K,h}^\top \boldsymbol{X}, \quad \forall h \in \{1,\ldots,H\}$ Then, the scaled dot product attention are applied to generate the H output $\{\boldsymbol{Z}_h\}_{h=1}^H$ and $\boldsymbol{Z}_h = \left[\boldsymbol{z}_{1,h},\ldots,\boldsymbol{z}_{n,h}\right] \in \mathbb{R}^{r\times n}$. The outputs are concatenated over different heads as$\boldsymbol{Z}_{\text{Multi-heads}} = \left[\|_{h=1}^H \boldsymbol{z}_{1,h}, \|_{h=1}^H \boldsymbol{z}_{2,h},\ldots,\|_{h=1}^H \boldsymbol{z}_{n,h},\right] \in \mathbb{R}^{rH\times n}$, where $\|$ represents concatenation [17]. Then, by a linear projection $\boldsymbol{W}_O^\top$, the total attention value is obtained: $\boldsymbol{z}_i^{\text{Total}} := \boldsymbol{W}_O^\top \|_{h=1}^H \boldsymbol{z}_{i,h}, \boldsymbol{Z}^{\text{Total}} := \boldsymbol{W}_O^\top \boldsymbol{Z}_{\text{Multi-heads}}$ and $\boldsymbol{Z}^{\text{Total}} = \left[\boldsymbol{z}_1^{\text{Total}},\ldots,\boldsymbol{z}_n^{\text{Total}}\right] \in \mathbb{R}^{rH\times n}$. On the quantum circuit, the input encoding is represented by the blue box, as described in 11.1(here in this subsection and the following subsection 11.3.2, for simplicity we omit the positional encoding described in 11.2 in the presentation). The attention function can be implemented by applying the block-encoding of $\boldsymbol{A}^\top$ and a parameterized quantum circuit for $\boldsymbol{W}_V^\top$ on the two quantum registers $Reg(i)$ and $Reg(k)$ respectively.

where we define $U_{\text{Single-head}} = (U_{\boldsymbol{A}_h^\top} \otimes U_{\boldsymbol{W}_{V,h}^\top})$ in which for each head, $U_{\boldsymbol{A}_h^\top}$ is the block-encoding of $\boldsymbol{A}_h^\top$, $U_{\boldsymbol{W}_{V,h}^\top}$ is a parameterized quantum circuit implementing $\boldsymbol{W}_{V,h}^\top$.

For a simple head, from Eqn.11.30, when $U_{\text{Single-head}}$ acting on the state $|\psi_{\boldsymbol{X}}\rangle \otimes |0\rangle$, the outcome state is

$$U_{\text{Single-head}}(|\psi_{\boldsymbol{X}}\rangle \otimes |0\rangle) = \left|\psi_{\boldsymbol{Z}_h}\right\rangle \otimes |0\rangle + \ldots \tag{11.40}$$

where $\left|\psi_{\boldsymbol{Z}_h}\right\rangle := \sum_{i=1}^n |i\rangle \otimes \left|\boldsymbol{z}_{i,h}\right\rangle$ and $\left|\boldsymbol{z}_{i,h}\right\rangle$ is the amplitude encoding of the vector $\boldsymbol{z}_{i,h}$.

When $U_{\text{Multi-heads}}$ acting on the state prepared as $\sum_h |h\rangle \otimes (|\psi_{\boldsymbol{X}}\rangle \otimes |0\rangle)$ by the blue box and Hadamard gates on $Reg(h)$, the outcome state is

$$|\psi_{\text{Multi-heads}}\rangle = U_{\text{Multi-heads}} \sum_h |h\rangle \otimes (|\psi_{\boldsymbol{X}}\rangle \otimes |0\rangle) = \sum_h |h\rangle \otimes (\left|\psi_{\boldsymbol{Z}_h}\right\rangle \otimes |0\rangle + ...) \tag{11.41}$$

upon post-selecting, we obtain the state

$$|\psi_{\text{Multi-heads}}\rangle = \sum_h |h\rangle \otimes \left|\psi_{\boldsymbol{Z_h}}\right\rangle \tag{11.42}$$

$$= \sum_h |h\rangle \otimes \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{z}_{i,h}\right\rangle \tag{11.43}$$

$$= \sum_{i=1}^{n} |i\rangle \otimes \sum_h |h\rangle \left|\boldsymbol{z}_{i,h}\right\rangle \tag{11.44}$$

$$= \sum_{i=1}^{n} |i\rangle \otimes \left|\|_{h=1}^{H} \boldsymbol{z}_{i,h}\right\rangle \tag{11.45}$$

where $\left|\|_{h=1}^{H} \boldsymbol{z}_{i,h}\right\rangle$ is the amplitude encoding of the vector $\|_{h=1}^{H} \boldsymbol{z}_{i,h}$.

Applying a parameterized quantum circuit implementing $\boldsymbol{W}_O^\top$ on $Reg(k)$ and $Reg(h)$, which act as $U_{\boldsymbol{W}_O^\top} \left|\|_{h=1}^{H} \boldsymbol{z}_{i,h}\right\rangle = \left|\boldsymbol{z}_i^{\text{Total}}\right\rangle$, obtain

$$U_{\boldsymbol{W}_O^\top} |\psi_{\text{Multi-heads}}\rangle = \sum_{i=1}^{n} |i\rangle \otimes U_{\boldsymbol{W}_O^\top} \left|\|_{h=1}^{H} \boldsymbol{z}_{i,h}\right\rangle \tag{11.46}$$

$$= \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{z}_i^{\text{Total}}\right\rangle \tag{11.47}$$

$$= \left|\psi_{\boldsymbol{Z}^{\text{Total}}}\right\rangle \tag{11.48}$$

which is the desired state in Eqn.11.38.

11.4 Residual-connection on Quantum computer

After the multihead attention module, the data (containing positional encoding) $\boldsymbol{x}_i'$ and the total attention value $\boldsymbol{z}_i^{\text{Total}}$ are added (often referred as "residual-connection" introduced by ResNet [18]):

$$\boldsymbol{z}_i' := \boldsymbol{z}_i^{\text{Total}} + \text{concat}(\boldsymbol{x}_i', \boldsymbol{x}_i', ...\boldsymbol{x}_i') \in \mathbb{R}^{rH} \tag{11.49}$$

where $\text{concat}(\boldsymbol{x}_i', \boldsymbol{x}_i', ...\boldsymbol{x}_i')$ represents the concatenation of H identical vectors[4] $\boldsymbol{x}_i'$, For later usage, define $|\text{concat}(\boldsymbol{x}_i', \boldsymbol{x}_i', ...\boldsymbol{x}_i')\rangle$ as the the amplitude encoding of the vector $\text{concat}(\boldsymbol{x}_i', \boldsymbol{x}_i', ...\boldsymbol{x}_i')$ and $\boldsymbol{Z}' := [\boldsymbol{z}_1', \ldots, \boldsymbol{z}_n'] \in \mathbb{R}^{rH \times n}$.

[4]Note that this is a bit different from the standard classical residual connection.

On quantum circuit, we aim to obtain the following quantum state

$$\left|\psi_{\boldsymbol{Z'}}\right\rangle := \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{z}_i'\right\rangle \tag{11.50}$$

where $|\boldsymbol{z}_i'\rangle$ is the amplitude encoding of the vector $\boldsymbol{z}_i'$. This can be achieved via the quantum circuit depicted in Fig.11.5.

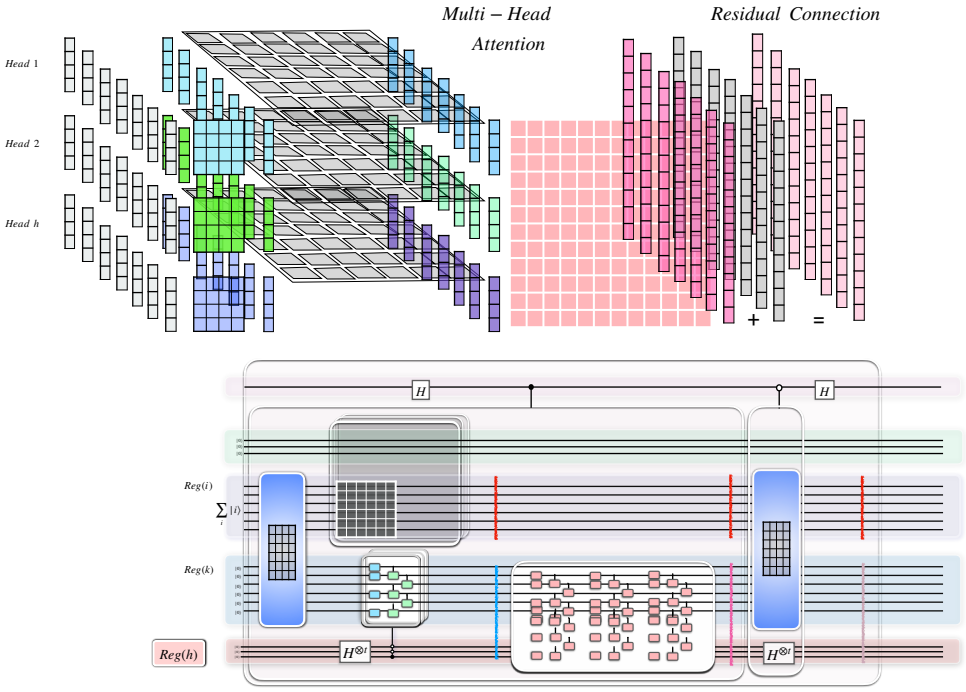

Figure 11.5: *Residual-connection on Quantum computer* The upper part of the figure provides the illustration of classical Multihead-Attention followed by Residual-connection, the lower part of the figure depicts their quantum implementation. After the multihead attention module, the data (containing positional encoding) $\boldsymbol{x}_i'$ and the total attention value $\boldsymbol{z}_i^{\mathsf{Total}}$ are added (often referred as "residual-connection" introduced by ResNet [18]):$\boldsymbol{z}_i' := \boldsymbol{z}_i^{\mathsf{Total}} + \text{concat}(\boldsymbol{x}_i', \boldsymbol{x}_i', ...\boldsymbol{x}_i') \in \mathbb{R}^{rH}$. The quantum circuit in this figure generates the following quantum state $\left|\psi_{\boldsymbol{Z'}}\right\rangle := \sum_{i=1}^{n} |i\rangle \otimes |\boldsymbol{z}_i'\rangle$ where $|\boldsymbol{z}_i'\rangle$ is the amplitude encoding of the vector $\boldsymbol{z}_i'$. The circuit implements Linear Combination of Unitaries of two operators grouped in the two transparent boxes controlled by the top ancillary qubit.

The first transparent box controlled by the top ancillary qubit in Fig.11.5 implements $U_{\boldsymbol{Z}^{\mathsf{Total}}}$ which acts as

$$U_{\boldsymbol{Z}^{\mathsf{Total}}} : |i\rangle \otimes |0\rangle \rightarrow |i\rangle \otimes \left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle \tag{11.51}$$

The blue box represents the data encoding (containing positional encoding) which acts as

$$U_{\boldsymbol{X'}} : |i\rangle \otimes |0\rangle \rightarrow |i\rangle \otimes \left|\boldsymbol{x}'\right\rangle \tag{11.52}$$

Including the Hadamard gates, the second transparent box controlled by the top ancillary qubit acts on the input state as

$$U_{\boldsymbol{X'}} \otimes H^{\otimes \log H} = \sum_{i=1}^{n} |i\rangle \otimes |0\rangle \otimes |0\rangle = \left|\psi_{\boldsymbol{X'}}\right\rangle \otimes \sum_{h} |h\rangle = \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{x'}_i\right\rangle \otimes \sum_{h} |h\rangle \tag{11.53}$$

The circuit in Fig.11.5 implements Linear Combination of two Unitaries as in the two transparent boxes controlled by the top ancillary qubit, it generates the state

$$\begin{aligned}
&\left|\psi_{\boldsymbol{Z}^{\mathsf{Total}}}\right\rangle + \left|\psi_{\boldsymbol{X'}}\right\rangle \otimes \sum_{h} |h\rangle \\
&\qquad = \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle + \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{x'}_i\right\rangle \otimes \sum_{h} |h\rangle \\
&\qquad = \sum_{i=1}^{n} |i\rangle \otimes \left(\left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle + \left|\boldsymbol{x'}_i\right\rangle \otimes \sum_{h} |h\rangle\right) \\
&\qquad = \sum_{i=1}^{n} |i\rangle \otimes \left(\left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle + \sum_{h} |h\rangle \left|\boldsymbol{x'}_i\right\rangle\right) \\
&\qquad = \sum_{i=1}^{n} |i\rangle \otimes \left(\left|\boldsymbol{z}_i^{\mathsf{Total}}\right\rangle + \left|\mathrm{concat}(\boldsymbol{x'}_i, \boldsymbol{x'}_i, ...\boldsymbol{x'}_i)\right\rangle\right) \\
&\qquad = \sum_{i=1}^{n} |i\rangle \otimes \left|\boldsymbol{z'}_i\right\rangle = \left|\psi_{\boldsymbol{Z'}}\right\rangle
\end{aligned}$$

which is the desired state in Eqn.11.50.

11.5 Feed-Forward Network on Quantum computer

After the multihead attention module and Residual-connection, a position-wise Feed-Forward Network(FFN) is applied[16]. The FFN is a fully connected feed-forward module that operates separately and identically on each $\boldsymbol{z'}_i$:

$$\mathrm{FFN}\left(\boldsymbol{z'}_i\right) = \mathbf{W}^{2\top}\,\mathrm{ReLU}\left(\mathbf{W}^{1\top}\boldsymbol{z'}_i + \mathbf{b}^1\right) + \mathbf{b}^2,$$

where $\mathbf{W}^1 \in \mathbb{R}^{rH \times d_{ff}}, \mathbf{W}^2 \in \mathbb{R}^{d_{ff} \times rH}, \mathbf{b}^1 \in \mathbb{R}^{d_{ff}}, \mathbf{b}^2 \in \mathbb{R}^{rH}$ are trainable parameters, d_{ff} is the intermediate dimension of the FFN. For simplicity we omit $\mathbf{b}^1, \mathbf{b}^2$ in the following discussion. Similar to we defined $\boldsymbol{X} := [\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n]$ before, we can write $\mathbf{W}^1 := \left[\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m, \ldots, \boldsymbol{w}_{d_{ff}}\right]$ where $\boldsymbol{w}_m \in \mathbb{R}^{rH}$.

Denote $\boldsymbol{y}_i := \mathbf{W}^{1\top}\boldsymbol{z'}_i \in \mathbb{R}^{d_{ff}}$, we have its elements as

$$\boldsymbol{y}_i^{(m)} = \boldsymbol{z'}_i \cdot \boldsymbol{w}_m, \forall m \in \{1, \ldots, d_{ff}\} \tag{11.54}$$

$\left|\psi_{\boldsymbol{Z'}}\right\rangle = \sum_{i=1}^{n} |i\rangle \otimes |\boldsymbol{z'}_i\rangle \otimes |0\rangle + ...$

by the unitary circled in the overall transparent box in Fig.11.5, denoted as $U_{\boldsymbol{Z'}}$, which act as

$$U_{\boldsymbol{Z'}} : |i\rangle\,|0\rangle_{k,h}\,|0\rangle_{\mathsf{other}} \to |i\rangle \left|\boldsymbol{z'}_i\right\rangle |0\rangle_{\mathsf{other}} + ..., \forall i \in \{1, \cdots n\}. \tag{11.55}$$

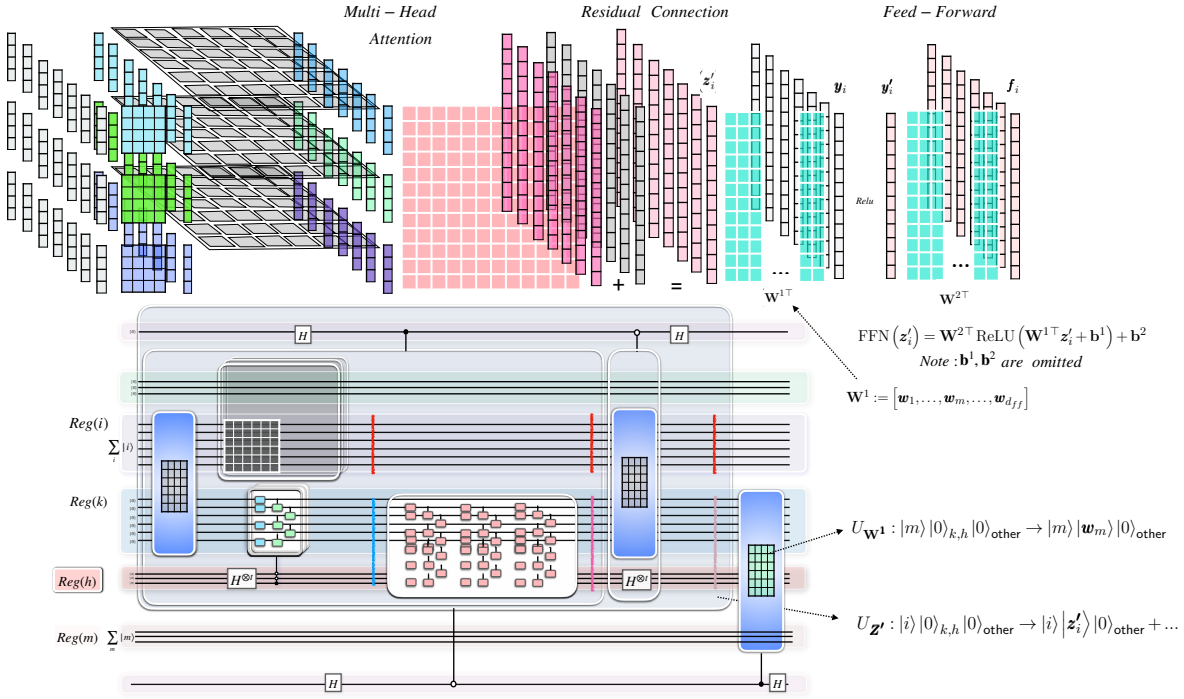

Figure 11.6: *Feed-Forward Network on Quantum computer 1* The upper part of the figure provides the illustration of classical Multihead-Attention followed by Residual-connection and Feed-Forward Network, and the lower part of the figure depicts their quantum implementation. After the multihead attention module and residual connection, a position-wise Feed-Forward Network(FFN) is applied. The FFN is a fully connected feed-forward module that operates separately and identically on each $\boldsymbol{z}_i'$: $\mathbf{W}^{2\top}\mathrm{ReLU}\left(\mathbf{W}^{1\top}\boldsymbol{z}_i' + \mathbf{b}^1\right) + \mathbf{b}^2$, we can write $\mathbf{W}^1 := \left[\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m, \ldots, \boldsymbol{w}_{d_{ff}}\right]$ where $\boldsymbol{w}_m \in \mathbb{R}^{rH}$,$d_{ff}$ is the intermediate dimension d_{ff} of the FFN. Denote $\boldsymbol{y}_i := \mathbf{W}^{1\top}\boldsymbol{z}_i'$, we have its elements as $\boldsymbol{y}_i^{(m)} = \boldsymbol{z}_i' \cdot \boldsymbol{w}_m$. Recall we created state on registered $Reg(i), Reg(k), Reg(h)$ and ancillas: $\left|\psi_{\boldsymbol{Z}'}\right\rangle = \sum_{i=1}^{n} |i\rangle \otimes |\boldsymbol{z}_i'\rangle \otimes |0\rangle + \ldots$, by the unitary circled in the overall transparent box in this figure, denoted as $U_{\boldsymbol{Z}'}$, which act as $U_{\boldsymbol{Z}'} : |i\rangle |0\rangle_{k,h} |0\rangle_{\text{other}} \to |i\rangle |\boldsymbol{z}_i'\rangle |0\rangle_{\text{other}} + \ldots, \forall i \in \{1, \cdots n\}$. For implementing $\mathbf{W}^1$, we can create a trainable unitary $U_{\mathbf{W}^1} : |m\rangle |0\rangle_{k,h} |0\rangle_{\text{other}} \to |m\rangle |\boldsymbol{w}_m\rangle |0\rangle_{\text{other}}, \forall m \in \{1, \ldots, d_{ff}\}$. with $|\boldsymbol{w}_m\rangle$ on $Reg(k), Reg(h)$ and $|m\rangle$ on an additional registered $Reg(m)$. $U_{\mathbf{W}^1}$, depicted as the blue box with a green centre in this figure, can be implemented as a series of controlled parameterised quantum circuits as $U_{\mathbf{W}^1} = \sum_m |m\rangle\langle m| U_m$ where each U_m, acting as $U_m : |0\rangle_{k,h} \to |\boldsymbol{w}_m\rangle$, is a parameterised quantum circuit.$\boldsymbol{y}_i^{(m)} = \langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle$ can be evaluated using Parallel Swap test for each $\boldsymbol{z}_i'$ and $\boldsymbol{w}_m$, via the quantum circuit depicted in this figure .

For implementing $\mathbf{W}^1 := \left[\boldsymbol{w}_1, \ldots, \boldsymbol{w}_m, \ldots, \boldsymbol{w}_{d_{ff}}\right]$, we can create a trainable unitary

$$U_{\mathbf{W}^1} : |m\rangle |0\rangle_{k,h} |0\rangle_{\text{other}} \to |m\rangle |\boldsymbol{w}_m\rangle |0\rangle_{\text{other}}, \forall m \in \{1, \ldots, d_{ff}\}. \tag{11.56}$$

with $|\boldsymbol{w}_m\rangle$ on $Reg(k), Reg(h)$ and $|m\rangle$ on an additional registered $Reg(m)$.

This trainable unitary $U_{\mathbf{W}^1}$ can be implemented as a series of controlled parameterised quantum circuits as $U_{\mathbf{W}^1} = \sum_m |m\rangle\langle m| U_m$ where each U_m, acting as $U_m : |0\rangle_{k,h} \to |\boldsymbol{w}_m\rangle$, is a parameterised quantum circuit.

Notice that

$$\boldsymbol{y}_i^{(m)} = \langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle, \forall m \in \{1, \ldots, d_{ff}\} \tag{11.57}$$

This can be evaluated using Parallel Swap test for each $\boldsymbol{z}_i' \in \mathbb{R}^{rH}$ and $\boldsymbol{w}_m \in \mathbb{R}^{rH}$, via the quantum circuit depicted in Fig.11.6.

The input state to the circuit is

$$|\Psi_0\rangle = \sum_i \sum_m |i\rangle|m\rangle \, |0\rangle_{k,h} \, |0\rangle_{\mathsf{other}} \, |0\rangle \tag{11.58}$$

For each branch $|i\rangle|m\rangle \, |0\rangle_{k,h} \, |0\rangle_{\mathsf{other}} \, |0\rangle$, applying a Hadamard gate on the bottom ancillary qubit, and controlled $U_{\boldsymbol{Z'}}$, $U_{\mathbf{W^1}}$ we obtain

$$|i\rangle|m\rangle(\left|\boldsymbol{z}_i'\right\rangle |0\rangle_{\mathsf{other}} + ...) \, |0\rangle + |i\rangle|m\rangle \, |\boldsymbol{w}_m\rangle \, |0\rangle_{\mathsf{other}} \, |1\rangle \tag{11.59}$$

Applying another Hadamard gate on the bottom ancillary qubit yield

$$|\psi_{im}\rangle = |i\rangle|m\rangle \left((\left|\boldsymbol{z}_i'\right\rangle |0\rangle_{\mathsf{other}} + ...) + |\boldsymbol{w}_m\rangle \, |0\rangle_{\mathsf{other}}\right) |0\rangle + |i\rangle|m\rangle \left((\left|\boldsymbol{z}_i'\right\rangle |0\rangle_{\mathsf{other}} + ...) - |\boldsymbol{w}_m\rangle \, |0\rangle_{\mathsf{other}}\right) |1\rangle \tag{11.60}$$

Denote $|u_{im}\rangle$ and $|v_{im}\rangle$ as the normalized states of $((|\boldsymbol{z}_i'\rangle \, |0\rangle_{\mathsf{other}} + ...) + |\boldsymbol{w}_m\rangle \, |0\rangle_{\mathsf{other}})$ and $((|\boldsymbol{z}_i'\rangle \, |0\rangle_{\mathsf{other}} + ...) - |\boldsymbol{w}_m\rangle \, |0\rangle_{\mathsf{other}})$ respectively. Then there is a real number θ_{im} such that

$$|\psi_{im}\rangle = |i\rangle|m\rangle \underbrace{(\sin\theta_{im} \, |u_{im}\rangle \, |0\rangle + \cos\theta_{im} \, |v_{im}\rangle \, |1\rangle)}_{|\phi_{im}\rangle} = |i\rangle|m\rangle \, |\phi_{im}\rangle \tag{11.61}$$

θ_{im} satisfies $\cos\theta_{im} = \sqrt{1 - \langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle} \, / \sqrt{2}$, $\sin\theta_{im} = \sqrt{1 + \langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle} \, / \sqrt{2}$, and we have:

$$\langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle = -\cos 2\theta_{im}. \tag{11.62}$$

To summarize, the quantum circuit depicted in Fig.11.6 , denoted as U, acts as

$$U : |i\rangle|m\rangle \, |0\rangle_{k,h} \, |0\rangle_{\mathsf{other}} \, |0\rangle \to |i\rangle|m\rangle \underbrace{(\sin\theta_{im} \, |u_{im}\rangle \, |0\rangle + \cos\theta_{im} \, |v_{im}\rangle \, |1\rangle)}_{|\phi_{im}\rangle} = |i\rangle|m\rangle \, |\phi_{im}\rangle \, , \tag{11.63}$$

where $\boldsymbol{y}_i^{(m)} = \langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle$ are encoded as:

$$\langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle = -\cos 2\theta_{im}. \tag{11.64}$$

When acting on the input state to the circuit $|\Psi_0\rangle = \sum_i \sum_m |i\rangle|m\rangle \, |0\rangle_{k,h} \, |0\rangle_{\mathsf{other}} \, |0\rangle$, U , also depicted as the pink box in Fig.11.7, produces the following state

$$|\Psi_1\rangle = \sum_i \sum_m |i\rangle|m\rangle \underbrace{(\sin\theta_{im} \, |u_{im}\rangle \, |0\rangle + \cos\theta_{im} \, |v_{im}\rangle \, |1\rangle)}_{|\phi_{im}\rangle} = \sum_i \sum_m |i\rangle|m\rangle \, |\phi_{im}\rangle \tag{11.65}$$

Next use amplitude estimation [90] to extract and store $\boldsymbol{y}_i^{(m)} = \langle \boldsymbol{z}_i' | \boldsymbol{w}_m \rangle$ into an additional register which we call the "amplitude register" $|0\rangle^t_{\textit{amplitude}}$ and the output state $|\Psi_1\rangle$ (using the same notation) becomes

$$|\Psi_3\rangle = \sum_i \sum_m |i\rangle|m\rangle \, |\phi_{im}\rangle \, |0\rangle^t_{\textit{amplitude}}, \tag{11.66}$$

where $|\phi_{im}\rangle$ can be decomposed as

$$|\phi_{im}\rangle = \frac{-i}{\sqrt{2}} \left(e^{i\theta_{im}} \, |\omega_+\rangle_{im} - e^{i(-\theta_{im})} \, |\omega_-\rangle_{im} \right). \tag{11.67}$$

where $|w_\pm\rangle_{im} = \frac{1}{\sqrt{2}}(|0\rangle|u_{im}\rangle \pm \boldsymbol{i}|1\rangle|v_{im}\rangle)$.

Hence, we have

$$|\Psi_1\rangle = \sum_i \sum_m \frac{-i}{\sqrt{2}} \left(e^{i\theta_{im}} |i\rangle \, |m\rangle \, |\omega_+\rangle_{im} - e^{i(-\theta_{im})} |i\rangle \, |m\rangle \, |\omega_-\rangle_{im} \right) |0\rangle^t_{\textit{amplitude}}. \tag{11.68}$$

The overall Grover operator G is defined as

$$G \coloneqq U C_2 U^{-1} C_1, \tag{11.69}$$

where C_1 is the Z gate on the bottom ancilla qubit in the pink box, and $C_2 = (I - 2|0\rangle\langle 0|) \otimes I_{i,m}$ is the "flip zero state" on registers other than $Reg(I), Reg(m)$(represented as S_0 in Fig.11.7).

Utilising Eqn. 11.63, it can be shown that G can be expressed as

$$G = \sum_i \sum_m |i\rangle \, |m\rangle \langle m| \, \langle i| \otimes G_{im}, \tag{11.70}$$

where G_{im} is defined as

$$G_{im} = (I - 2|\phi_{im}\rangle\langle\phi_{im}|))(Z \otimes I) \tag{11.71}$$

It is easy to check that $|w_\pm\rangle_{im}$ are the eigenstates of G_{im}, that is,

$$G_{im}|w_\pm\rangle_{im} = e^{\pm \boldsymbol{i} 2\theta_{im}} |w_\pm\rangle_{im}. \tag{11.72}$$

Therefore overall Grover operator G possess the following eigen-relation:

$$G \, |i\rangle \, |m\rangle \, |\omega_\pm\rangle_{im} = e^{i(\pm 2\theta_{im})} \, |i\rangle \, |m\rangle \, |\omega_\pm\rangle_{im}. \tag{11.73}$$

Next, we apply phase estimation of the overall Grover operator G on the input state $|\Psi_1\rangle$. The resulting state $|\Psi_2\rangle$ can be written as

$$|\Psi_2\rangle = \sum_i \sum_m \frac{-i}{\sqrt{2}} \left(e^{i\theta_{im}} \, |i\rangle \, |m\rangle \, |\omega_+\rangle_{im} \, |2\theta_{im}\rangle - e^{i(-\theta_{im})} \, |i\rangle \, |m\rangle \, |\omega_-\rangle_{im} \, |-2\theta_{im}\rangle \right). \tag{11.74}$$

Note here in Eq. 11.74, $|\pm 2\theta_{im}\rangle$ denotes the eigenvalues $\pm 2\theta_{im}$ being stored in the amplitude register with some finite precision.

Next we apply an oracle U_O on the amplitude register and an extra ancilla register $|0\rangle_{\mathsf{ReLU}}$, which acts as

$$U_O\,|0\rangle_{\mathsf{ReLU}}\,|\pm 2\theta_{im}\rangle = \left|\mathrm{ReLU}(\boldsymbol{y}_i^{(m)})\right\rangle |\pm 2\theta_{im}\rangle\,, \tag{11.75}$$

The state after the oracle can be written as

$$|\Psi_3\rangle = \sum_i \sum_m \frac{-i}{\sqrt{2}} \left|\mathrm{ReLU}(\boldsymbol{y}_i^{(m)})\right\rangle \left(e^{i\theta_{im}}\,|i\rangle\,|m\rangle\,|\omega_+\rangle_{im}\,|2\theta_{im}\rangle - e^{i(-\theta_{im})}\,|i\rangle\,|m\rangle\,|\omega_-\rangle_{im}\,|-2\theta_{im}\rangle\right). \tag{11.76}$$

Then we perform the uncomputation of Phase estimation, the resulting state is

$$|\Psi_4\rangle = \sum_i \sum_m \frac{-i}{\sqrt{2}} \left|\mathrm{ReLU}(\boldsymbol{y}_i^{(m)})\right\rangle \left(e^{i\theta_{im}}\,|i\rangle\,|m\rangle\,|\omega_+\rangle_{im}\,|0\rangle^t_{\mathit{amplitude}} - e^{i(-\theta_{im})}\,|i\rangle\,|m\rangle\,|\omega_-\rangle_{im}\,|0\rangle^t_{\mathit{amplitude}}\right) \tag{11.77}$$

$$= \sum_i \sum_m \left|\mathrm{ReLU}(\boldsymbol{y}_i^{(m)})\right\rangle |i\rangle|m\rangle\,|\phi_{im}\rangle\,|0\rangle^t_{\mathit{amplitude}} \tag{11.78}$$

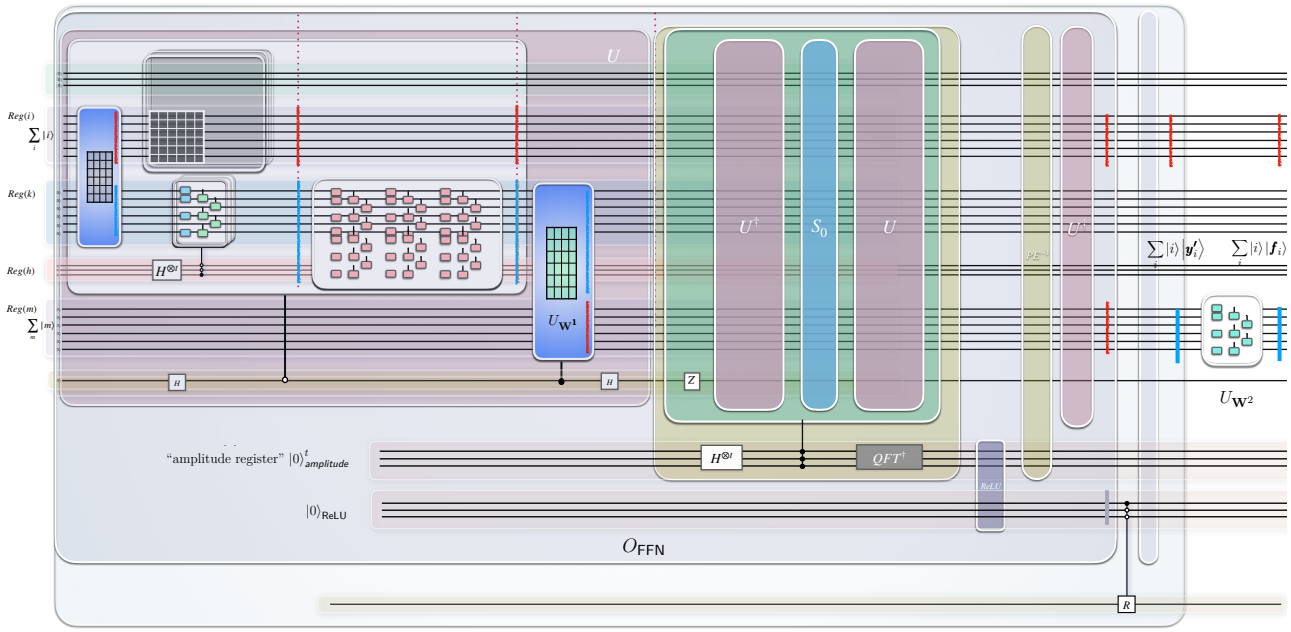

Figure 11.7: *Feed-Forward Network on Quantum computer 2* The figure illustrates the description from Eqn.11.63 to 11.83.The pink box in this figure, denoted as U, is meant to be the circuit in Fig.11.6, but for simplicity, we omitted the Residual-connection as in Fig.11.6, however the derivation follows the same.

Finally, we perform $U^\dagger$ and the resulting state is

$$|\Psi_5\rangle = \sum_i \sum_m \left|\mathrm{ReLU}(\boldsymbol{y}_i^{(m)})\right\rangle |i\rangle|m\rangle\,|0\rangle_{k,h}\,|0\rangle_{\mathsf{other}}\,|0\rangle\,|0\rangle^t_{\mathit{amplitude}}\,. \tag{11.79}$$

The above steps, as gathered in the grey box in Fig. 11.7, implement an oracle O_{FFN} such that:

$$O_{\mathsf{FFN}} : |i\rangle |m\rangle |0\rangle_{k,h} |0\rangle_{\mathsf{other}} |0\rangle |0\rangle_{\mathsf{ReLU}} \rightarrow |i\rangle |m\rangle |0\rangle_{k,h} |0\rangle_{\mathsf{other}} |0\rangle \left| \mathrm{ReLU}(\boldsymbol{y}_i^{(m)}) \right\rangle \tag{11.80}$$

Which produces the state

$$\sum_i \sum_m |i\rangle |m\rangle \left| \boldsymbol{y}_i'^{(m)} \right\rangle |0\rangle_{k,h} |0\rangle_{\mathsf{other}} |0\rangle \tag{11.81}$$

where we denote $\boldsymbol{y}_i'^{(m)} = \mathrm{ReLU}(\boldsymbol{y}_i^{(m)})$

Next, the "Conditional Rotation" (Theorem 3.5 in Ref. [13]) and uncomputation of O_{FFN} are applied, obtaining

$$\sum_i \sum_m |i\rangle \boldsymbol{y}_i'^{(m)} |m\rangle = \sum_i |i\rangle \sum_m \boldsymbol{y}_i'^{(m)} |m\rangle = \sum_i |i\rangle \left| \boldsymbol{y}_i' \right\rangle \tag{11.82}$$

where $|\boldsymbol{y}_i'\rangle := \sum_m \boldsymbol{y}_i'^{(m)} |m\rangle$ and we omitted the zero registers.

Finally, a trainable unitary $U_{\mathbf{W}^2}$ (a parameterised quantum circuit) implementing $\mathbf{W}^{2\top}$ is applied

$$|\Psi_{\boldsymbol{F}}\rangle = \sum_i |i\rangle U_{\mathbf{W}^2} \left| \boldsymbol{y}_i' \right\rangle = \sum_i |i\rangle |\boldsymbol{f}_i\rangle \tag{11.83}$$

where $\boldsymbol{f}_i := \mathbf{W}^{2\top} \boldsymbol{y}_i' = \mathrm{FFN}(\boldsymbol{z}_i') \in \mathbb{R}^{rH}$ and we define $\boldsymbol{F} := [\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n] \in \mathbb{R}^{rH \times n}$

By the quantum circuit in Fig. 11.7 we have obtained the final state from a Transformer block.

A tomography procedure (Theorem 4.3 from Ref.[13]) is performed to read out the amplitudes of the final state of a Transformer block. The results are then used as the input for the next Transformer block.

11.6 Generative Pre-training on Quantum Computer

Using the same notation as in the previous section, we denote the final state from the last Transformer block in GPT as:

$$|\Psi_{\boldsymbol{F}}\rangle = \sum_i |i\rangle |\boldsymbol{f}_i\rangle \tag{11.84}$$

where $\boldsymbol{f}_i \in \mathbb{R}^{rH}$ and $\boldsymbol{F} := [\boldsymbol{f}_1, \ldots, \boldsymbol{f}_n] \in \mathbb{R}^{rH \times n}$.

For language modeling, we then apply a linear layer $\boldsymbol{W}_E \in \mathbb{R}^{V \times rH}$ to map the output back to the vocabulary space as $\boldsymbol{f}_i' = \boldsymbol{W}_E \boldsymbol{f}_i \in \mathbb{R}^V$. $\boldsymbol{W}_E$ can be implemented on a quantum circuit by applying a trainable unitary $U_{\boldsymbol{W}_E}$ (a parameterised quantum circuit) on $Reg(k), Reg(m)$ and some extra qubits (since $V \gg rH$) and we obtain the state

$$\left|\Psi_{\boldsymbol{F'}}\right\rangle = U_{\boldsymbol{W}_E}(|\Psi_{\boldsymbol{F}}\rangle \otimes |0\rangle) = \sum_i |i\rangle\, U_{\boldsymbol{W}_E}(|\boldsymbol{f}_i\rangle \otimes |0\rangle) = \sum_i |i\rangle \left|\boldsymbol{f}'_i\right\rangle \tag{11.85}$$

where $\boldsymbol{F'} := \left[\boldsymbol{f}'_1, \ldots, \boldsymbol{f}'_n\right] \in \mathbb{R}^{V\times n}$.

For the tth batch in Generative Pre-training, we examine $\boldsymbol{f}'_{t+1}$ in the output, given the input $[\boldsymbol{x}_1, \ldots, \boldsymbol{x}_t]$. The loss function for this batch is defined as the cross-entropy between $\text{softmax}(\boldsymbol{f}'_{t+1})$ and the one-hot encoding of the $(t+1)$th token in the training text, denoted by $\boldsymbol{b}_{t+1} \in \mathbb{B}^V$. On the quantum circuit, the loss function can be correspondingly defined as the overlap between $\left|\boldsymbol{f}'_{t+1}\right\rangle$ and $|\boldsymbol{b}_{t+1}\rangle$[5], which can be evaluated via swap test on $\left|\Psi_{\boldsymbol{F'}}\right\rangle = \sum_i |i\rangle \left|\boldsymbol{f}'_i\right\rangle$ and an additional state $\left|\Psi_{\boldsymbol{b}_{t+1}}\right\rangle = |t+1\rangle\,|\boldsymbol{b}_{t+1}\rangle$. The cumulative loss is calculated across all batches, followed by the execution of certain optimization methods to adjust the model's parameters.

[5] $\left|\boldsymbol{f}'_{t+1}\right\rangle$ and $|\boldsymbol{b}_{t+1}\rangle$ represent the amplitude encoding of $\boldsymbol{f}'_{t+1}$ and $\boldsymbol{b}_{t+1}$.

Chapter 12

Conclusion

In this thesis, we have explored the frontier of Quantum Neural Networks (QNNs), delving into their architecture design, training methodologies, and practical applications. Our contributions span three core areas: the development of quantum-optimization-powered training methods, the design of QNNs tailored for graph-structured data, and the pioneering exploration of implementing GPT on quantum computers.

Our quantum training methods, leveraging quantum optimization algorithms, have shown promise in mitigating barren plateau issues and improving the efficiency of training QNNs. By exploiting hidden structures within QNN optimization problems, our framework demonstrates the potential for beyond-Grover speedup in quantum training.

For graph-structured data, we designed QNN architectures that incorporate inductive biases, aligning with classical Graph Neural Networks. Compared to their classical counterparts, these quantum architectures promise better scalability and expressivity. Compared to conventional problem-agnostic QNN, the number of parameters in our QNNs could be significantly reduced, improving trainability of the model.

Lastly, our exploration into the quantum implementation of GPT represents a significant step towards integrating large language models with quantum computing. This endeavour not only bridges the gap between quantum machine learning and state-of-the-art language models, but also sets the stage for future advancements in quantum-enhanced artificial intelligence.

Overall, this thesis contributes to the fundamental aspects of QNNs, offering novel insights and methodologies that pave the way for future research including: 1) Analysing the quantum advantages of our QNN architectures in detail, 2) Exploring the possibility of applying other quantum optimisation algorithms to QNN training. As we stand on the brink of a new era in computing, this research underscores the importance of continued exploration and innovation at

the intersection of quantum computing and machine learning.

Appendix A

Appendix

A.1 Implementation of the "selective copying" operation

In this section, we show that the "selective copying" operation can be implemented by a circuit with constant depth, as depicted in Fig.A.1.

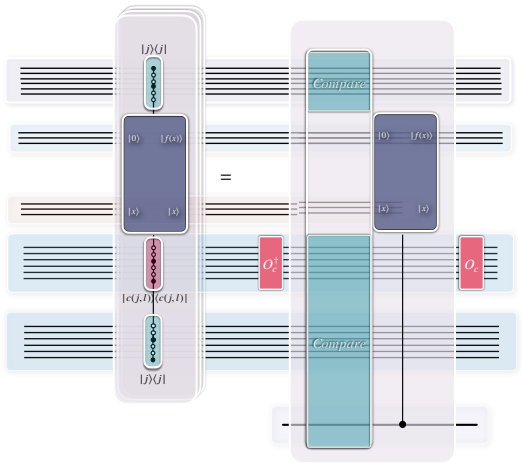

Figure A.1: The multiple multi-controlled unitaries for the "selective copying" can be implemented by a circuit with constant depth.

First, for each j, the multi-controlled unitaries can be rewritten as in Fig.A.2.

Then by piling up all the multi-controlled unitaries, we see that cancellation happens in the middle as in Fig.A.3 and we have the result depicted in Fig.A.4

The stack on the right side can be implemented by a comparing unitary followed by a controlled copy, as depicted in Fig.A.5.

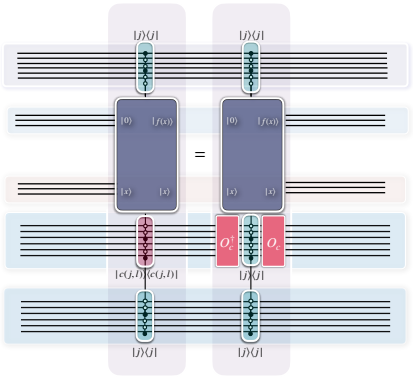

Figure A.2: For each j, the multi-controlled unitaries can be rewritten as in this figure.

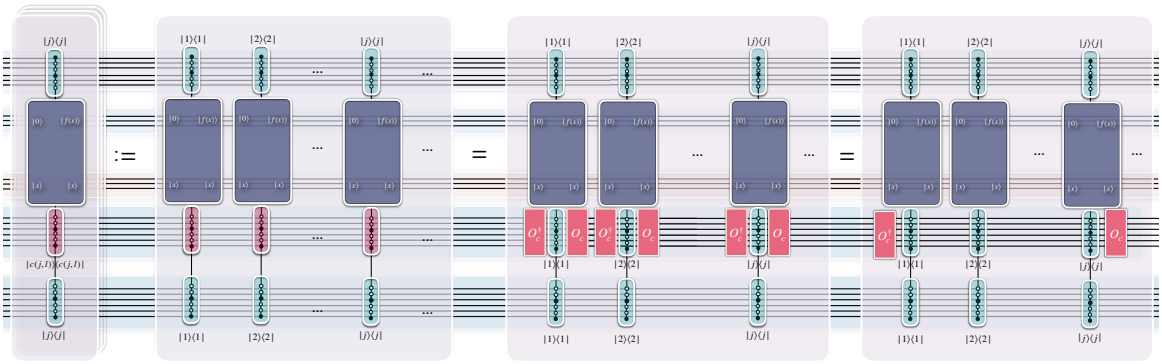

Figure A.3: By piling up all the multi-controlled unitaries we see that cancellation happens in the middle.

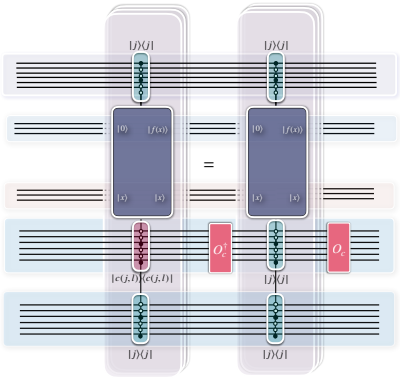

Figure A.4: Result of piling up all the multi-controlled unitaries, cancellation happens as in Fig.A.3

A.2 Proofs for the claims mentioned in Section 4.1

Here we provide proofs for the claims mentioned in Section 4.1

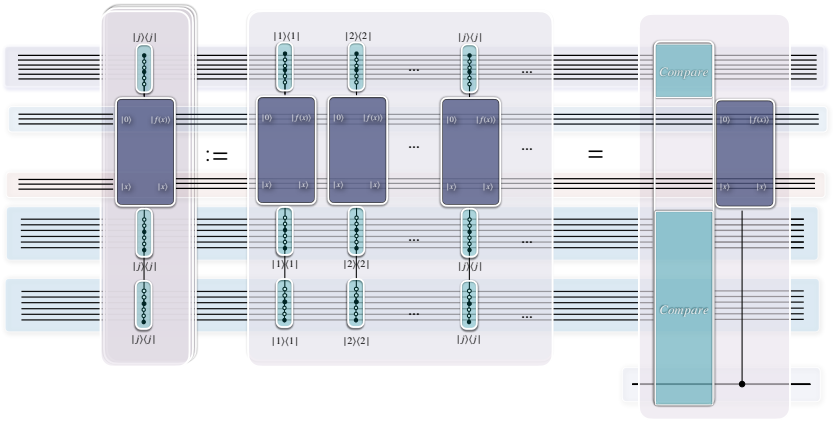

Figure A.5: The stack on the right side in Fig.A.4 can be implemented by a comparing unitary followed by a controlled copy

$$U = \sum_j |j\rangle\langle j| \otimes U_j$$

Proof.

$$U = [H \otimes I \otimes I \otimes I] \cdot [|0\rangle\langle 0| \otimes (\sum_j |j\rangle\langle j| \otimes P_j \otimes T) + |1\rangle\langle 1| \otimes (\sum_j |j\rangle\langle j| \otimes T \otimes P_j)] \cdot [H \otimes I \otimes I \otimes I] =$$

$$\sum_j [H \otimes I \otimes I \otimes I] \cdot [|0\rangle\langle 0| \otimes |j\rangle\langle j| \otimes P_j \otimes T + |1\rangle\langle 1| \otimes |j\rangle\langle j| \otimes T \otimes P_j] \cdot [H \otimes I \otimes I \otimes I] =$$

$$\sum_j [I \otimes H \otimes I \otimes I] \cdot [|j\rangle\langle j| \otimes |0\rangle\langle 0| \otimes P_j \otimes T + |j\rangle\langle j| \otimes |1\rangle\langle 1| \otimes T \otimes P_j] \cdot [I \otimes H \otimes I \otimes I] =$$

$$\sum_j [I \otimes H \otimes I \otimes I] \cdot [|j\rangle\langle j| \otimes (|0\rangle\langle 0| \otimes P_j \otimes T + |1\rangle\langle 1| \otimes T \otimes P_j)] \cdot [I \otimes H \otimes I \otimes I] =$$

$$\sum_j (I \cdot |j\rangle\langle j| \cdot I) \otimes ([H \otimes I \otimes I] \cdot [|0\rangle\langle 0| \otimes P_j \otimes T + |1\rangle\langle 1| \otimes T \otimes P_j] \cdot [H \otimes I \otimes I]) =$$

$$\sum_j |j\rangle\langle j| \otimes U_j$$

□

$$G = \sum_j |j\rangle\langle j| \otimes G_j$$

Proof.

$$G = C_1 U^{-1} C_2 U = \tag{A.1}$$

$$C_1 \left(\sum_j |j\rangle\langle j| \otimes U_j^\dagger \right) C_2 \left(\sum_k |k\rangle\langle k| \otimes U_k \right) = \tag{A.2}$$

$$\sum_j \sum_k C_1 \left(|j\rangle\langle j| \otimes U_j^\dagger \right) C_2 \left(|k\rangle\langle k| \otimes U_k \right) = \tag{A.3}$$

$$\sum_j |j\rangle\langle j| \otimes C_1 U_j^\dagger C_2 U_j = \tag{A.4}$$

$$\sum_j |j\rangle\langle j| \otimes G_j \tag{A.5}$$

□

A.3 Performance of quantum training using Grover Adaptive Search

It has been shown in Ref. [62] that Global Optimization by Grover Adaptive Search takes $O(\sqrt{N/s}\,)$ calls of Grover Oracle in which N is the dimension of the search space, s is the number of global optima and assuming s is small compared to N. A unique optimum will be found after $O(\log N)$ improvements on the threshold, in expectation. The number of measurements invoked between the improvements is no larger than $O(\log \sqrt{N}\,)$. Therefore the total number of measurements for Grover Adaptive Search to find a global optimum is no larger than $O(\log N \log \sqrt{N}\,)$.

Here we apply the above results to our QNN training problem. Taking training VQE as an example, we evaluate the number of "controlled-QNN" runs, the number of QNN runs, and the number of measurements respectively as follows.

A.3.1 Number of "controlled-QNN" runs

For our quantum training, each Grover iteration consists of the steps of Amplitude Encoding(AE), Phase estimation(PE), Threshold Oracle, and Uncomputation. Taking training VQE as an example, the number of the number of "controlled-QNN" runs of each step can be listed as:

- *In Amplitude Encoding(AE):* $n_{AE} = 1$
- *In Phase estimation(PE):* $n_{PE} = (2^t - 1)2 = 2^{t+1} - 2$
- *In Threshold Oracle:* 0

- *In Uncomputation:* $n_{PE}+n_{AE}$

in which t is the number of qubits in the amplitude register for the phase estimation.

The number of "controlled-QNN" runs of each Grover iteration, which we denote as N_0, is the sum of the above numbers:

$$N_0 = 2(n_{AE}+n_{PE}) = 2(2^{t+1}-1) \tag{A.6}$$

To obtain phase accurate to n' bits with probability of success at least $1-\epsilon_1$, t is chosen as

$$t = n' + \lceil \log(2+\frac{1}{2\epsilon_1}) \rceil. \tag{A.7}$$

Hence:

$$N_0 \approx (2^{n'+2}(2+\frac{1}{2\epsilon_1})-2) \tag{A.8}$$

For small ϵ_1 we have:

$$N_0 \approx 2^{n'+2}\frac{1}{2\epsilon_1} \tag{A.9}$$

n' determines the precision of the QNN cost function evaluated by phase estimation, which we denote as ϵ_2, and we have:

$$2^{-n'} = \epsilon_2 \tag{A.10}$$

therefore:

$$N_0 \approx \frac{2}{\epsilon_2\epsilon_1} \tag{A.11}$$

Therefore the total number of "controlled-QNN" runs for $O(\sqrt{N/s}\,)$ Grover iterations scales as

$$N_{\textit{controlled-QNN}} \sim O(\frac{1}{\epsilon_2\epsilon_1}\sqrt{N/s}\,) \tag{A.12}$$

in which N is the dimension of the parameter space of QNN and s is the number of global optima of the QNN cost function.

Let r be the number of parameters(rotation angles)in QNN, d be the number of control qubits for each rotation angle. Therefore

$$N = 2^{dr} \tag{A.13}$$

On the other hand,

$$2^{-d} = \epsilon_0, \tag{A.14}$$

where ϵ_0 is the precision of each angle value. Hence

$$N = (\frac{1}{\epsilon_0})^r \tag{A.15}$$

Inserting Eq. A.15 into A.12 we get

$$N_{\textit{controlled-QNN}} \sim O\left(\frac{1}{\epsilon_2\epsilon_1}\left(\frac{1}{\epsilon_0}\right)^{r/2} s^{-\frac{1}{2}}\right) \tag{A.16}$$

A.3.2 Number of Measurements

As mentioned before, the total number of measurements for Grover Adaptive Search to find a global optimum scale as

$$N_{Measurements} \sim O(\log N \log \sqrt{N}\,) \tag{A.17}$$

Inserting Eq. A.13 into A.17 we get

$$N_{Measurements} \sim O\left((dr)^{3/2}\right) \tag{A.18}$$

From Eq. A.14 we have $d \sim O\left(\log\left(\frac{1}{\epsilon_0}\right)\right)$, therefore

$$N_{Measurements} \sim O\left(\left(r \log\left(\frac{1}{\epsilon_0}\right)\right)^{1.5}\right) \tag{A.19}$$

A.3.3 Number of QNN runs

After each measurement on the parameter register, we obtain a specific parameter configuration of QNN. We then need to estimate the cost function for this particular parameter configuration. For VQE, the cost function is the expectation value of some Hamiltonian and the number of the estimation scale as $O(1/\epsilon^{\alpha})$ for some small power α [137](α is a small integer about 1 or 2), where ϵ is the desired accuracy of the expectation value. For our QNN training, we choose the accuracy ϵ to be ϵ_2 defined above. Taking $\alpha = 1$, the number of QNN runs after each measurement scale as $O(1/\epsilon_2)$ and the total number of QNN runs all the measurements during the quantum training scale as

$$N_{QNN} \sim O\left(\frac{1}{\epsilon_2}\right) N_{Measurements}. \tag{A.20}$$

Inserting A.19 into A.20 we have

$$N_{QNN} \sim O\left(\frac{1}{\epsilon_2}\left(r \log\left(\frac{1}{\epsilon_0}\right)\right)^{1.5}\right) \tag{A.21}$$

A.4 Number of qubits needed for Quantum training by AC-QAOA

Taking training VQE as an example, the number of qubits in each register can be listed as:

- *For QNN register*: n qubits
- *For Parameter register*: dr qubits (r is the number of parameters in QNN, d is the number of control qubits for each rotation angle.)
- *For Hadamard test*: 1 ancilla qubit

- *For LCU register and other registers*: $O(\log\log(1/\epsilon))$ [4] qubits (ϵ is the precision of implementing the phase oracle by LCU)

In total, the number of qubits needed for quantum training is:

$$n_{total} \sim n + dr + O(\log\log(1/\epsilon)) \tag{A.22}$$

For instance, when $n = 5$, $d = 5$, $r = 10$, $\epsilon = 10^{-8}$, $n_{total} \approx 60$.

Bibliography

[1] Michael M Bronstein, Joan Bruna, Taco Cohen, and Petar Veličković. Geometric deep learning: Grids, groups, graphs, geodesics, and gauges. *arXiv preprint arXiv:2104.13478*, 2021.

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

[3] Guillaume Verdon, Jason Pye, and Michael Broughton. A universal training algorithm for quantum deep learning. *arXiv preprint arXiv:1806.09729*, 2018.

[4] András Gilyén, Srinivasan Arunachalam, and Nathan Wiebe. Optimizing quantum optimization algorithms via faster quantum gradient computation. *Proceedings of the Thirtieth Annual ACM-SIAM Symposium on Discrete Algorithms*, page 1425–1444, Jan 2019.

[5] Linghua Zhu, Ho Lun Tang, George S Barron, Nicholas J Mayhall, Edwin Barnes, and Sophia E Economou. Adaptive quantum approximate optimization algorithm for solving combinatorial problems on a quantum computer. *Physical Review Research*, 4(3):033029, 2022.

[6] Dominic W. Berry, Andrew M. Childs, Richard Cleve, Robin Kothari, and Rolando D. Somma. Simulating hamiltonian dynamics with a truncated taylor series. *Phys. Rev. Lett.*, 114:090502, Mar 2015.

[7] András Gilyén, Yuan Su, Guang Hao Low, and Nathan Wiebe. Quantum singular value transformation and beyond: exponential improvements for quantum matrix arithmetics. *Proceedings of the 51st Annual ACM SIGACT Symposium on Theory of Computing - STOC 2019*, 2019.

[8] Andrew M Childs and Nathan Wiebe. Hamiltonian simulation using linear combinations of unitary operations. *arXiv preprint arXiv:1202.5822*, 2012.

[9] Pei Yuan and Shengyu Zhang. Optimal (controlled) quantum state preparation and improved unitary synthesis by quantum circuits with any number of ancillary qubits. *Quantum*, 7:956, 2023.

[10] Naixu Guo, Kosuke Mitarai, and Keisuke Fujii. Nonlinear transformation of complex amplitudes via quantum singular value transformation. *arXiv preprint arXiv:2107.10764*, 2021.

[11] Thomas N Kipf and Max Welling. Semi-supervised classification with graph convolutional networks. In *Proc. of ICLR*, 2017.

[12] Yidong Liao, Daniel Ebler, Feiyang Liu, and Oscar Dahlsten. Quantum speed-up in global optimization of binary neural nets. *New Journal of Physics*, 2020.

[13] Jonas Landman. Quantum algorithms for unsupervised machine learning and neural networks. *arXiv preprint arXiv:2111.03598*, 2021.

[14] Alec Radford, Karthik Narasimhan, Tim Salimans, Ilya Sutskever, et al. Improving language understanding by generative pre-training. 2018.

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

[16] Benyamin Ghojogh and Ali Ghodsi. Attention mechanism, transformers, bert, and gpt: tutorial and survey. *OSF Preprints*, 2020.

[17] Petar Velickovic, Guillem Cucurull, Arantxa Casanova, Adriana Romero, Pietro Lio, and Yoshua Bengio. Graph attention networks. In *Proc. of ICLR*, 2017.

[18] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Deep residual learning for image recognition. In *Proceedings of the IEEE conference on computer vision and pattern recognition*, pages 770–778, 2016.

[19] James G. Morley, Nicholas Chancellor, Sougato Bose, and Viv Kendon. Quantum search with hybrid adiabatic–quantum-walk algorithms and realistic noise. *Phys. Rev. A*, 99:022339, Feb 2019.

[20] Jiahao Yao, Lin Lin, and Marin Bukov. Reinforcement learning for many-body ground state preparation based on counter-diabatic driving, 2020.

[21] S. Hadfield, Z. Wang, B. O'Gorman, E. G. Rieffel, D. Venturelli, and R. Biswas. From the quantum approximate optimization algorithm to a quantum alternating operator ansatz. *Algorithms*, 12(2):34, Feb 2019.

[22] Ian Goodfellow, Yoshua Bengio, and Aaron Courville. *Deep Learning*. MIT Press, 2016.

[23] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. Deep learning. *nature*, 521(7553):436–444, 2015.

[24] Jacob Biamonte, Peter Wittek, Nicola Pancotti, Patrick Rebentrost, Nathan Wiebe, and Seth Lloyd. Quantum machine learning. *Nature*, 549(7671):195–202, Sep 2017.

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

[26] Hsin-Yuan Huang, Richard Kueng, Giacomo Torlai, Victor V. Albert, and John Preskill. Provably efficient machine learning for quantum many-body problems. *Science*, 377(6613):eabk3333, 2022.

[27] Vojtěch Havlíček, Antonio D Córcoles, Kristan Temme, Aram W Harrow, Abhinav Kandala, Jerry M Chow, and Jay M Gambetta. Supervised learning with quantum-enhanced feature spaces. *Nature*, 567(7747):209–212, 2019.

[28] M Cerezo, Guillaume Verdon, Hsin-Yuan Huang, Lukasz Cincio, and Patrick J Coles. Challenges and opportunities in quantum machine learning. *Nature Computational Science*, 2022.

[29] Jarrod R McClean, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Barren plateaus in quantum neural network training landscapes. *Nature Communications*, 9(1):1–6, 2018.

[30] Michael Ragone, Quynh T. Nguyen, Louis Schatzki, Paolo Braccia, Martin Larocca, Frederic Sauvage, Patrick J. Coles, and M. Cerezo. Representation theory for geometric quantum machine learning. *arXiv preprint arXiv:2210.07980*, 2022.

[31] Junyu Liu, Minzhao Liu, Jin-Peng Liu, Ziyu Ye, Yunfei Wang, Yuri Alexeev, Jens Eisert, and Liang Jiang. Towards provably efficient quantum algorithms for large-scale machine-learning models. *Nature Communications*, 15(1):434, 2024.

[32] Jarrod R McClean, Jonathan Romero, Ryan Babbush, and Alán Aspuru-Guzik. The theory of variational hybrid quantum-classical algorithms. *New Journal of Physics*, 18(2):023023, 2016.

[33] Robin Lorenz, Anna Pearson, Konstantinos Meichanetzidis, Dimitri Kartsaklis, and Bob Coecke. Qnlp in practice: Running compositional models of meaning on a quantum computer. *Journal of Artificial Intelligence Research*, 76:1305–1342, April 2023.

[34] Tiffany Duneau, Saskia Bruhn, Gabriel Matos, Tuomas Laakkonen, Katerina Saiti, Anna Pearson, Konstantinos Meichanetzidis, and Bob Coecke. Scalable and interpretable quantum natural language processing: an implementation on trapped ions, 2024.

[35] Tuomas Laakkonen, Konstantinos Meichanetzidis, and Bob Coecke. Quantum algorithms for compositional text processing. *Electronic Proceedings in Theoretical Computer Science*, 406:162–196, August 2024.

[36] Naixu Guo, Zhan Yu, Matthew Choi, Aman Agrawal, Kouhei Nakaji, Alán Aspuru-Guzik, and Patrick Rebentrost. Quantum linear algebra is all you need for transformer architectures. *arXiv preprint arXiv:2402.16714*, 2024.

[37] Ethan N Evans, Matthew Cook, Zachary P Bradshaw, and Margarite L LaBorde. Learning with sasquatch: a novel variational quantum transformer architecture with kernel-based self-attention. *arXiv preprint arXiv:2403.14753*, 2024.

[38] Martin Larocca, Nathan Ju, Diego García-Martín, Patrick J. Coles, and M. Cerezo. Theory of overparametrization in quantum neural networks. *Nature Computational Science*, 3(6):542—551, 2023.

[39] A. Kandala, A. Mezzacapo, K. Temme, M. Takita, M. Brink, J. M. Chow, and J. M. Gambetta. Hardware-efficient variational quantum eigensolver for small molecules and quantum magnets. *Nature*, 549(7671):242, 2017.

[40] Alexandre Choquette, Agustin Di Paolo, Panagiotis Kl Barkoutsos, David Sénéchal, Ivano Tavernelli, and Alexandre Blais. Quantum-optimal-control-inspired ansatz for variational quantum algorithms. *Physical Review Research*, 3(2):023092, 2021.

[41] Yudong Cao, Gian Giacomo Guerreschi, and Alán Aspuru-Guzik. Quantum neuron: an elementary building block for machine learning on quantum computers. *arXiv preprint arXiv:1711.11240*, 2017.

[42] Erik Torrontegui and Juan José García-Ripoll. Unitary quantum perceptron as efficient universal approximator. *EPL (Europhys. Lett.)*, 125(3):30004, 2019.

[43] Maria Schuld, Ilya Sinayskiy, and Francesco Petruccione. How to simulate a perceptron using quantum circuits. *Physics Letters A*, 379:660–663, 2015.

[44] Samuel Marsh and Jingbo Wang. A quantum walk assisted approximate algorithm for bounded np optimisation problems, 2018.

[45] Zhang Jiang, Eleanor G. Rieffel, and Zhihui Wang. Near-optimal quantum circuit for grover's unstructured search using a transverse field. *Physical Review A*, 95(6), Jun 2017.

[46] Glen Bigan Mbeng, Rosario Fazio, and Giuseppe Santoro. Quantum annealing: a journey through digitalization, control, and hybrid quantum variational schemes, 2019.

[47] Yan Wang. A quantum walk enhanced grover search algorithm for global optimization, 2017.

[48] D. Guéry-Odelin, A. Ruschhaupt, A. Kiely, E. Torrontegui, S. Martínez-Garaot, and J.G. Muga. Shortcuts to adiabaticity: Concepts, methods, and applications. *Reviews of Modern Physics*, 91(4), Oct 2019.

[49] Narendra N. Hegade, Koushik Paul, Yongcheng Ding, Mikel Sanz, F. Albarrán-Arriagada, Enrique Solano, and Xi Chen. Shortcuts to adiabaticity in digitized adiabatic quantum computing, 2020.

[50] James D. Whitfield, César A. Rodríguez-Rosario, and Alán Aspuru-Guzik. Quantum stochastic walks: A generalization of classical random walks and quantum walks. *Physical Review A*, 81(2), Feb 2010.

[51] David Bulger. Quantum basin hopping with gradient-based local optimisation. *arXiv e-prints*, pages quant–ph/0507193, July 2005.

[52] Edward Farhi, Jeffrey Goldstone, and Sam Gutmann. A quantum approximate optimization algorithm, 2014.

[53] Michael Streif and Martin Leib. Comparison of qaoa with quantum and simulated annealing, 2019.

[54] Murphy Yuezhen Niu, Sirui Lu, and Isaac L. Chuang. Optimizing qaoa: Success probability and runtime dependence on circuit depth, 2019.

[55] Panagiotis Kl. Barkoutsos, Giacomo Nannicini, Anton Robert, Ivano Tavernelli, and Stefan Woerner. Improving variational quantum optimization using cvar. *Quantum*, 4:256, Apr 2020.

[56] Mauro E. S. Morales, Jacob Biamonte, and Zoltán Zimborás. On the universality of the quantum approximate optimization algorithm, 2019.

[57] Seth Lloyd. Quantum approximate optimization is computationally universal. *arXiv preprint arXiv:1812.11075*, 2018.

[58] Sergey Bravyi, Alexander Kliesch, Robert Koenig, and Eugene Tang. Obstacles to state preparation and variational optimization from symmetry protection, 2019.

[59] Edward Farhi, Jeffrey Goldstone, Sam Gutmann, and Hartmut Neven. Quantum algorithms for fixed qubit architectures. *arXiv preprint arXiv:1703.06199*, 2017.

[60] Guillaume Verdon, Juan Miguel Arrazola, Kamil Brádler, and Nathan Killoran. A quantum approximate optimization algorithm for continuous problems, 2019.

[61] Christoph Durr and Peter Hoyer. A Quantum Algorithm for Finding the Minimum. *arXiv e-prints*, pages quant–ph/9607014, July 1996.

[62] W. P. Baritompa, D. W. Bulger, and G. R. Wood. Grovers quantum algorithm applied to global optimization. *SIAM Journal on Optimization*, 15(4):1170–1184, January 2005.

[63] Austin Gilliam, Stefan Woerner, and Constantin Gonciulea. Grover adaptive search for constrained polynomial binary optimization, 2020.

[64] Harry Buhrman, Richard Cleve, John Watrous, and Ronald de Wolf. Quantum fingerprinting. *Physical Review Letters*, 87(16), Sep 2001.

[65] Christoph Sünderhauf, Earl Campbell, and Joan Camps. Block-encoding structured matrices for data input in quantum computing. *Quantum*, 8:1226, 2024.

[66] Lin Lin. Lecture notes on quantum algorithms for scientific computation. *arXiv preprint arXiv:2201.08309*, 2022.

[67] A. Peruzzo, J. McClean, P. Shadbolt, M.-H. Yung, X.-Q. Zhou, P. J. Love, A. Aspuru-Guzik, and J. L. O'Brien. A variational eigenvalue solver on a photonic quantum processor. *Nature Communications*, 5:4213, 2014.

[68] Edward Farhi and Hartmut Neven. Classification with quantum neural networks on near term processors. *arXiv preprint arXiv:1802.06002*, 2018.

[69] Maria Schuld, Alex Bocharov, Krysta M. Svore, and Nathan Wiebe. Circuit-centric quantum classifiers. *Physical Review A*, 101(3), Mar 2020.

[70] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. Implementable quantum classifier for nonlinear data, 2018.

[71] Marcello Benedetti, Delfina Garcia-Pintos, Oscar Perdomo, Vicente Leyton-Ortega, Yunseong Nam, and Alejandro Perdomo-Ortiz. A generative modeling approach for benchmarking and training shallow quantum circuits. *npj Quantum Information*, 5(1), May 2019.

[72] Jinfeng Zeng, Yufeng Wu, Jin-Guo Liu, Lei Wang, and Jiangping Hu. Learning and inference on generative adversarial quantum circuits. *Physical Review A*, 99(5), May 2019.

[73] Kathleen E. Hamilton, Eugene F. Dumitrescu, and Raphael C. Pooser. Generative model benchmarks for superconducting qubits. *Physical Review A*, 99(6), Jun 2019.

[74] Kosuke Mitarai, Makoto Negoro, Masahiro Kitagawa, and Keisuke Fujii. Quantum circuit learning. *Physical Review A*, 98(3):032309, 2018.

[75] He-Liang Huang, Yuxuan Du, Ming Gong, Youwei Zhao, Yulin Wu, Chaoyue Wang, Shaowei Li, Futian Liang, Jin Lin, Yu Xu, Rui Yang, Tongliang Liu, Min-Hsiu Hsieh, Hui Deng, Hao Rong, Cheng-Zhi Peng, Chao-Yang Lu, Yu-Ao Chen, Dacheng Tao, Xiaobo

Zhu, and Jian-Wei Pan. Experimental quantum generative adversarial networks for image generation, 2020.

[76] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, and Dacheng Tao. The expressive power of parameterized quantum circuits. *arXiv preprint arXiv:1810.11922*, 2018.

[77] Andrea Skolik, Jarrod R McClean, Masoud Mohseni, Patrick van der Smagt, and Martin Leib. Layerwise learning for quantum neural networks. *Quantum Machine Intelligence*, 3(1):1–11, 2021.

[78] M. Cerezo, Akira Sone, Tyler Volkoff, Lukasz Cincio, and Patrick J Coles. Cost function dependent barren plateaus in shallow parametrized quantum circuits. *Nature Communications*, 12(1):1–12, 2021.

[79] Tyler Volkoff and Patrick J Coles. Large gradients via correlation in random parameterized quantum circuits. *arXiv preprint arXiv:2005.12200*, 2020.

[80] Guillaume Verdon, Michael Broughton, Jarrod R McClean, Kevin J Sung, Ryan Babbush, Zhang Jiang, Hartmut Neven, and Masoud Mohseni. Learning to learn with quantum neural networks via classical neural networks. *arXiv preprint arXiv:1907.05415*, 2019.

[81] Yuxuan Du, Min-Hsiu Hsieh, Tongliang Liu, Shan You, and Dacheng Tao. On the learnability of quantum neural networks. *PRX Quantum*, 2(4):040337, 2021.

[82] Yuxuan Du, Tao Huang, Shan You, Min-Hsiu Hsieh, and Dacheng Tao. Quantum circuit architecture search: error mitigation and trainability enhancement for variational quantum solvers. *npj Quantum Information*, 8(1):62, 2022.

[83] Kaining Zhang, Min-Hsiu Hsieh, Liu Liu, and Dacheng Tao. Toward trainability of quantum neural networks, 2020.

[84] Samson Wang, Enrico Fontana, M. Cerezo, Kunal Sharma, Akira Sone, Lukasz Cincio, and Patrick J. Coles. Noise-induced barren plateaus in variational quantum algorithms, 2020.

[85] Ernesto Campos, Aly Nasrallah, and Jacob Biamonte. Abrupt transitions in variational quantum circuit training, 2020.

[86] Carlos Ortiz Marrero, Mária Kieferová, and Nathan Wiebe. Entanglement-induced barren plateaus. *PRX Quantum*, 2(4):040316, 2021.

[87] Andrew Arrasmith, M. Cerezo, Piotr Czarnik, Lukasz Cincio, and Patrick J Coles. Effect of barren plateaus on gradient-free optimization. *Quantum*, 5:558, 2021.

[88] Matteo M. Wauters, Emanuele Panizon, Glen B. Mbeng, and Giuseppe E. Santoro. Reinforcement-learning-assisted quantum optimization. *Phys. Rev. Research*, 2:033446, Sep 2020.

[89] A. Warren, L. Zhu, H. L. Tang, Khadijeh Najafi, Edwin Barnes, and S. Economou. Rnn-vqe: a machine learning approach to generating variational ansatze. *Bulletin of the American Physical Society*, 2020.

[90] Gilles Brassard, Peter Hoyer, Michele Mosca, and Alain Tapp. Quantum Amplitude Amplification and Estimation. *arXiv e-prints*, pages quant–ph/0005055, May 2000.

[91] Jarrod R McClean, Matthew P Harrigan, Masoud Mohseni, Nicholas C Rubin, Zhang Jiang, Sergio Boixo, Vadim N Smelyanskiy, Ryan Babbush, and Hartmut Neven. Low depth mechanisms for quantum optimization. *arXiv preprint arXiv:2008.08615*, 2020.

[92] Christian Weedbrook, Stefano Pirandola, Raúl García-Patrón, Nicolas J. Cerf, Timothy C. Ralph, Jeffrey H. Shapiro, and Seth Lloyd. Gaussian quantum information. *Rev. Mod. Phys.*, 84:621–669, May 2012.

[93] Stephen D. Bartlett, Barry C. Sanders, Samuel L. Braunstein, and Kae Nemoto. Efficient classical simulation of continuous variable quantum information processes. *Phys. Rev. Lett.*, 88:097904, Feb 2002.

[94] Sami Khairy, Ruslan Shaydulin, Lukasz Cincio, Yuri Alexeev, and Prasanna Balaprakash. Reinforcement-learning-based variational quantum circuits optimization for combinatorial problems, 2019.

[95] Jiahao Yao, Marin Bukov, and Lin Lin. Policy gradient based quantum approximate optimization algorithm, 2020.

[96] Kyle Poland, Kerstin Beer, and Tobias J Osborne. No free lunch for quantum machine learning. *arXiv preprint arXiv:2003.14103*, 2020.

[97] Zonghan Wu, Shirui Pan, Fengwen Chen, Guodong Long, Chengqi Zhang, and S Yu Philip. A comprehensive survey on graph neural networks. *IEEE transactions on neural networks and learning systems*, 2020.

[98] Michaël Defferrard, Xavier Bresson, and Pierre Vandergheynst. Convolutional neural networks on graphs with fast localized spectral filtering. In *Proc. of NIPS*, pages 3844–3852, 2016.

[99] Justin Gilmer, Samuel S Schoenholz, Patrick F Riley, Oriol Vinyals, and George E Dahl. Neural message passing for quantum chemistry. In *International conference on machine learning*, pages 1263–1272. PMLR, 2017.

[100] Guillaume Verdon, Trevor McCourt, Enxhell Luzhnica, Vikash Singh, Stefan Leichenauer, and Jack Hidary. Quantum graph neural networks. *arXiv preprint arXiv:1909.12264*, 2019.

[101] Yidong Liao, Min-Hsiu Hsieh, and Chris Ferrie. Quantum optimization for training quantum neural networks. *arXiv preprint arXiv:2103.17047*, 2021.

[102] Kerstin Beer, Megha Khosla, Julius Köhler, and Tobias J Osborne. Quantum machine learning of graph-structured data. *arXiv preprint arXiv:2103.10837*, 2021.

[103] Andrea Skolik, Michele Cattelan, Sheir Yarkoni, Thomas Bäck, and Vedran Dunjko. Equivariant quantum circuits for learning on weighted graphs. *npj Quantum Information*, 9(1):47, 2023.

[104] Xing Ai, Zhihong Zhang, Luzhe Sun, Junchi Yan, and Edwin Hancock. Decompositional quantum graph neural network. *arXiv preprint arXiv:2201.05158*, 2022.

[105] Chaitanya K. Joshi. Recent advances in efficient and scalable graph neural networks. *chaitjo.com*, 2022.

[106] Cenk Tüysüz, Carla Rieger, Kristiane Novotny, Bilge Demirköz, Daniel Dobos, Karolos Potamianos, Sofia Vallecorsa, Jean-Roch Vlimant, and Richard Forster. Hybrid quantum classical graph neural networks for particle track reconstruction. *Quantum Machine Intelligence*, 3(2):1–20, 2021.

[107] Péter Mernyei, Konstantinos Meichanetzidis, and Ismail Ilkan Ceylan. Equivariant quantum graph circuits. In *International Conference on Machine Learning*, pages 15401–15420. PMLR, 2022.

[108] Zhirui Hu, Jinyang Li, Zhenyu Pan, Shanglin Zhou, Lei Yang, Caiwen Ding, Omer Khan, Tong Geng, and Weiwen Jiang. On the design of quantum graph convolutional neural network in the nisq-era and beyond. In *2022 IEEE 40th International Conference on Computer Design (ICCD)*, pages 290–297. IEEE, 2022.

[109] Jin Zheng, Qing Gao, and Yanxuan Lü. Quantum graph convolutional neural networks. *arXiv preprint arXiv:2107.03257*, 2021.

[110] Yanhu Chen, Cen Wang, Hongxiang Guo, and Wu Jian. Novel architecture of parameterized quantum circuit for graph convolutional network. *arXiv preprint arXiv:2203.03251*, 2022.

[111] Muhammet Balcilar, Guillaume Renton, Pierre Héroux, Benoit Gauzere, Sebastien Adam, and Paul Honeine. Bridging the gap between spectral and spatial domains in graph neural networks. *arXiv preprint arXiv:2003.11702*, 2020.

[112] Maria Schuld and Francesco Petruccione. *Machine learning with quantum computers*. Springer.

[113] Gui-Lu Long and Yang Sun. Efficient scheme for initializing a quantum register with an arbitrary superposed state. *Phys. Rev. A*, 64:014303, Jun 2001.

[114] Lov Grover and Terry Rudolph. Creating superpositions that correspond to efficiently integrable probability distributions. *arXiv:quant-ph/0208112*, 2002.

[115] Mikko Möttönen, Juha J Vartiainen, Ville Bergholm, and Martti M Salomaa. Transformation of quantum states using uniformly controlled rotations. *Quantum. Inf. Comput.*, 5(6):467–473, 2005.

[116] Martin Plesch and Časlav Brukner. Quantum-state preparation with universal gate decompositions. *Phy. Rev. A*, 83(3):032302, 2011.

[117] Xiao-Ming Zhang, Man-Hong Yung, and Xiao Yuan. Low-depth quantum state preparation. *Phys. Rev. Res.*, 3:043200, Dec 2021.

[118] Xiaoming Sun, Guojing Tian, Shuai Yang, Pei Yuan, and Shengyu Zhang. Asymptotically optimal circuit depth for quantum state preparation and general unitary synthesis. *arXiv:2108.06150v2*, 2021.

[119] Gregory Rosenthal. Query and depth upper bounds for quantum unitaries via Grover search. *arXiv:2111.07992*, 2021.

[120] Xiao-Ming Zhang, Tongyang Li, and Xiao Yuan. Quantum state preparation with optimal circuit depth: Implementations and applications. *Physical Review Letters*, 129(23):230504, 2022.

[121] B David Clader, Alexander M Dalzell, Nikitas Stamatopoulos, Grant Salton, Mario Berta, and William J Zeng. Quantum resources required to block-encode a matrix of classical data. *arXiv:2206.03505*, 2022.

[122] Kaiwen Gui, Alexander M Dalzell, Alessandro Achille, Martin Suchara, and Frederic T Chong. Spacetime-efficient low-depth quantum state preparation with applications. *Quantum*, 8:1257, 2024.

[123] Xiao-Ming Zhang and Xiao Yuan. Circuit complexity of quantum access models for encoding classical data. *npj Quantum Information*, 10(1):42, 2024.

[124] Felix Wu, Amauri Souza, Tianyi Zhang, Christopher Fifty, Tao Yu, and Kilian Weinberger. Simplifying graph convolutional networks. In *International conference on machine learning*, pages 6861–6871. PMLR, 2019.

[125] J. Knörzer, D. Malz, and J. I. Cirac. Cross-platform verification in quantum networks. *Phys. Rev. A*, 107:062424, Jun 2023.

[126] Alessandro Luongo. Quantum algorithms for data analysis. Quantum Algorithms, 2023.

[127] Stavros P Adam, Stamatios-Aggelos N Alexandropoulos, Panos M Pardalos, and Michael N Vrahatis. No free lunch theorem: a review. In *Approximation and Optimization*, pages 57–82. Springer, 2019.

[128] Larry Medsker and Lakhmi C Jain. *Recurrent neural networks: design and applications*. CRC press, 1999.

[129] Sepp Hochreiter and Jürgen Schmidhuber. Long short-term memory. *Neural computation*, 9(8):1735–1780, 1997.

[130] Tomas Mikolov, Kai Chen, Greg Corrado, and Jeff Dean. Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*, 2013.

[131] Jeffrey Pennington, Richard Socher, and Christopher D Manning. Glove: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, 2014.

[132] Yoav Goldberg and Omer Levy. word2vec explained: deriving mikolov et al.'s negative-sampling word-embedding method. *arXiv preprint arXiv:1402.3722*, 2014.

[133] Tomas Mikolov, Quoc V Le, and Ilya Sutskever. Computing numeric representations of words in a high-dimensional space. In *Machine Learning and Knowledge Discovery in Databases*, pages 522–536. Springer, 2015.

[134] Jiachen Lu, Jinghan Yao, Junge Zhang, Xiatian Zhu, Hang Xu, Weiguo Gao, Chunjing Xu, Tao Xiang, and Li Zhang. Soft: Softmax-free transformer with linear complexity. *Advances in Neural Information Processing Systems*, 34:21297–21309, 2021.

[135] Zhen Qin, Weixuan Sun, Hui Deng, Dongxu Li, Yunshen Wei, Baohong Lv, Junjie Yan, Lingpeng Kong, and Yiran Zhong. cosformer: Rethinking softmax in attention. *arXiv preprint arXiv:2202.08791*, 2022.

[136] Quynh T Nguyen, Bobak T Kiani, and Seth Lloyd. Block-encoding dense and full-rank kernels using hierarchical matrices: applications in quantum numerical linear algebra. *Quantum*, 6:876, 2022.

[137] Emanuel Knill, Gerardo Ortiz, and Rolando D. Somma. Optimal quantum measurements of expectation values of observables. *Phys. Rev. A*, 75:012328, Jan 2007.